Blockchain: Research and Applications 3 (2022) 100031

Contents lists available at ScienceDirect

Blockchain: Research and Applications

journal homepage: www.journals.elsevier.com/blockchain-research-and-applications

A scalable decentralized system for fair token distribution and seamless users onboarding

Francesco Bruschi [a,*], Manuel Tumiati [b], Vincenzo Rana [a], Mattia Bianchi [b], Donatella Sciuto [a]

[a] Politecnico di Milano, Milano, 20133, Italy
[b] KNOBS srl, Milan, 20121, Italy

ARTICLE INFO

Keywords:
Blockchain
Tokens
Onboarding
Wallets

ABSTRACT

Tokens are digital, transferable, and programmable assets and one of the most promising tools offered by blockchains. They could enable a wide range of applications, from down to earth to futuristic. One of the main issues in achieving wide adoption of tokens is onboarding: main platforms require users to deal with specific tools such as wallets, transaction fees, key generation, and storage.

The most common solution to offer a familiar experience to naive users are custodial intermediaries, which have the important drawback of centralizing the process, and keeping users away from the advantages of self-sovereign assets control.

In this paper, we present a process for the distribution of digital tokens to end users, exploiting "physical" objects for initial distribution. The process is aimed at making the onboarding of users not yet accustomed to blockchain tools and concepts as easy as possible, in a secure and decentralized way.

1. Introduction

One of the possibilities introduced by the blockchain technology is to represent digital assets that can be easily and securely owned through cryptographic identifiers that can be generated autonomously by users. The assets can be seamlessly transferred, and the logic and mechanics of the transactions can be programmed reliably and transparently. Bitcoin [1] itself, the system that historically introduced blockchains, can be seen as a platform for the storage, ownership, and programmable transfer of a certain value token, the bitcoin. If we focus on the bitcoin token, we observe some peculiar features:

- it has a capped supply;
- it is minted/distributed with a predefined, deterministic mechanism (as a reward for miners);
- it has one basic function (can be used to pay transaction fees in the network);

The concept then underwent a process of abstraction, and frameworks were conceived to design and develop other kinds of tokens, with different features. Omnilayer [2], one of the first programmable assets frameworks, provided a layer on top of bitcoin. Upon it, one of the first (and as of writing, still the most capitalized) stablecoin was built, Tether [3]. In 2012, Ethereum [4] introduced a fully programmable (Turing complete) blockchain. From the very beginning, the definition of a token with custom features was one of the motivating applications proposed for Ethereum. The idea stuck, and two standard abstract interfaces, ERC20 [5] and ERC721 [6], were defined to maximize adoption and interoperability. Since then, a Cambrian era of new tokens, services, securities, and utilities began, in which infamous scams were conceived along with very successful projects. A basic and useful partition of tokens is between fungible and non fungible. Fungible tokens are those that are interchangeable, such as dollar bills. Non fungible tokens, in turn, have meaningfully individual features, such as collectibles, or tickets for an event, etc. Fungible tokens can be used to represent, for example, monetary-like value or access to services (in the latter case, they are called utility tokens). Sometimes, like in the cases of ether (the token with which computation is paid for in Ethereum) and bitcoin, the same token can both serve as a utility and as a means of value exchange and storage. Some applications of non fungible tokens include representation of ownership of physical/virtual assets/objects, collectibles, etc.

* Corresponding author.
E-mail addresses: francesco.bruschi@polimi.it (F. Bruschi), manuel.tumiati@knobs.it (M. Tumiati), vincenzo.rana@polimi.it (V. Rana), mattia.bianchi@knobs.it (M. Bianchi), donatella.sciuto@polimi.it (D. Sciuto).

https://doi.org/10.1016/j.bcra.2021.100031
Received 30 December 2020; Received in revised form 16 September 2021; Accepted 22 September 2021

In most applications, the initial distribution of tokens is key. The best token distribution strategy varies with the economic model underlying the token (sometimes called token economy). Typical distribution strategies include Initial Coin Offerings (ICOs), in which tokens are put on time-limited sales, sometimes with auction mechanisms and airdrops, in which tokens are distributed for free to a set of users. We will review most of these strategies in Section 2. Another crucial dimension is the type of ownership control that users are given. While in some cases, tokens are aimed at users with some confidence with blockchain tools, other times it is desirable to engage "crypto-naive" users. This can pose challenges such as how to make the user pay the fees, and how to generate and keep the keys. Currently, custodial ownership architectures in which an intermediary keeps the keys and pays for the fees, providing a "classical" interface (user login), are the most considered option. Custodial models, in spite of their friendliness, have many drawbacks, among which the "recentralization" of the system, the natural counterparty risk, and the introduction of more vulnerable moving parts in the system. Last but not least, from a "pedagogical" point of view, they do not allow the users to get accustomed to the features of blockchain assets ownership.

In this paper, we propose a system that aims at distributing tokens by means of QR codes, and that does not require any software tools nor previous acquaintance with any blockchain or crypto concepts. The process is non custodial and completely decentralized. We then show an implementation of the system, and show performance and cost figures taken in a real-world usage context, along with a deep analysis of the scalability of the proposed approach, in terms of both costs and performance.

This article is an extended version of Ref. [7]. With respect to the original paper, we analyze the performance limits of the proposed system (latency, throughput, cost), and examine and review the main tools and technologies available to improve scalability. After analyzing the different options, we focus on optimistic rollups, and we experiment using them to improve scalability. We then describe and analyze the results obtained.

2. State of the art

There are different ways to distribute new tokens. In some cases, tokens have a tangible value since their inception, and can then be sold in exchange for other values (fiat, or other value tokens). Other times, tokens have no initial value (that is, no buyers are willing to pay value for them), but have a role in bootstrapping the system. In some other cases, tokens are non fungible representations of some assets that you want to distribute (e.g., collectibles). In this section, we review the state of the art of token distribution mechanisms and strategies.

2.1. Systems for the distribution of tokens

In Ref. [8], the authors review and analyze the main strategies for token distribution mechanisms at genesis, that is, at the inception of the system, be it a new blockchain or an application living on top of an existing one. The review mainly focuses on fungible tokens, and it acknowledges that token sales are the most frequent strategies. There are different types though, including:

- Initial Coin Offerings (ICOs), in which tokens are sold, typically via smart contracts, and paid in value tokens (e.g. ether);
- Initial Exchange Offerings (IEOs), similar to ICOs, except that the tokens are put on sale on custodial exchange platforms, and thus involve the role of an actor subject to some jurisdiction;
- Simple Agreement for Future Tokens (SAFTS), in which tokens are not minted and sold upfront, but rather investors buy a promise to deliver tokens [9]. This strategy is mainly aimed at accredited investors.

The article then reviews the strategies adopted by the main projects in the blockchain/cryptocurrency space:

- Bitcoin didn't distribute any tokens at genesis. All the tokens (the bitcoins) are continuously distributed as rewards for miners;
- Ethereum "pre-mined" some tokens, and then sold them in an "initial coin offering" (the first in history);
- Z-cash has a mechanism that assigns part of the tokens minted through mining to creators and early investors.

In addition, there are then strategies known as airdrops. In airdrops, tokens are distributed to active users (for instance, to a subset of Ethereum addresses with a balance greater than zero). While airdrops can easily target niche potential users, they may fail to generate engagement, since the user receives something passively, without "skin in the game". In contrast, interactive airdrops request some more or less costly action to the user willing to acquire the tokens. There are different declinations of interactive airdrops. In lockdrops, users can claim tokens by locking some value in a contract, for a given amount of time. In this way, the user shows a commitment and sustains a cost, in the form of opportunity lost with the value locked. In Merkle mine, a number of tokens are minted initially, and then users can reclaim them by providing a Merkle proof of inclusion of their addresses in the network state at some block height.

2.2. Burner wallets

If the distribution aims at users not accustomed to blockchain interaction tools, there are many cognitive and practical obstacles they have to overcome. For a pitiless but enjoyable representation of the paradoxes that a newcomer has to face, see Ref. [10]. Some of the most relevant barriers are:

- new users don't have a token custody application (a wallet);
- they cannot pay for the transaction fees (since they don't already have the necessary value tokens).

One of the most radical approaches to these issues is a tool called burner wallet, which was developed within the xdai project [11]. Xdai is an Ethereum Virtual Machine compatible blockchain with specific features, with a native value token, the xdai that can be swapped with the dai stablecoin on the Ethereum mainnet. The network is intended as a faster and lighter sidechain of Ethereum, for use cases such as payments. The burner wallet is an application that allows the transfer of xdais to a new user by means of a non custodial web app. If a sender wants to transmit some xdais, it accesses Burner [12], a web application that connects to the user's wallet and, with the sender's approval, sends the tokens to a special contract. At the same time, the app generates a link (and corresponding QR code) that redirects to a wallet created on the fly, to which the tokens can be redeemed. The initial fees to redeem the tokens are paid by the system. In this way, the user that gets the link can redeem the tokens, hold them in the wallet created on the fly (the burner wallet), and transfer them at any time. The keys of the created account are stored on the local storage of the browser. Application instructions recommend transferring the received tokens to a safer place, such as a cold storage wallet. The approach taken by the burner wallet significantly lowers users' onboarding barriers. Still, there are some points of centralization, among which the web server that provides the application that, in the case of unavailability for some reasons, makes token collection impossible.

3. Gap to be filled

From the analysis of the state of the art, numerous strategies and processes for the distribution of tokens emerge. Most strategies are either aimed at crypto proficient users or are custodial. The approach that is most "inclusive" towards crypto naive users is that of burner wallet, which generates URLs redirecting to a web app wallet. The system is limited to the xdai token, on the corresponding blockchain. It is then appropriate to evaluate the possibility of a general system for the

distribution of tokens, be they fungible or non fungible, with the following properties:

- minimal entering friction: no wallet should be required upfront, nor any other specific app; even more important, no initial token should be required to pay the initial transaction fees;
- the system should be maximally decentralized: user keys should not be held or known by any third party; token acquisition should be uncensorable, and the process should not rely on any server controlled by a third party (ideally, not even the web server that serves the app);
- it should be maximally fair: it should allow the possibility to assign tokens to users as in a provably fair lottery.

In this paper, we present a system with those properties, that also allows the distribution by means of physical objects, such as Near Field Communication (NFC) tags or printed QR codes. After having described the process and its implementation, we provide experimental performance and cost figures of the system in a real use case.

4. How we fill the gap

To understand the simplicity of the user experience, let us describe the steps that a user, Bob, has to follow in order to get onboarded and receive a certain amount of tokens on a brand new wallet:

- Bob receives somehow (via e-mail, physical token, printed on a paper sheet) a QR code;
- Bob scans the QR code with his smartphone camera;
- a popup appears on the smartphone asking Bob if he wants to open a link (specified in the QR code) with the web browser;
- the browser opens the link and redirects Bob to a web application;
- the web application asks Bob to perform the following tasks:
  - click on a button to generate a new account (a new wallet);
  - click on a button to redeem the token associated with the QR code;
- at this point, the application notifies Bob that the token has been received in his new wallet.

Bob is now the owner of the tokens, and he can use them in order to buy a service or a product, or to download a uniquely crafted document by simply proving ownership of the token.

Since a new wallet has been generated, the user is now able to redeem new tokens just by scanning other QR codes, without the need to repeat the entire flow from the start. This approach is also useful for the non-crypto users to better understand what is happening under the hood during the onboarding phase and it also increases the usability for redeeming further tokens.

Among other functionalities, the web application allows Bob to export the credential of his wallet (i.e., the private key of the address), so that he may transfer his wallet on other applications/tools (e.g., Metamask [13]).

4.1. System architecture

The overall system architecture is represented in Fig. 1. The on-chain logic of the system is composed of two smart contracts, written in Solidity [9] and deployed on the Ethereum mainnet. The contracts are:

- a webapp versioning contract [14];
- a token distribution contract [15].

The off-chain logic of the application is composed of:

- two web applications, stored on Interplanetary File System (IPFS) [16,17];
- a transaction relaying server.

4.1.1. QR code generation

The first steps are the initial minting of the tokens and the generation of the public/private key pairs (one for each token to be distributed). Each token is in fact linked, within the smart contract and during the minting phase, with the public key of a particular credential. The corresponding private key is used, combined with the URL of the web application, to generate the QR code to be distributed to the users. This QR code will then make it possible, for the user that receives it, to redeem the token associated with the public key corresponding to the private key contained in the QR code.

4.1.2. Webapp URL management

In the proposed system architecture, the HTML files of the web application are provided through the IPFS [18]. This makes it possible to:

- avoid the centralization due to a single server providing the web application to the users since IPFS is a decentralized file storage;
- guarantee the users that the application they are running on their browser has not been altered/modified/hacked, since IPFS files are addressed by their hashes.

The drawback of this solution is the impossibility of updating the application after its deployment since any change of the HTML changes the hash embedded in the QR code given to users. In the proposed architecture, we solved this issue with a couple of HTML web applications and a versioning smart contract that always contains the hash of the last version of the web application.

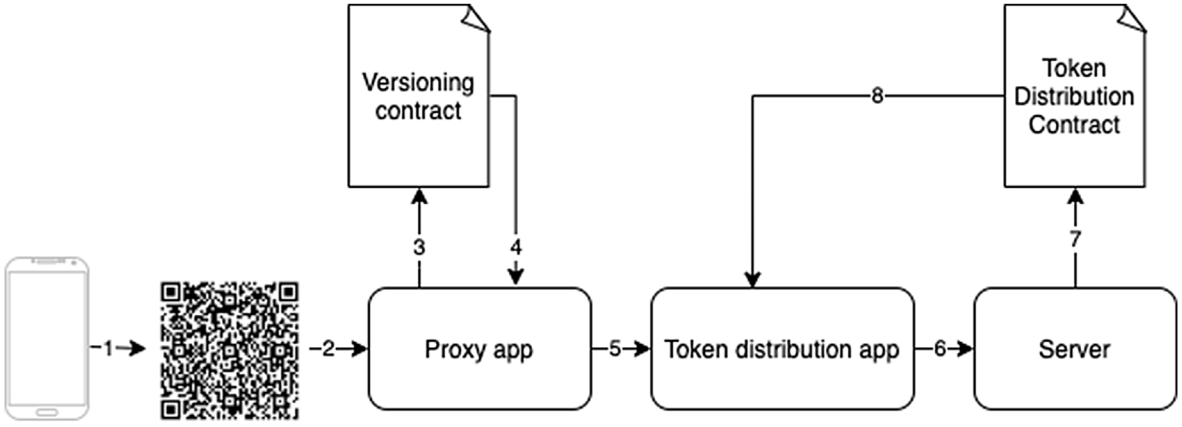

Fig. 1. Schema of the proposed architecture.

4.1.3. Versioning contract and proxy app

The Versioning Contract is used to locate the latest version of the wallet web application. To do so, it keeps track of all the application versions by storing their hashes in a list. The contract offers a public method that returns the last item on the list. Basically, the proxy application (whose hash is embedded in the QR codes provided to the users) only contains the logic to access the versioning smart contract, retrieving the hash of the last version of the web application and redirecting the user to it. The update of the web application, thus, requires only the change of its HTML code, the computation of the new hash, and the update of the versioning smart contract with this new hash.

4.1.4. Token Distribution App

The Proxy App redirects the user to the current version of the actual Token Distribution App. This is essentially a wallet that can generate a new Ethereum account (private key and address) on the spot and can interact with the Token Distribution Contract on-chain. The wallet receives, in the URL encoded in the QR code, a unique private key, whose corresponding public key is associated with one of the minted tokens. The key allows the application, signing a request, to redeem the corresponding token. To avoid transmitting the code to the IPFS gateway, this is inserted in the fragment portion of the URL, and is thus available only to the JavaScript application local to the device. The application then allows the user to:

- generate a new Ethereum account;
- generate a redeeming transaction, that contains a request, signed with the redeeming private key, to transfer a token to the newly generated account;
- send the transaction to a relying server (described below);
- export the generated account.

4.1.5. Token Distribution Contract

The token distribution contract manages the creation of unique digital assets that can be redeemed only by users that prove to own the rights. To serve this purpose, the contract implements an ERC721 interface and, for each non fungible token, keeps track of an asset (a PDF document in our experiment) it is associated with. Moreover, the contract stores a set of public keys that represent the ownership of the tokens.

All documents are first stored in IPFS, then the corresponding hash is registered on the blockchain and associated with a newly minted token calling a specific method, reserved to the contract owner. Some tokens can also be marked differently from the others, in order, for instance, to have “standard” and “premium” assets, as requested by the specific experiment. Another method allows adding the public keys that will be able to reclaim the tokens. Tokens and public keys are stored in different lists and will be dynamically associated during the redeem process, so the association is not known in advance. The private keys will be distributed to the users to produce a valid signature verifiable by the corresponding public keys.

4.1.6. Transaction relaying server

In order to make it possible for the users to redeem the token without requiring them to pay for the transaction fee, the proposed system architecture also envisages the use of a relay server. The web application retrieved by the users from IPFS tries to connect with this transaction relaying server, sending it a signed message that the server can include in a new transaction, thus paying the transaction fees. The token distribution smart contract is able to analyze the message sent by the relay server, which includes the message signed by the user web application, from which the token distribution smart contract can retrieve the address to which the new minted token has to be sent. In this context, the transaction relaying server cannot change the signed message or forge a fake signed message, thus the user (the owner of the private key) is the only one able to create and sign a valid message to redeem the token.

Note that this is not a single point of failure since the users always have the possibility of sending the transaction to the token distribution smart contract autonomously, with the only drawback that they have to pay for the transaction fees.

4.1.7. Properties

To summary, our solution has several properties:

- **easy access**: the decentralized application is web-based, therefore it does not require downloading any external applications;
- **availability over time**: the decentralized application is built in a single file and stored on IPFS, so anyone can keep it available by pinning that file (the control over the availability is on the users' hands);
- **flexibility**: users can choose whether to create a new wallet on the fly, to get an address of an existing wallet from Metamask, or to manually insert the address in which they want to receive the token;
- **security**: the QR code mechanism and the cryptography layer ensure that the relaying server can't act maliciously, “stealing” the requested token;
- **decentralization**: the system is designed to work even without the relaying server;
- **openness**: anyone can create his own interface to interact with the smart contracts by following the redeeming protocol, which is publicly available.

5. Experimental results

In this section, we present some details of the implementation and cost and performance figure of the solution, measured during a public event in which the system was used to distribute ERC721 tokens that gave the right to download a document of which different versions were provided.

The interface of the Token Distribution contract is a redeem function (named “getReport” in our experiment), that takes three arguments:

- the public key corresponding to the private key used to sign the message;
- the receiver address;
- the signature of the message containing the receiver address.

By using the information provided, the function is able to check the validity of the signature and compare the signer public key with the ones previously registered in the smart contract. If the signature is valid and has not been used yet, a token will be dynamically chosen and transferred to the provided address. This dynamic association is done using a “fair lottery” mechanism configured with a predefined probability that allows one to randomly assign special assets to some users and the non-special ones to others. To achieve an acceptable level of randomness in a deterministic system, it is necessary to consider different entropy sources. On the blockchain, miners have a little control on the current timestamp (around 15 s in Ethereum) and information on previous blocks. This may allow them to predict random numbers that rely only on these parameters. Our approach, as the code shown in Fig. 2, was to use a combination of block related values hashed together, a comparison between a byte taken from that hash and a byte taken from the signature (that cannot be known in advance by the system) in the function call. This makes it almost impossible for a user (and not convenient for a miner) to try predicting (or influencing) the outcome.

The contract uses two external libraries:

- ERC721 by OpenZeppelin to manage non fungible tokens[1];
- Solidity-signature-verify verify the signature[2].

[1] https://github.com/OpenZeppelin/openzeppelin-contracts.
[2] https://github.com/kabl/solidity-signature-verify.

```
function getReport(address      to,
                   bytes memory signature,
                   address      inputPublicKey)
                    public returns (uint256) {
        // [...]

        bool getSpecial = false;

        if(specialReportsIndex < specialReports.length &&
           reportsIndex < reports.length){
            bytes32 randomness = keccak256(
                abi.encodePacked(
                    block.coinbase,
                    block.timestamp,
                    block.difficulty,
                    blockhash(block.number-1),
                    block.number
                )
            );

            if((signature[31] >> 6) == (randomness[31] >> 6)
               ){
                getSpecial = true;
            }else{
                getSpecial = false;
            }
        }
        // [...]

        return tokenTable[inputPublicKey];
    }
```

Fig. 2. getReport function.

Table 1
Versioning contract size.

Source Code	Lines of code
Full Contract	20
Bytecode	**Bytes**
Full bytecode Size	2063
Deployed bytecode	1402
Initialization and constructor code	661

Table 2
Token distribution contract size.

Source Code	**Lines of code**
Full Contract (including libraries)	1326
Report Contract (excluding libraries)	120
Bytecode	**Bytes**
Full bytecode Size	17228
Deployed bytecode	16318
Initialization and constructor code	910

5.1. Contract size

Tables 1 and 2 provide an overview of the contract size considering source code and the bytecode generated. In this specific case, as highlighted in Table 1, a significant portion of code comes from external libraries, leaving room for further optimizations.

5.2. Gas analysis

Table 3 provides a detailed analysis of the transactions needed for contract deployment and methods execution, including cost in terms of gas and estimation in euro (considering a gas price of 11 gwei and the eur/eth change fixed to 195.43). As shown in Table 4, transaction costs for the whole project are sustainable with the given number of function calls ("# calls"), considering that the only infrastructure needed in addition to IPFS pinning and blockchain is the relay server.

Table 3
Versioning contract—gas analysis.

Solc version: 0.5.16+commit.9c3226ce		**Optimizer:** false		**Runs:** 200	**Block limit:** 6721975 gas	
Methods		11 gwei/gas				
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**eur (avg)**
DappURLResolver	newUrl	–	–	91714	2	–
Deployments					**% limit**	
DappURLResolver		–	–	566787	8.4%	–

5.3. Landing web application

The entry point of the system is a simple landing page that interacts with the versioning contract, retrieves the latest application hash, and redirects the user to the corresponding version of the Token Distribution application stored on IPFS, preserving the URL fragment.

When the user lands on this page, the underlying application instantiates the versioning contract and retrieves the link to the latest application by calling the getCurrentUrl function. Once done, the application redirects the user to the token distribution application forwarding all the parameters that are present in the URL fragment. The proxy application, after bundling, has a size of 403.96 KB.

5.4. Token redeem application

The web application is developed in Preact [19], a modern JavaScript framework with a small footprint and high performance, and ethers.js [20], a fully featured JavaScript library to handle blockchain interactions and wallet management. After loading the token redeem application, the

Table 4
Token distribution contract—gas analysis.

Solc version: 0.5.16+commit.9c3226ce		**Optimizer:** false		**Runs:** 200	**Block limit:** 6721975 gas	
Methods		11 gwei/gas				
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**eur (avg)**
Report	addReport	124631	169695	124854	450	0.27
Report	addSpecialReport	124694	154822	125405	50	0.27
Report	getReport	245963	276107	247655	500	0.53
Deployments				**% limit**		
Report		–	–	4604336	68.5%	9.81

user can decide to create a new wallet or to specify his own address manually or through the Metamask extension, if present. Then, the application allows the user to ask for his token. When the user starts the redeem process, the application reads a private key from the URL fragment, generates the signature, and sends a transaction to the smart contract. At the end of the process, the token will be transferred to the address specified in the contract call. In order to reach the maximum level of transparency, the application has been published on IPFS, which guarantees decentralization and immutability. The application bundle had been packed in a single HTML file paying attention to the bundle size to be smaller as possible. The token distribution application, after bundling, has a size of 993.20 KiB.

5.5. IPFS gateway

Since not all web browsers are yet able to retrieve documents directly from the IPFS network, gateways play an important role in making the service available to everyone. Two of the major gateways, Cloudflare [21] and Infura [22] were taken into consideration and compared in terms of reliability and performance. Finally, Cloudflare solution was chosen, also due to its gzip support.

5.6. Relayer service

Since we assumed that users had no value token to pay the redeeming transaction, a transaction relayer service was introduced.

The relayer service runs on a remote server and is divided into two interacting processes. The first offers a simple HTTP server able to receive HTTP requests from the redeem webapp, and only allows POST requests with three components:

- the signature of a message containing the target address that will receive the token, signed with the private key retrieved from the QR code;
- the corresponding public key;
- the address that will receive the token.

The information is then forwarded through a queue to the second process, which uses the private key of an external account in order to execute the token requests previously stored in the database. Those requests are first analyzed in order to check their legitimacy in two steps. First, the signature is decoded using the given public key, and the result must be equal to the given address that is going to be the recipient of the token; then the public key related to the scanned QR code must be verified as eligible by asking the smart contract if it has already been redeemed. These checks prevent illicit use of the relayer service, which could lead malicious users to force the system to submit failing transactions, draining all the available funds.

Once verified the legitimacy of a request, the relayer service proceeds to send the transaction to the smart contract, through an address preloaded with ethers, specifying the receiver address. Upon completion, the token is sent to the address specified.

It is impossible for the said service to manipulate the recipient address, inasmuch as also the smart contract performs the same checks on incoming token requests. Censorship actions are also not viable possibilities, since this service is offered as only one of the possibilities to request tokens. If it was down, the user could send the transaction autonomously, provided that he got some ether to pay the fee.

5.7. Real event application

The entire process described was used during an event held at the end of January 2020 at Politecnico di Milano. The system distributed tokens that entitled participants to download the annual report of the Blockchain and Distributed Ledger Observatory. A total of five hundred unique tokens were minted. Among those, fifty tokens were graded “winners” and only one of them was the first prize, granting the owner the possibility to attend some future courses. Five hundred private keys were printed on small chips in the form of QR codes and distributed among the participants. Once scanned with a smartphone, that QR code redirected the user to the web application, allowing it to request a token, through the relayer service. In the first hour of the event, more than 100 tokens (114) were successfully redeemed, other 190 tokens were distributed during the rest of the conference (3 h) and a total of 370 tokens were assigned overall. The cost of every transaction was US $0.27 (0.00125 Ether) and globally 881 transactions were submitted, including the first 500 for minting the tokens. The details of the transaction can be explored at the address of the contract with an Ethereum blockchain explorer such as Etherscan [15].

6. Scalability analysis

As known, two issues in using public blockchains are fees volatility and scalability. In the remainder, we consider different strategies for reducing fees and improving the throughput and latency of our system. Among all the different possibilities, we consider two approaches: sidechains and rollups, for which actual implementations currently exist. First we analyze and review their principles and their operations, and then we propose an experiment with a particular kind of rollups, the so-called “optimistic rollups”.

6.1. Sidechains

Sidechains [23–25] are blockchains that run with “lighter” consensus mechanisms (for example, Proof of Authority (PoA) [26,27]), that allow the system to run with lower latency and higher throughput, at the expenses of decentralization and resistance to certain forms of attacks (e.g., collusion of majority of validators). The trade-off should be justified by the value handled by the sidechain: the value handled by the chain should be lower than by the cost of the attacks.

Most sidechains strive to connect to one main, highly capitalized blockchain, such as Ethereum, (which is often referred to as the mainnet) for two reasons: to inherit part of the security features of the mainnet, and to be interoperable with the assets and smart contracts active there.

Sidechains can be connected to the mainnet with different mechanisms. One is to periodically commit their state on mainnet, sealing it from future tampering [28]. State commitments can also be used to access the state of the parent chain from the sidechain, for example, if the

commitment has the form of a Merkle root, it is possible for the logic on the sidechain to evaluate proofs about the parent chain state. It is also possible to two-way commit states (sidechain state is committed on the parent chain every so often, and vice versa), allowing mechanisms for the transfer of tokens and assets back and forth from/to the parent chain and sidechain [29].

Another connection mechanism is represented by bridges: they allow communication, potentially in various ways, with the mainnet. A particular kind of bridge is token bridges, which provide the possibility to send and receive tokens to and from the sidechain, but the mechanism can be generalized to any form of communication. Bridges usually rely on trusted oracles, responsible for relaying messages from one chain to the other. To reduce centralization, oracle activity is typically distributed among different actors. It makes sense to entitle validators of a PoA sidechain to act as information relayer towards the mainnet since they can control the state of the sidechain anyway.

The main advantage of side chains is their extreme flexibility: there is almost complete freedom over any parameter of the chain/system: consensus mechanism, scripting capabilities, account model, etc. On the other hand, the main problems are reliance on oracles for the communication to/from the mainnet and weaknesses related to the sidechain consensus (e.g., in a PoA sidechain, validators collusion can easily bring to censorship, funds freezing, etc).

In the Ethereum ecosystem, one of the main examples of sidechains is the xdai PoA sidechain, which we have already discussed in relation to burner wallets in Section 2.2.

6.2. Rollups

Another category of scaling strategies is rollups [30]. In rollups, transactions data is committed on the mainnet, but the execution (state evolution, storage, and computation of side effects) is performed off chain, and only the effects are "actuated" on chain. In other words, transactions are bundled and "rolled up", executed elsewhere, and their effects translated on chain. This allows enjoying the major benefits of the main blockchain (data availability, security, and incensurability).

The general question is: how is it possible to ensure that the computation performed off chain is correct? There are different mechanisms and technologies with which the problem can be addressed. The two main and more current approaches are zero knowledge proofs and optimistic virtual machines.

ZK-Rollups Zero Knowledge (or ZK-) rollups [31] use, as the name suggests, zero knowledge proofs to guarantee that the effects of many transactions are correctly taken into account and compounded. Moreover, ZK rollups exploit particular representations of transactions that use less information (e.g., sender and receiver are coded as indexes in an address book rather than as Ethereum addresses).

The basic idea is that a contract manages funds (either ethers or tokens) on behalf of its users, and keeps track of who owns what, in a way not dissimilar to a centralized exchange. A user can enroll/register in the system, and then can deposit, send and receive ethers and tokens from other users. The transfers among users only affect the internal state of the rollup, while deposits and withdrawals also affect other accounts. The state of the contract only contains the Merkle roots of the information of the users (address, balance, nonce). When a user wants to perform an operation, they specify the required information (his index in the address book, the recipient index, the amount, the fee he's willing to pay), sign it, and send it to a relayer. Relayers take batches of transactions, put them in a definite order, compute the effect on the state and the side effects (transfers of ethers and tokens), compute the new Merkle root, and create a ZK-snark [32,33] that demonstrates that:

- all the transactions were correctly signed;
- the cumulative effect of the transactions (state update + effects) is correct.

A crucial point is that anyone can play as a relayer. A user that thinks that relayers are censoring his transaction could just act as a relayer of himself, and post the update to the contract. Relayers are economically incentivized to include transactions since users can devote a portion of the transfer as a fee to the relayer that includes their transaction. The economic dynamics are similar to that of miners.

Since information is included in transactions and in logs, anyone can access all the information needed to create proofs, even if only Merkle roots are stored in the blockchain state. Transactions and log information cost much less than state storage, but have nonetheless a high degree of availability, guaranteed by being recorded on chain.

ZK-rollup can lower the cost of ethers transfers by 24x and of ERC20 token transfers by 50x [34]. The main advantages of ZK rollups are:

- more than order of magnitude savings in gas cost;
- Zero knowledge proofs make it impossible for a relayer to claim false consequences of users transactions.

While the main disadvantages are:

- ZK proofs verification systems currently used for rollups require an initial trusted setup: it is necessary to produce entropy data that must be destroyed after setup. If the entropy data is kept or known, it is possible to forge proofs of false claims. It is possible to generate, temporarily use, and then destroy entropy with protocols that offer a so-called "any-trust" guarantee: it suffices only one honest participant in the procedure to guarantee that the entropy has indeed been destroyed. These protocols are effective but require complex a "social" setup (participants have to be many or trustworthy).
- they work only for assets transfer, not for general smart contracts.

Let's consider whether zk-rollups would be applicable or convenient in our system, to accelerate token distribution and lower fees. Our system is composed of:

- an initial distribution system (a lottery);
- an ERC721 token.

While the exchange of the tokens could be accelerated and made cheaper by a zk-rollup mechanism, the initial distribution would be not. So, all in all, the advantages would be limited or marginal.

6.2.1. Optimistic rollups

Another type of rollups is "optimistic rollups" [35]. As in zk-rollups, the logic being scaled (e.g., a smart contract) is represented on chain with a commitment to the current state, while the state information itself is kept off-chain. Users submit transactions that are bundled by validators. Validators, in turn, compute the effect of a batch of transactions and call an on-chain contract with an assertion about the evolution of the state due to the transactions in the batch, and about the effects, such as ether transfers outside the contract, token transfers, and calls to other contracts.

Validators also insert the users transactions on-chain, via transaction data or in logs, so that all the information about all the transactions is available. While in zk-rollups validators also generated and submitted correctness proofs that were checked by a smart contract, optimistic rollups exploit another mechanism.

When a validator submits an assertion, he has to put down some stake. The effects of the assertion are not executed immediately. Rather, they are delayed by a "challenge period". Anybody can check whether his assertion is correct, because the transaction data is available, and their effects can be simulated. If anybody sees that the assertion is not correct, he can challenge the validator that posted it. The challenge is regulated by some on-chain logic. The system is designed to guarantee that, if a validator makes an untrue assertion, anyone can demonstrate that to the logic on-chain.

Once the validator is proved wrong, his stake is slashed, and the assertion is invalidated. The mechanisms for managing and resolving challenges vary among implementations. In Arbitrum [3] the validator and the challenger engage a first round in which they narrow down, through repeated bisections, the disagreement on the execution to a single instruction. At that point, the offending instruction is simulated by the logic on-chain, and the dispute is resolved. This mechanism features a trust property called any-trust: a single active honest actor is enough to guarantee that the effects of the transactions are correctly taken into account and executed.

The advantages of this solution are:

- with respect to sidechains, optimistic rollups offer stronger guarantees with lower trust assumptions (a single honest actor can force correct behavior);
- with respect to sidechains, interoperability with the mainnet is easier, since the control logic is executed already on it;
- with respect to zk-rollups, optimistic rollup works with arbitrary smart contracts and not just token transfers.

The main disadvantage of optimistic rollups is their strong asynchronicity: since a proper time for actors to challenge a validator assertion must be guaranteed, any transaction effect can be considered finalized by other smart contracts only after the challenge period. According to estimates [36], the correct sizing of the challenge period could go from a few hours to some days. Meanwhile, the assets accounted for in the rolled up contract cannot be transferred to other addresses. Note that, however, an external observer knows that a transaction will be correctly executed right after it is written on-chain, and well before the expiration of the challenge period. This could open up the possibility to lend liquidity in exchange for tokens that have been transferred in a rollup but are frozen until challenge period expiration. For instance, Alice could transfer 10 ethers to Bob inside a rollup. Bob, after Alice's transaction has been recorded on-chain, is sure that he will get, at the end of the challenge period, the 10 ethers. Meanwhile, he can lend the same amount on L1 to Alice for a fee.

6.3. Application to the token distribution system

At the moment, there are different implementations of optimistic rollups, the two main ones being Offchain Labs Arbitrum, and Optimism[4]. Arbitrum has recently developed a testnet environment for their solution. We used Arbitrum to experiment with porting the token distribution system to L2.

The idea is to deploy both the contract that represents the tokens (ERC721) and the distribution lottery on L2, and then to allow users to exchange tokens efficiently on L2, and the possibility to, of course, transfer them on L1 (for instance, to exchange them on OpensSea [37]). We focused in particular on the evaluation of the costs to create and distribute tokens within the rollup.

6.4. Scalability experiments

The Arbitrum Rollup provides a fully functional local environment and a public testnet built on top of the Kovan Ethereum testnet[5]. The validators network that is responsible for executing transactions off-chain provides a JSON-rpc interface that is compatible with web3 specifications [38] through a provider wrapper library, allowing it to interact in the same way as the Ethereum network. It is then possible to use the same tools (e.g., Truffle) to deploy contracts on L2.

We ran an experiment by directly interacting with the Arbitrum rollup network and we analyzed the transaction details using the provided explorer interface. We focused on the main three steps needed for our application to work, which are:

- Contract Creation: we deployed the token distribution contract directly on the rollup chain;
- addReport: we set up the smart contract by registering a new report;
- getReport: we called the function that allows the user to redeem the token associated with the previously added report.

We observed that, for each function call, a copy of the calldata of the transaction is sent by aggregators to L1 to the global inbox contract (see sendL1Message method). In order to reduce costs, multiple function calls can be batched in a single L1 transaction.

Table 5
Gas Cost comparison between rollup and L1 methods execution.

Function	Arbitrum: gas required for calldata	Arbitrum: gas required for the whole L1 transaction	Ethereum Mainnet Gas
Contract creation	266.279	303.238	4604.336
addReport	3.252	40.211	124.854

6.4.1. Costs analysis

Table 5 compares the gas required for directly calling the contract functions on the Ethereum network with the gas required for registering the call data of the same function on the Arbitrum global inbox contract. Since Arbitrum supports the registration of batch transactions, the cost may change according to the number of transactions to the same contract that will be included in the batch. For that reason, we decided to consider the worst case scenario in which only one transaction is included in the batch mentioned above.

The table has four columns that represent, respectively:

1. the function that is executed in the transaction;
2. the exact amount of gas that is needed to include the calldata in the transaction sent to the global inbox;
3. the amount of gas in column two to which we added the fixed amount (36.959) needed for calling the sendL1Message function on the global inbox contract with no data;
4. the amount of gas we measured during the live event in which interacting with the contract directly on the Ethereum main net.

As shown in Table 6, given the gas price for a standard transaction at the time of writing (28 gwei) and the current EUR/ETH exchange rate (484.92 EUR/ETH), the cost for running the contract on the Ethereum main network is very high compared to the Arbitrum counterpart.

At the time of writing this article, the Arbitrum project is still under development, therefore transactions inside the rollup network are completely free of charge. The protocol is designed to measure the computational resources needed for the aggregators and validators to execute transactions within the side chain (arbgas [39]). According to the Arbitrum documentation, the fee that will be necessary at the main net launch will be much lower than the current cost in the Ethereum main net.

6.4.2. Latency analysis

Since every side chain block must be committed on the mainnet, the block time cannot be lower than the mainnet. Even if this commitment has a challenging period in which it can be reverted if not valid, it is not necessary to wait until the block is finalized to accept its transactions. Since optimistic rollup is fork-free and all block data is available on-chain, it is always possible to perform client-side validation to check and accept them immediately.

[3] https://offchainlabs.com.
[4] https://optimism.io/.
[5] https://kovan-testnet.github.io/website/.

Table 6
Fiat cost comparison.

Function cost	Arbitrum: gas required for the whole L1 transaction	Cost	Ethereum Mainnet Gas	Cost
Contract creation	303.238	4.18 €	4604.336	62.52 €
addReport	40.211	0.55 €	124.854	1.70 €
getReport	40.977	0.57 €	247.655	3.37 €

7. Conclusions

We described a framework for the distribution of tokens to final users new to blockchain tools. The system is completely decentralized, non custodial, and provides a seamless user experience. The process was tested on the Ethereum main net during an event held at Politecnico di Milano in January 2020. Observations show that costs are comparable to those of a centralized solution, even though they could be further reduced by using new experimental technologies such as rollups. Further possible work includes the evaluation of distributed key generation for the generation of the redeeming keys.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Satoshi Nakamoto, A Peer-To-Peer Electronic Cash System, 2008. Available online: https://nakamotoinstitute.org/bitcoin/. (Accessed 18 December 2020).
[2] Omnilayer protocol specification, 2020. Available online: https://github.com/OmniLayer/spec. Accessed: 18 Dec 2020.
[3] Tether: Fiat currencies on the Bitcoin blockchain, 2016. Available online: https://tether.to/wp-content/uploads/2016/06/TetherWhitePaper.pdf. Accessed: 18 Dec 2020.
[4] Vitalik Buterin, Ethereum White Paper: A Next Gerneration Smart Contract and Decentralized Application Platform, 2014. Available online: https://bravenewcoin.com/insights/ethereum-white-paper-a-next-generation-smart-contract-and-decentralized. (Accessed 18 December 2020).
[5] F. Vogelsteller, V. Buterin, ERC-20 Token Standard, 2015. Available online: https://eips.ethereum.org/EIPS/eip-20. (Accessed 18 December 2020).
[6] W. Entriken, D. Shirley, J. Evans, et al., ERC-721: Non Fungible Token Standard, 2018. Available online: https://eips.ethereum.org/EIPS/eip-721. (Accessed 18 December 2020).
[7] F. Bruschi, M. Tumiati, V. Rana, et al., A decentralized system for fair token distribution and seamless users onboarding, IEEE Symposium on Computers and Communications 2020; 7–10 Jul 2020; Rennes, France. IEEE, Piscataway, NJ, USA (2020) 1–6.
[8] Smith+Crown, Introduction to token distribution mechanisms, 2019. Available online: https://sci.smithandcrown.com/research/introduction-token-distribution-mechanisms. (Accessed 18 Dec 2020).
[9] Solidity language documentation, 2016. Available online: https://solidity.readthedocs.io/en/v0.6.7/. Accessed: 18 Dec 2020.
[10] How dapps work, 2018. Available online: https://www.youtube.com/watch?v=XVZxjVJz4ds]. Accessed: 18 Dec 2020.
[11] XDai documentation. Available online: https://www.xdaichain.com/. Accessed: 18 Dec 2020.
[12] Burner Wallet Collective, Available online: https://burnerwallet.co/. Accessed: 18 Dec 2020.
[13] Metamask. Available online: https://metamask.io/. Accessed: 18 Dec 2020.
[14] Versioning contract on etherscan, 2020. Available online: https://etherscan.io/address/0x5d4F8c5AD3F0F1C6ccc95A89F6f3624B4e21bf81. Accessed: 18 Dec 2020.
[15] Distribution contract on etherscan, 2020. Available online: https://etherscan.io/address/0x9a2b38227b28ddd7d683f1801dd43444c090867b. Accessed: 18 Dec 2020.
[16] Proxy web application on IPFS, Available online: https://cloudflare-ipfs.com/ipfs/QmeA3j1DabhSja197Y7J4N7tPQY6nGroLVr6nePcfXBraJ. Accessed: 18 Dec 2020.
[17] Platform web application on IPFS, Available online: https://cloudflare-ipfs.com/ipfs/QmcvcUdseESddXB5M9YGCvegspGoNyu386UjReo3G4GQvL. Accessed: 18 Dec 2020.
[18] J. Benet, IPFS-content addressed, versioned P2P file system, arXiv, 2014 preprint. arXiv :1407.3561.
[19] Preact Library, Available online: https://preactjs.com/. Accessed: 18 Dec 2020.
[20] Etherjs Library, Available online: https://docs.ethers.io/ethers.js/html/. Accessed: 18 Dec 2020.
[21] IPFS gateway · Cloudflare distributed web gateway docs, Available online: https://developers.cloudflare.com/distributed-web/ipfs-gateway. Accessed: 18 Dec 2020.
[22] Ethereum API—IPFS API gateway—ETH nodes as a service—Infura, Available online: https://infura.io. Accessed: 18 Dec 2020.
[23] B.N. Musungate, B. Candan, U.C. Çabuk, et al., Sidechains: highlights and challenges, 2019 Innovations in Intelligent Systems and Applications Conference (ASYU); 31 Oct–2 Nov 2019; Izmir, Turkey. IEEE, Piscataway, NJ, USA, 2020, pp. 1–5.
[24] C. Worley, A. Skjellum, Blockchain tradeoffs and challenges for current and emerging applications: generalization, fragmentation, sidechains, and scalability, in: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData); 30 Jul–3 Aug 2018; Halifax, NS, Canada., IEEE, Piscataway, NJ, USA, 2018, pp. 1582–1587.
[25] R.M. Parizi, S. Homayoun, A. Yazdinejad, et al., Integrating privacy enhancing techniques into blockchains using sidechains, in: 2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE); 5–8 May 2019; Edmonton, AB, Canada., IEEE, Piscataway, NJ, USA, 2019, pp. 1–4.
[26] X.F. Liu, G.S. Zhan, X.M. Wang, et al., MDP-based quantitative analysis framework for proof of authority, in: 2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery; 2 Jan 2020; Guilin, China., IEEE, Piscataway, NJ, USA, 2019, pp. 227–236.
[27] N.A. Asad, M.T. Elahi, A.A. Hasan, et al., Permission-based blockchain with proof of Authority for secured healthcare data Sharing, in: 2020 2nd International Conference on Advanced Information and Communication Technology; 28–29 Nov 2020; Dhaka, Bangladesh, IEEE, Piscataway, NJ, USA, 2020, pp. 35–40.
[28] S. Johnson, P. Robinson, J. Brainard, Sidechains and interoperability, arXiv. 2019. preprint. arXiv: 1903.04077.
[29] A. Back, M. Corallo, L. Dashjr, et al., Enabling blockchain innovations with pegged sidechains, 2014. Available online: https://blockstream.com/sidechains.pdf. Accessed: 18 Dec 2020.
[30] Layer 2 scaling, Rollups, 2021. Available online: https://Ethereum.org/en/developers/docs/layer-2-scaling/#rollups. Accessed: 15 Sep 2021.
[31] Layer 2 scaling, Zk-rollups, 2021. Available online: https://Ethereum.org/en/developers/docs/layer-2-scaling/#zk-rollups. Accessed: 15 Sep 2021.
[32] J. Groth, On the size of pairing-based non-interactive arguments, in: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques; 8–12 May 2016; Vienna, Australia., Springer, Berlin, Heidelberg, Germany, 2016, pp. 305–326.
[33] B. Parno, J. Howell, C. Gentry, et al., Pinocchio: nearly practical verifiable computation, 2014 IEEE Symposium on Security and Privacy;19–22 May 2013; Berkeley, CA, USA. IEEE, Piscataway, NJ, USA (2013) 238–252.
[34] V. Buterin, An incomplete guide to rollups, 2021. Available online: https://vitalik.ca/general/2021/01/05/rollup.html. Accessed: 6 Jan 2021.
[35] Layer 2 scaling, Optimistic rollups, 2021. Available online: https://Ethereum.org/en/developers/docs/layer-2-scaling/#optimistic-rollups. Accessed: 15 Sep 2021.
[36] The optimistic rollup dilemma, 2020. Available online: https://medium.com/starkware/the-optimistic-rollup-dilemma-c8fc470ca10c. Accessed: 15 Sep 2021.
[37] OpenSea. Available online: https://opensea.io/. Accessed: 18 Dec 2020.
[38] Web3 API reference, Available online: https://web3js.readthedocs.io/en/v1.3.4/web3.html. Accessed: 18 Dec 2020.
[39] ArbGas and running time in Arbitrum · Offchain Labs dev center, Available online: https://developer.offchainlabs.com/docs/arbgas. Accessed: 18 Dec 2020.