future internet

MDPI

Article

A Blockchain-Based Framework to Enhance Anonymous Services with Accountability Guarantees

Francesco Buccafurri *, Vincenzo De Angelis and Sara Lazzaro

Department of Information Engineering, Infrastructure and Sustainable Energy (DIIES), Università Mediterranea di Reggio Calabria, Via dell'Università 25, 89122 Reggio Calabria, Italy
* Correspondence: bucca@unirc.it

Abstract: Anonymous service delivery has attracted the interest of research and the industry for many decades. To obtain effective solutions, anonymity should be guaranteed against the service provider itself. However, if the full anonymity of users is implemented, no accountability mechanism can be provided. This represents a problem, especially when referring to scenarios in which a user, protected by anonymity, may perform illegally when leveraging the anonymous service. In this paper, we propose a blockchain-based solution to the trade-off between anonymity and accountability. In particular, our solution relies on three independent parties (one of which is the service provider itself) such that only the collaboration of all three actors allows for the disclosure of the real identity of the user. In all other cases, anonymity is guaranteed. To show the feasibility of the proposal, we developed a prototype with user-friendly interfaces that minimize the client-side operations. Our solution is then also effective from the point of view of usability.

Keywords: Ethereum; anonymity; accountability; smart contracts

Citation: Buccafurri, F.; De Angelis, V.; Lazzaro, S. A Blockchain-Based Framework to Enhance Anonymous Services with Accountability Guarantees. *Future Internet* **2022**, *14*, 243. https://doi.org/10.3390/fi14080243

Academic Editors: Massimo Cafaro, Italo Epicoco and Marco Pulimeno

Received: 20 July 2022
Accepted: 18 August 2022
Published: 21 August 2022

1. Introduction

Anonymous services are privacy-preserving services offered to users without requiring them to disclose their identities.

We can distinguish two types of services. The first type is represented by one-time services exploited by users just once. Some examples are electronic auctions [1,2], anonymous surveys [3], or e-voting [4]. In the second type of services, users keep an anonymous account and exploit the same service more times. Each user is associated with a pseudonymous username to which their account's activity is linked. The action of users also includes possible data generated by the user when leveraging the service. A typical example is represented by anonymous social networks [5,6], in which, a pseudonymous username is associated with some data, such as private messages or posts. In this paper, we refer to the second type of services.

A minimal requirement is to provide anonymity to the users with respect to other users leveraging the same service. However, in order to be effective, anonymity should also be guaranteed against the service provider itself.

Even though, for the first type of services, full anonymity may be required, when referring to the second type, it is impractical to pursue user anonymity without taking accountability into consideration. Indeed, without the fear of being identified, held responsible, and punished when abusing the services, users are likely to misbehave due to selfishness or malice, thereby disrupting system operations and harming everyone else [7]. Therefore, a trade-off between anonymity and accountability exists [8].

In this paper, we try to solve this trade-off by proposing a blockchain-based protocol including two further parties, in addition to the user and the service provider. They are: (1) an identity gateway, which interfaces a standard identity provider (knowing the real identity of the user) with the other entities of our proposal, and (2) a linkage agency, acting

as a third independent party, which allows for the re-identification of the users if needed (e.g., when required by a court order).

In particular, our solution allows for the identification of the users only when all three parties (i.e., service provider, identity gateway, and linkage agency) collaborate. In all other cases, anonymity is guaranteed. Our paper reaches similar goals of approaches such as [9–11], with the relevant difference that, in our solution, the collaboration of just two third parties is not enough to de-anonymize the user.

From the security point of view, in our threat model, we require only a minimum degree of trust in the identity provider, in the sense that it does not perform active attacks such as impersonation. However, we allow it to passively collude with, at most, one of the other entities by disclosing the information that it knows (i.e., the real identity of the users), still preserving unlinkability. No trust is required in the identity gateway, service provider, and linkage agency.

At the basis of our solution, we rely on a challenge–response mechanism (see Section 4) leveraging a smart contract deployed on the Ethereum blockchain. This smart contract also provides an account recovery mechanism in the case where the user loses their access credentials. Indeed, traditional recovery mechanisms based on the telephone number or email address cannot be adopted since they would reveal personal information about users. Finally, another benefit of the blockchain technology is that the re-identification of the user as a consequence of malicious behaviour can be publicly verifiable by anyone.

To show the applicability of our proposal, we provide a user-friendly implementation, including a time-cost analysis. In this implementation, we care about usability by reducing the number of operations performed client-side. This shows that our solution can be easily applied in real-life contexts.

The structure of the paper is the following. In Section 2, we survey the related literature and provide some background notions about the Ethereum blockchain. The entities and the notations used to describe our proposal are introduced in Section 3. In Section 4, we provide the challenge–response mechanism at the basis of our solution. Then, in Section 5, we describe our proposal in detail. Its implementation is discussed in Section 6 and its security is analyzed in Section 7. Finally, in Section 8, we draw our conclusions and discuss future work.

2. Related Work and Background

In the scientific literature, there exist several papers dealing with the general issue of balancing anonymity and accountability in different fields.

Refs. [12–14] rely on anonymous credential systems and related schemes. Anonymous credential systems allow users to interact with a service provider in an anonymous yet accountable way [15–17]. Ref. [12] is the first paper proposing an anonymous credential system, in which, to prevent the misuse of anonymity, the anonymity property can be revoked for particular transactions. Ref. [13] introduces an approach relying on anonymous credentials that allows access control systems to offer fully anonymous access to resources along with strong accountability guarantees. It is worth noting that the proposed approach relies on a trusted third party to build a mechanism to escrow the identity of the user. Most of the solutions present in the literature assume a client–server architecture in which only the clients care about their privacy. On the contrary, Ref. [18] aims to reach the right balance between privacy and accountability in P2P systems, where both clients and servers are peer users. In [19], the authors deal with the issue of accountability in anonymous publication and storage services where malicious servers can make documents unrecoverable. To discourage this kind of behavior, the authors propose the creation of a "buddy system" that creates an association between pairs of shares from a given document. Therefore, a server holding a given share is responsible for detecting any anomaly regarding its buddy.

Ref. [20] proposes a framework to provide an anonymous mutual authentication protocol in wireless mesh networks. The proposed framework utilizes group signatures, where the private key and the credentials of the users are generated through a secure three-party protocol. User accountability is implemented via a user revocation protocol, whose execution can be carried out by two semi-trusted authorities, one of which is the network operator.

Refs. [21,22] deal with the issue of accountability in anonymous communication networks. Indeed, Ref. [22] highlights that both anonymity and accountability requirements should be satisfied to gain support for the deployment of large-scale anonymity infrastructures. To achieve this, Ref. [21] presents a mechanism that provides practical repudiation for the proxy nodes by tracing back a selected traffic to the predecessor node through a cryptographically verifiable chain.

A proposal that includes some similarities to our work is presented in [23]. Indeed, the authors proposed a solution that leverages the Bitcoin blockchain as a platform to manage and determine ownership of users' access credentials. The authors design an authentication scheme that is able to provide anonymity and accountability without relying on any trusted third party. Regardless, the proposed solution achieves accountability only in the sense that the service provider can blacklist the misbehaving credentials related to a user. Indeed, as highlighted by [24], accountability can still be obtained by eliminating the reliance on a third trusted party for credentials revocation, but this comes with a cost: the revoked user remains anonymous. Another solution relying on blockchain to support anonymous authentication in VANET is provided in [25].

We conclude this section by discussing some proposals about anonymous social networks [6]. Indeed, these appear as services that might take advantage of adopting our solution.

Anonymous social networks completely shift the traditional social networks paradigm. Indeed, while the latter put the user identity and its social link first, anonymous social networks encourage communication between strangers and allow users to express themselves without fear of bullying or retaliation [5]. Among the most famous anonymous social networks, we found Whisper [5] and Yik Yak [26].

Several studies in the literature focus on the anonymity guarantees offered by popular anonymous social networks [5,26,27]. However, to the best of our knowledge, no paper concerning the accountability problem is available, although several studies suggest that the lack of accountability may encourage users to engage in illegal behavior, potentially harming other users [7].

We conclude this section by providing some background notions about the blockchain technology with a focus on Ethereum. They will be useful for the understanding of some concepts that we recall in our protocol.

Blockchain is a peer-to-peer network that keeps track of the occurrence of events. An entity can generate a transaction towards another entity to exchange a value. This transaction is validated by peers participating in the network, and thus does not require any third-trusted party to validate the transaction. This represents the main advantage offered by this technology. Other features are:

- The transactions have to be validated and cannot be modified after their validation.
- Users cannot repudiate a transaction that they had generated.
- Anyone can access and verify the transactions stored on the blockchain.
- The users generating the transactions should remain anonymous.

Regarding the latter point, for example, Ethereum guarantees pseudonymity, in the sense that the identity of the users is not revealed, but their transactions are linkable.

The above properties are generic of the blockchain technology and apply to almost all of the existing public-permissionless implementations [28,29], even though there exist blockchains supporting full anonymity [30,31].

In order to give a concrete proposal, in this paper, we refer to the Ethereum blockchain.

In Ethereum, there are two types of *accounts*. The first type is represented by the *external owned accounts* (EOAs), which are controlled by users through private/public key pairs. Specifically, a user U generates a random string of 32 bytes. It represents the private key PRK_U of U.

Ethereum leverages the elliptic curve digital signature algorithm (ECDSA) [32] by selecting the curve `secp256k1`. Then, by applying the ECDSA algorithm to PRK_U, U obtains a public key PBK_U of 64 bytes (it represents the coordinates of a point of the elliptic curve). Finally, U applies the hash function `Keccak256` [33] to PBK_U and takes the last 20 bytes to obtain an *Ethereum address*. This Ethereum address is used to send and receive transactions, and identifies (in pseudonymous form) U in the network.

Each transaction generated by U is signed with the private key PRK_U. The signed transaction is broadcasted in the blockchain network. Anyone can verify the signature by retrieving the public key of the signer (from the signature and the transaction) and checking that it is equal to the expected public key. Furthermore, the Ethereum address of the signer can be obtained from the public key by applying `Keccak256` as described above.

The second type of account is called a contract account, and represents an account controlled by code (i.e., a smart contract). A smart contract [34] can be viewed as a collection of data and functions. To deploy an instance of a smart contract, an EOA sends an Ethereum transaction containing the compiled code (on the Ethereum VM) of the smart contract without specifying any recipient. Each instance has its own data (also called state) and functions, and it is associated with an Ethereum address. To invoke a function of the smart contract, an EOA generates a transaction, including the input of the function, intended for the address of an instance of the smart contract. This function may change the state of such an instance. Since the transaction invoking the function is stored on the blockchain, due to the properties of immutability and non-repudiation, anyone can verify the correct state of any instance at any time.

3. Actors and Notation

Our solution involves four actors:

- The user U: they require access to an anonymous service offered by a service provider SP.
- The service provider SP: it offers the anonymous service to U. U interacts with SP through a pseudonym username $user_U$ not linkable to thereal identity ID_U of U.
- The identity gateway IG: it does not know ID_U, but acts as a gateway between an identity provider who knows ID_U and the Ethereum network. In principle, it may be included directly in the identity provider. Regardless, we keep it as a separated entity in order to not require technological changes in the identity provider and maintain it as compliant with the eIDAS regulation [35].
- The linkage agency LA: it is a third independent party that, after receiving an order by a court, is able, through the collaboration of IG and SP, to link the real identity ID_U with the username $user_U$.

As discussed in Section 1 and shown in Section 7, the collaboration of just two out of IG, SP, and LA is not enough to associate ID_U with $user_U$, and all three parties have to be involved.

During the interaction between U and SP, some information is generated and associated with $user_U$ (since SP does not know the real identity ID_U). We denote by I_U such account information. For example, if SP is an anonymous social network, I_U is represented by the private messages exchanged between U and other users, the messages posted by U, the “likes" released by U, and so on. Like almost all of the web services, I_U is maintained server-side by SP. Observe that the access to I_U is what the user has to recover in case of a loss of credentials.

An association among EOAs (see Section 2) and actors is established. Specifically, U owns two Ethereum addresses Add_1^U and Add_2^U. SP owns the Ethereum address Add^{SP}. IG owns the Ethereum address Add^{IG}. Finally, LA owns the Ethereum address Add^{LA}.

Observe that Add_1^U and Add_2^U are pseudonyms of U not linkable between them or with ID_U. Add^{SP}, Add^{IG}, and Add^{LA} are made publicly available by SP, IG, and LA, respectively.

As we will see in Section 4, our proposal is based on a challenge–response mechanism implemented by the smart contract reported in Listing 1. A number of instances of such a smart contract are deployed by SP, IG, and LA.

Listing 1. Smart contract implementing a challenge–response mechanism.

```
contract SmartContract {

  address private owner;

  mapping(address => string ) public stateMap;
  mapping(address => bytes32 ) challengeMap;

  modifier isOwner() {
   require(msg.sender==owner, "Caller is not owner");
    _;
  }

  constructor() {
    owner=msg.sender;
  }

 function setChallenge(address user, bytes32 challenge) public isOwner {
    stateMap[user]="To Confirm";
    challengeMap[user]=challenge;
  }

  function solveChallenge(bytes32 solution) public{
    address sender=msg.sender;
    if(compareStrings(stateMap[sender],"To Confirm")){
      bytes32 challenge=challengeMap[sender];
      if(keccak256(abi.encode(solution))==challenge){
           stateMap[sender]="Confirmed";
            }
        }
    }

  function compareStrings(string memory a, string memory b) public view
  returns (bool) {
    return (keccak256(abi.encodePacked((a))) ==
    keccak256(abi.encodePacked((b))));
  }
}
```

Specifically, SP deploys two instances C_1^{SP} and C_2^{SP} with Ethereum addresses $Add^{C_1^{SP}}$ and $Add^{C_2^{SP}}$, respectively. C_1^{SP} is used by U during the registration phase with SP. It allows SP to associate the Ethereum address Add_2^U with $user_U$. Clearly, this operation is performed by SP without knowing the real identity ID_U of U. Moreover, $user_U$ is not publicly disclosed. C_2^{SP} is used by U to retrieve I_U in case U loses the credentials to connect with SP.

IG deploys the instance C^{IG} with Ethereum address $Add^{C^{IG}}$. It is used by IG and the identity provider to associate Add_1^U with the real identity ID_U of U without disclosing it to IG.

Finally, LA deploys the instance C^{LA} with Ethereum address $Add^{C^{LA}}$. C^{LA} is used by LA to link the Ethereum addresses Add_1^U and Add_2^U of U (without knowing the real identity ID_U and the username $user_U$).

To conclude this section, we introduce a notation to model the Ethereum transactions. In our application, the transactions are generated by the actors only towards instances of a smart contract to invoke some functions. No user-to-user transaction is performed and no Ether transfer is needed. Therefore, we modeled an Ethereum transaction as a tuple $T = \langle sender, destination, function(params) \rangle$.

$sender \in \{Add_1^U, Add_2^U, Add^{SP}, Add^{IG}, Add^{LA}\}$ represents the Ethereum address of the actor generating the transaction. $destination \in \{Add^{C_1^{SP}}, Add^{C_2^{SP}}, AddC^{IG}, AddC^{LA}\}$ represents the destination address of the transaction. It can be an instance of the smart contract. Finally, $function(params)$ represents the invoked function of the smart contract along with the input parameters.

4. Challenge–Response Mechanism

At the basis of our protocol, there is a challenge–response mechanism implemented by the smart contract reported in Listing 1.

It offers the following three security properties.

- P1: IG, SP, and LA have to be sure that the users really own the claimed Ethereum addresses.
- P2: if the previous check is satisfied and the owner of an address is able to solve a given challenge, then the smart contract sets, in a verifiable way, the state of such an address as ‘‘`Confirmed`’’.
- P3: the smart contract allows us to notarize some information.

Even though, at this stage of the paper, it is not clear why we need these properties, we show as the smart contract allows us to satisfy them. Further details are provided in Section 5, where we exploit the proposed mechanism to implement our protocol.

Our smart contract includes two mappings, called `stateMap` and `challengeMap`, and two functions, called `setChallenge` and `solveChallenge`.

`stateMap` associates an Ethereum address with a string representing the state of such an address. We admit three possible states: ‘‘’’ (the built-in default state of Ethereum smart contracts), ‘‘`To Confirm`’’, and ‘‘`Confirmed`’’. By default, all of the addresses are in state ‘‘’’.

`challengeMap` associates an Ethereum address with a `bytes32` variable containing the challenge that such an address has to solve.

`setChallenge` can only be invoked by the owner (i.e., the entity who deployed) of the instance of the smart contract (in our application, the owner can be IG, SP, or LA). It receives as an input an Ethereum address and a challenge. Then, it associates the challenge with the address in the `challengeMap` and sets the state of the address to `"To Confirm"` in the `stateMap`.

`solveChallenge` can be invoked by anyone and receives the solution to the challenge as an input. The solution to the challenge is a 32-byte word (corresponding to the digest of a cryptographic hash function). The technical detail about how this solution is computed is described in Section 5. First, it checks that the address originating the transaction invoking this function is in state ‘‘`To Confirm`’’. This guarantees P1, since only the owner of an Ethereum address can generate a transaction from such an address. If the check is positive, then the function verifies that the solution solves the challenge associated with this address. In the positive case, the state of the address is set to ‘‘`Confirmed`’’ in the `stateMap`. This clarifies how the smart contract guarantees P2.

The mechanism to verify the solution to the challenge is straightforward. The smart contract simply checks that $challenge = Keccak256(solution)$.

Regarding P3, the solution to the challenge itself represents the information to notarize. Observe that, since this information is published clearly on the blockchain, it should not compromise the anonymity of the user. This will be explained in the next section, in which, the whole protocol is described.

5. The Proposed Approach

Our solution involves three interactions performed just once (in a registration phase) by U with IG, SP, and LA, respectively. The goal is to obtain accountability only if LA collaborates with SP and IG to discover the real identity of a user performing illegally when using the anonymous service offered by SP. Furthermore, the proposed approach provides a mechanism to recover the account information I_U when U loses the access credentials.

5.1. Interaction between U and IG

Through this interaction, the Ethereum address Add_1^U is associated with the real identity ID_U of U. This is performed without IG learning anything about ID_U. At the end of this phase, Add_1^U will result in state ''`Confirmed`'' and will be used by LA in the next interaction. This phase proceeds as follows.

At the start, we assume that U is registered with an identity provider that knows their real identity ID_U.

In our solution, IG offers a federated-based authentication scheme [36] possibly relying on different identity providers.

First, U authenticates with IG through the federated scheme by selecting the identity provider that U is registered to. We denote by IP such an identity provider.

As a standard federated authentication, when IP identifies U, it provides IG with a signed assertion not containing information associated with ID_U. In this assertion, IP includes $Keccak256(R||ID_U)$, where R is a random value. R is then associated with ID_U and stored along with $Keccak256(R||ID_U)$ by IP.

Once the authentication is performed, U provides IG with Add_1^U.

Then, IG computes a challenge $ch = Keccak256(Keccak256(R\|ID_U)))$ and generates a transaction $T_1 = \langle Add^{IG}, Add^{C^{IG}}, setChallenge(Add_1^U, ch)\rangle$ intended for the instance C^{IG} of the smart contract. This transaction invokes the function `setChallenge` that associates the challenge ch with the address Add_1^U and sets Add_1^U in state ''`To Confirm`''. Finally, IG sends U the digest $Keccak256(R\|ID_U)$.

After receiving this value, U uses it as solution to the challenge ch, i.e., $sol = Keccak256(R\|ID_U)$, and generates the transaction $T_2 = \langle Add_1^U, Add^{C^{IG}}, solveChallenge(sol)\rangle$.

This proves to IG that U is the owner of Add_1^U. The function `solveChallenge` sets Add_1^U in state `"Confirmed"`.

Observe that the random R acts as a salt for the hash function $Keccak256$. This way, ID_U cannot be reversed by sol, even when performing dictionary attacks by testing all of the possible real identities. On the other hand, if the identity provider discloses R to LA, the association $ID_U - Add_1^U$ can be verified through the instance C^{IG} of the smart contract.

Finally, IG verifies the state ''`Confirmed`'' of the address Add_1^U and locally stores the tuple $\langle Add_1^U, Keccak256(R\|ID_U)\rangle$ to provide LA if needed.

The above steps are summarized in the sequence diagram of Figure 1.

The next two interactions are between U and LA and between U and SP, respectively. They have to be performed in anonymous form without any authentication of U. Clearly, to avoid IP geolocation, anonymous communication protocols such as Tor [37] and VPNs [38] can be adopted. We do not treat this aspect in detail since it is out of the scope of this paper.

5.2. Interaction between U and LA

Through this interaction, LA links Add_1^U and Add_2^U without knowing the real identity of U.

First, U (anonymously) contacts LA and provides it with Add_1^U. LA needs to verify two conditions. The first is that Add_1^U is an Ethereum address associated with a real identity. However, LA does not need to know such a real identity. This condition can be easily verified by checking that Add_1^U is in state ''`Confirmed`'' on C^{IG}. The second condition is that Add_1^U really belongs to the user contacting LA. This is carried out to avoid a user trying to impersonate another user already authenticated with IG.

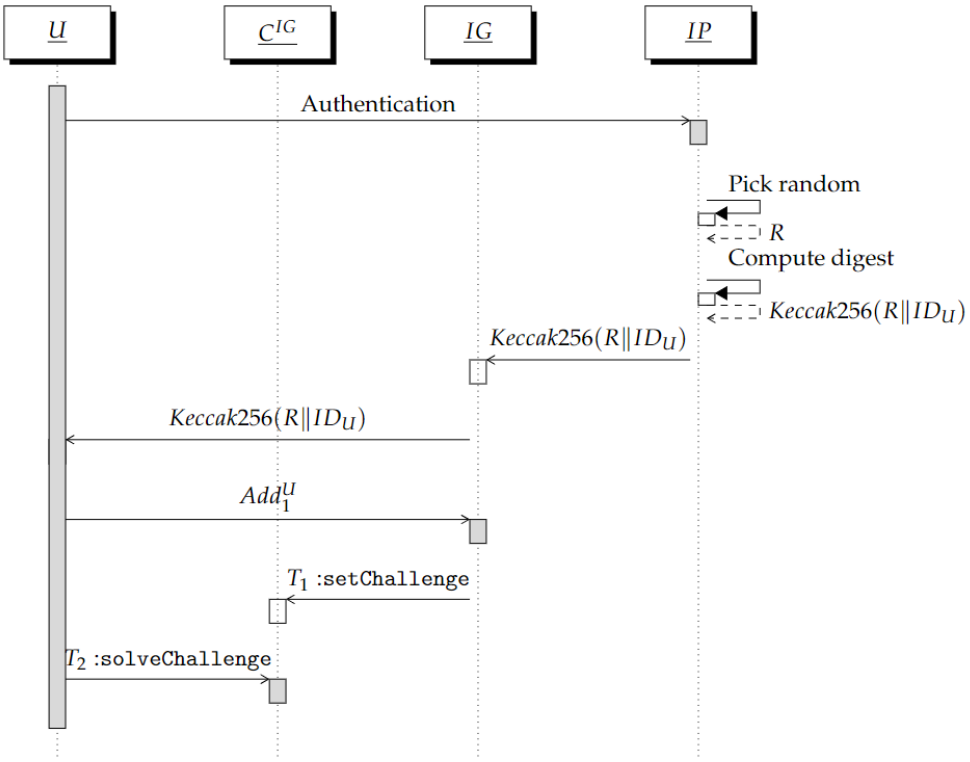

Figure 1. Sequence diagram of the interaction between U and IG.

Even though the challenge–response mechanism used in the previous section allows this task, we should not use it. Indeed, it requires some interactions through the blockchain that would disclose the linkage $Add_1^U - Add_2^U$ to anyone.

Therefore, to verify the second condition, LA generates a random R' and provides U with it. U signs R' with the private key associated with Add_1^U, obtaining a signature σ. Finally, U replies to LA with the pair (σ, Add_2^U).

Starting from σ and R', LA verifies the signature and retrieves the public key associated with Add_1^U. Then, as explained in Section 2, LA computes the hash function `Keccak256` on this public key and takes the last 20 bytes to obtain the address $\overline{Add_1^U}$. If $\overline{Add_1^U} = Add_1^U$, then LA is sure that the user contacting it owns Add_1^U.

At this point, LA needs to verify that U owns Add_2^U. The same challenge–response mechanism used with IG is adopted. This way, Add_2^U will move to the state "`Confirmed`" on the instance C^{LA} and will be used by SP in the next interaction with U.

In detail, LA picks a random $\bar{R}$ and generates a challenge $ch = Keccak256(Keccak256(\bar{R} \| Add_1^U \| Add_2^U \| \sigma))$.

Then, LA generates a transaction $T_3 = \langle Add^{LA}, Add^{C^{LA}}, setChallenge(Add_2^U, ch) \rangle$. This transaction invokes the function `setChallenge` that associates the challenge ch with the address Add_2^U and sets Add_2^U in state "`To Confirm`". The random $\bar{R}$ is provided to U.

U computes the solution of the challenge as $sol = Keccak256(\bar{R} \| Add_1^U \| Add_2^U \| \sigma)$ and generates the transaction $T_4 = \langle Add_2^U, Add^{C^{LA}}, solveChallenge(sol) \rangle$. This proves to LA that U is the owner of Add_2^U, which will be set in state "`Confirmed`" by the function `solveChallenge`.

Finally, LA verifies the state "`Confirmed`" of the address Add_2^U and locally stores the tuple $\langle Add_1^U, Add_2^U, \bar{R}, \sigma \rangle$.

The above steps are summarized in the sequence diagram of Figure 2.

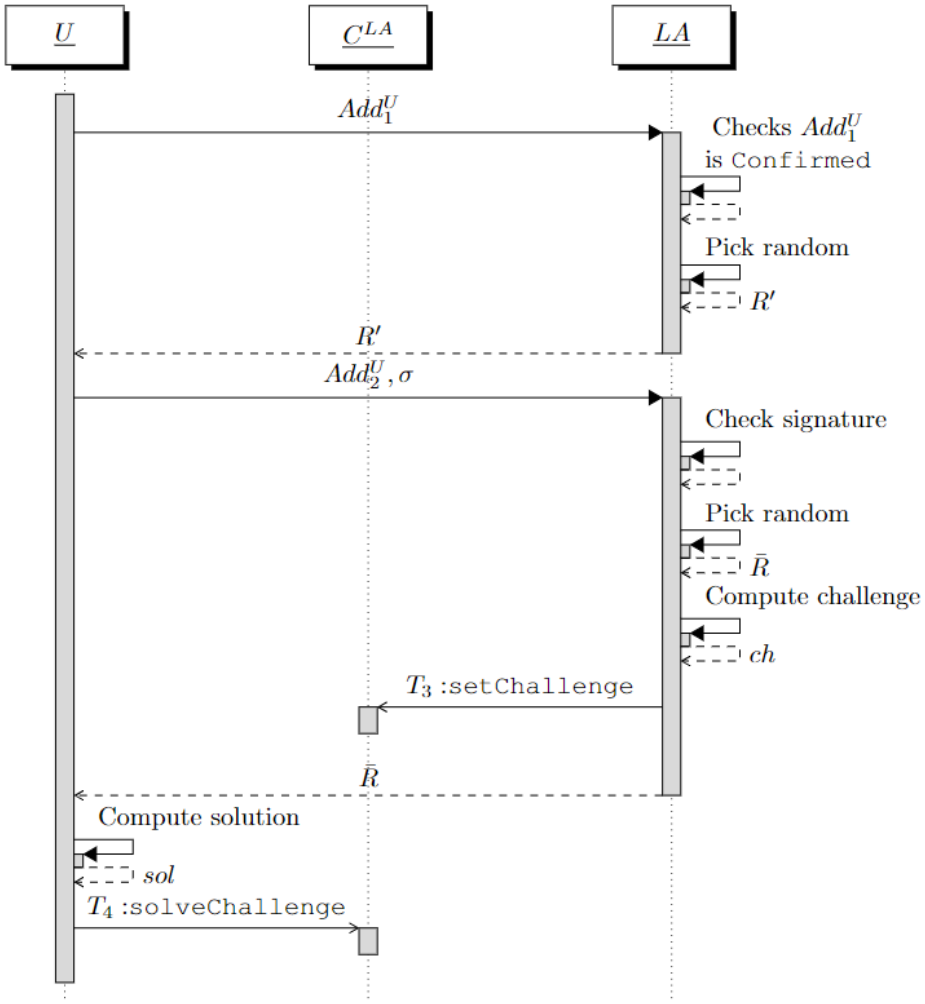

Figure 2. Sequence diagram of the interaction between U and LA.

5.3. Interaction between U and SP

Through the third interaction of our protocol, SP associates Add_2^U with a pseudonym $user_U$ chosen by U. For simplicity, we assume that the access credentials of U are represented by a username–password pair. However, the password can be replaced by the device ID of the user's phone, as with Whisper [5].

Again, we exploit the challenge–response mechanism of Section 4.

U provides SP with $(Add_2^U, user_U, pass_U)$, where $pass_U$ is the password chosen by U. SP verifies that Add_2^U is in state "`Confirmed`" on the instance C^{LA}. Then, it picks a random $\hat{R}$ and generates the challenge $ch = Keccak256(Keccak256(\hat{R} \| user_U))$. Finally, SP generates a transaction $T_5 = \langle Add^{SP}, Add^{C_1^{SP}}, setChallenge(Add_2^U, ch) \rangle$. This transaction invokes the function `setChallenge`, which associates the challenge ch with the address Add_2^U and sets Add_2^U to state "`To Confirm`". The random $\bar{R}$ is provided to U.

U computes the solution to the challenge as $sol = Keccak256(\hat{R} \| user_U)$ and generates the transaction $T_6 = \langle Add_2^U, Add^{C_1^{SP}}, solveChallenge(sol) \rangle$. This proves to SP that U is the owner of Add_2^U, which will be set to state "`Confirmed`" by the function `solveChallenge`.

Finally, SP verifies the state "`Confirmed`" of the address Add_2^U on the instance C_1^{SP} and locally stores the tuple $\langle Add_2^U, user_U, \hat{R}, pass_U \rangle$. Observe that the password $pass_U$ is not included in the notarized information since it is used by SP only to authenticate U.

The above steps are summarized in the sequence diagram of Figure 3.

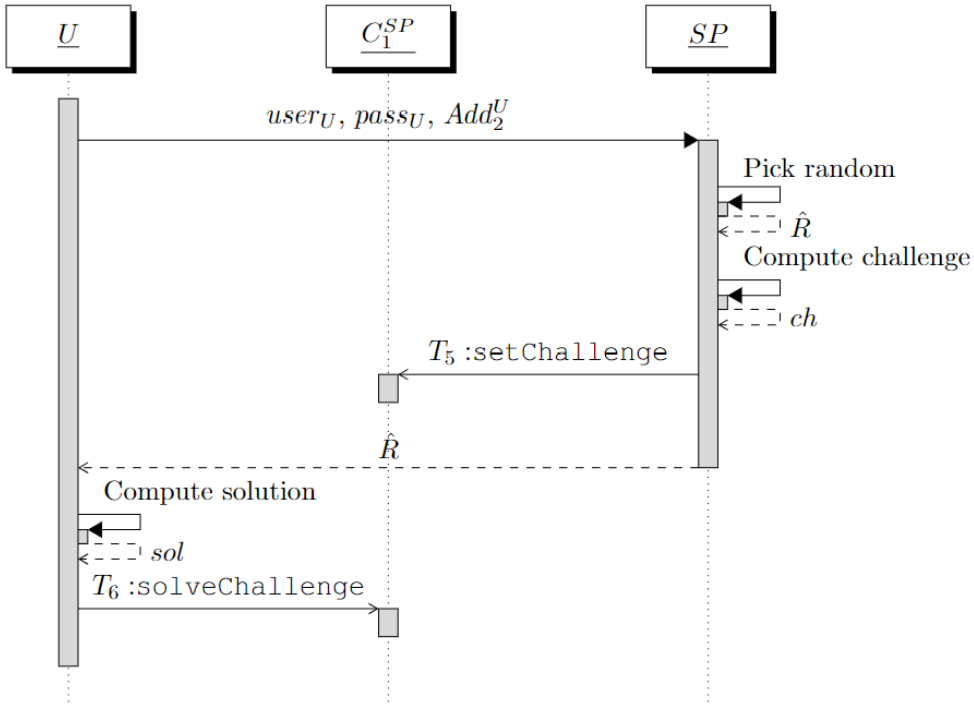

Figure 3. Sequence diagram of the interaction between U and SP.

As a final remark, we want to observe that, if the user is not registered with IP, SP does not provide the anonymous service. Indeed, it checks that the address Add_2^U is in the state ''`Confirmed`''. Moreover, this state is set by LA, which, in turn, verifies that Add_1^U is in the state ''`Confirmed`''. The state ''`Confirmed`'' of Add_1^U is set by IG only after the identity of U is verified by IP.

We conclude this section by observing that, as we will see in Section 6.2, in order to develop a user-friendly application, almost allof the client-side operations are performed automatically by some scripts without requiring the interaction of the users. They just have to confirm some operations (the generation of the transactions and signature of messages) for security reasons. Furthermore, to simplify the client-side operations, the interactions of U with SP and LA are merged into a single interaction with SP in which the user is redirected to LA.

5.4. Account Information Recovery

Suppose U loses the access credentials with SP and wants to recover the account information I_U. U just needs to prove to SP the ownership of the Ethereum address Add_2^U with which U is registered at SP. This is carried out, as already discussed in the previous sections, through the challenge–response mechanism involving the instance C_2^{SP} of the smart contract. After verifying the possession of Add_2^U, SP enables the standard module to change the password and provides U with I_U.

5.5. Law-Enforced Re-Identification of U

Suppose a law court receives a notification about an illegal behavior of a user with username $user_U$ when leveraging the anonymous service provided by SP. After verifying whether such behavior violates the law, it emits an order to ask LA to retrieve, in a verifiable way, the real identity ID_U of the user with username $user_U$. First, LA contacts SP to obtain the Ethereum address Add_2^U associated with $user_U$. Then, LA retrieves the Ethereum address Add_1^U linked with Add_2^U (this linkage is maintained by LA itself). Subsequently, LA contacts IG to obtain the digest $Keccak256(R||ID_U)$ associated with Add_1^U. Finally, LA contacts IP to obtain the real identity ID_U of U associated with $Keccak256(R||ID_U)$. Observe that the mapping $user_U - Add_2^U$ is verifiable through the instance C_1^{SP} by disclosing

the random $\hat{R}$. Similarly, the mapping $Add_1^U - Add_2^U$ is verifiable through the instance C^{LA} by disclosing the random $\bar{R}$. Finally, the mapping $Add_1^U - ID_U$ is verifiable through the instance C^{IP} by disclosing the random R.

5.6. Linkage of Multiple Anonymous Accounts

Suppose that a law court wants to monitor the anonymous activity of the user U on multiple service providers. This means that the court wants to associate the identity ID_U with all of the usernames used in the various service providers. As in Section 5.5, suppose that the court charges the linkage agency LA to perform this operation. First, LA contacts IP. We recall that IP stores a set of tuples in the form $T = (R, Keccak256(R||ID_U))$. Observe that each tuple is associated with a service provider to which U requires an anonymous service. For each tuple T, LA contacts IG, and, through $Keccak256(R||ID_U)$, it retrieves the address Add_1^U. Then, through the mapping maintained by LA itself, LA retrieves the address Add_2^U used by U with a service provider SP. Finally, through SP, LA discovers the username $user_U$.

As in Section 5.5, all of the linkages are verifiable on blockchain.

6. Implementation and Time-Cost Analysis

In this section, we describe the prototype developed to implement the solution proposed in Section 5. The source code of our prototype is publicly available on GitHub at https://github.com/SaraLazz/AccountAnon (accessed on 18 August 2022).

6.1. Adopted Technologies

We developed three JAVA web applications to implement IG, SP, and LA. Each application leverages the Apache Struts2 web framework [39]. Struts is an open-source framework that employs a model, view, controller (MVC) architecture and that enables the creation of maintainable and flexible web applications. The view part of our web applications was constructed using JavaServer Pages (JSP) [40]. To improve the user experience, we used asynchronous JavaScript and XML (AJAX) [41], thus allowing the user's interaction with the application to happen asynchronously.

As explained in the previous sections, our web applications need to interact with the Ethereum blockchain both server-side and client-side.

Regarding server-side, our applications leverage on Infura [42]. The Infura API suite allows us to access the Ethereum network through HTTPS and WebSockets. Infura has the main advantage of providing all of the necessary tools to develop on Ethereum, without the need to locally run any blockchain node.

Regarding client-side, our applications interface with MetaMask [43] via JavaScript. MetaMask is a very popular application that allows users to write on the blockchain. One of the main benefits of using MetaMask is that users' passwords and keys remain on the user device since they are not shared with any other parties interacting with it.

6.2. Implementation Detail and Prototype Functionalities

Through this section, we describe in detail the implementation of the prototype (including the three web applications) and describe how it works.

The implementation of the IG web application (whose user interface is represented in Figure 4) follows on from what has been described in Section 5. The selection of the address Add_1^U is executed client-side via JavaScript connecting to MetaMask. It replies with the address that is currently selected in the application. At this point, the challenge–response mechanism starts. IG generates the challenge server-side and invokes the smart contract function `setChallenge` via Infura API. After this, the user, client-side, solves the challenge by invoking the smart contract function `solveChallenge` via MetaMask.

Now, we move on to the other two applications.

Even though the high-level workflow described in Section 5 only requires separate interactions between U and the two parties SP and LA, the technical implementation of the

prototype performs this transparently for the user through a redirection of the web traffic from SP to LA .

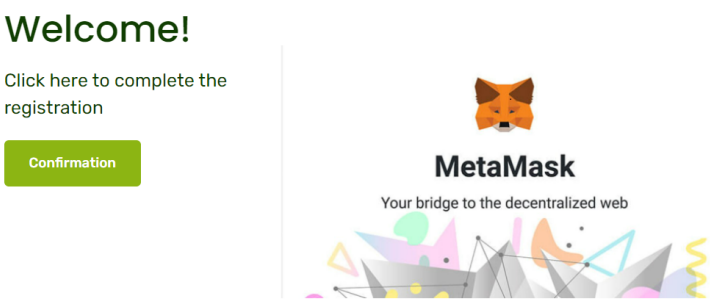

Figure 4. User interface of IG.

In detail, when the user asks to subscribe to SP, it will redirect the user to LA for the verification of their Ethereum addresses. The procedure executed with LA is divided into four steps, as represented in Figure 5.

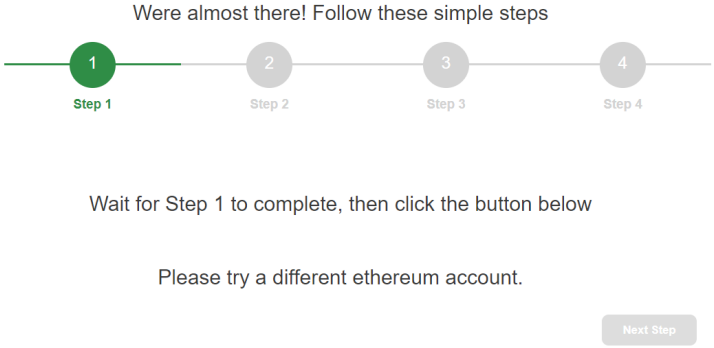

Figure 5. Selection of a IG-validated Ethereum account.

The first step is to aid the user in the selection (on Metamask) of the address Add_1^U so that the user does not have to remember it. The LA client-side application asks MetaMask the current address selected by the user and verifies whether it is or is not validated by the IG (i.e., if it is memorized in state "`Confirmed`" in the instance C^{IG}). In the case where it is not validated, the LA application returns to the user a message asking them to select a different address (Figure 5). The procedure continues until the user selects an address validated by IG. At this point, the user can proceed with the second step (Figure 6).

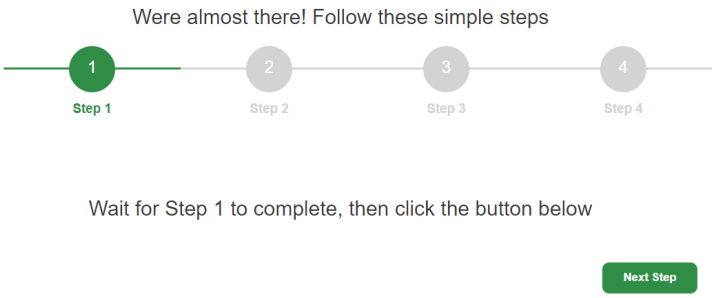

Figure 6. Completion of Step 1.

During the second step, the user is asked to sign a random provided by the server (Figure 7). Then, the signature is sent to the server along with the address Add_1^U so that the server can verify that the user owns the address Add_1^U.

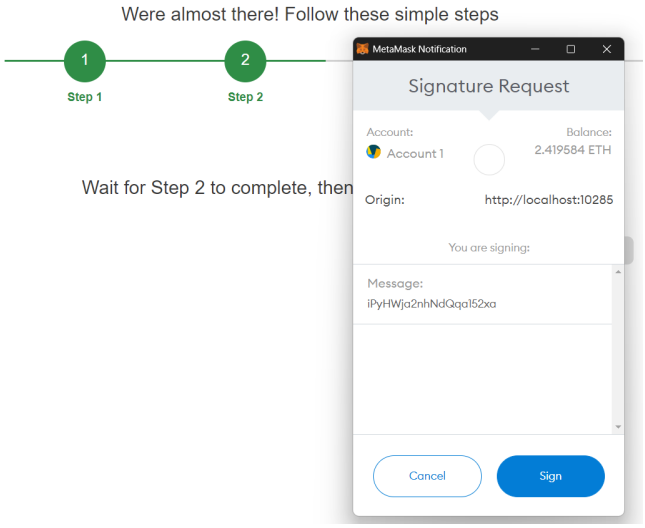

Figure 7. Signature of the LA challenge.

During the third step, the user is asked to switch to an address that is different from Add_1^U (Figure 8). After carrying this out, the newly selected address (Add_2^U) is sent to LA. At this point, the challenge–response mechanism starts. Therefore, LA sets a challenge on the instance C^{LA} and provides the client with all of the information necessary to solve this challenge (i.e., a random $\bar{R}$). Once received, the user just has to authorize the transaction on MetaMask in order to invoke the `solveChallenge` function of the smart contract (Figure 9).

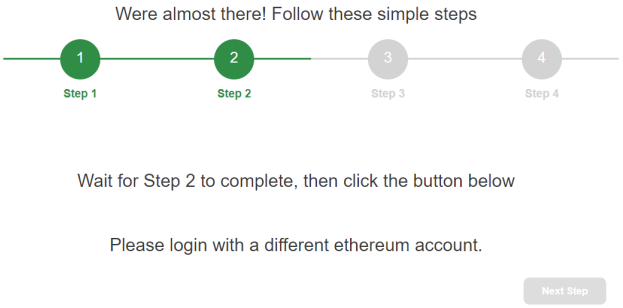

Figure 8. Selection of the Ethereum address Add_2^U.

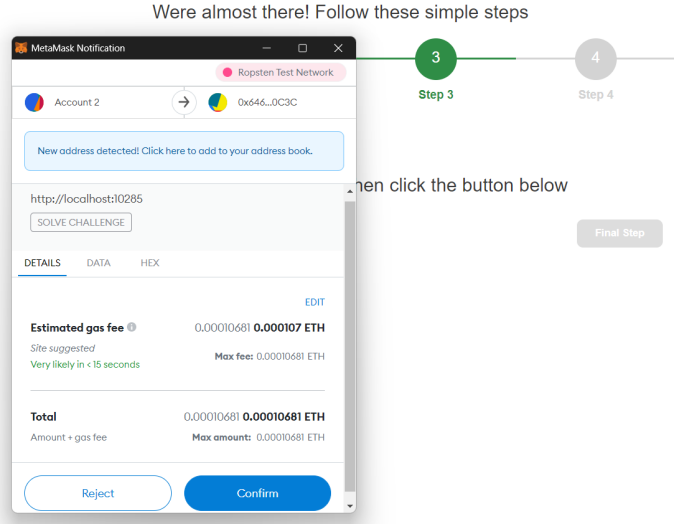

Figure 9. Transaction to solve the LA challenge.

Once this transaction is confirmed, the user can proceed to the fourth step, which consists of a message confirming that the whole procedure succeeded. At this point, LA will redirect the user to SP, forwarding the address Add_2^U (Figure 10).

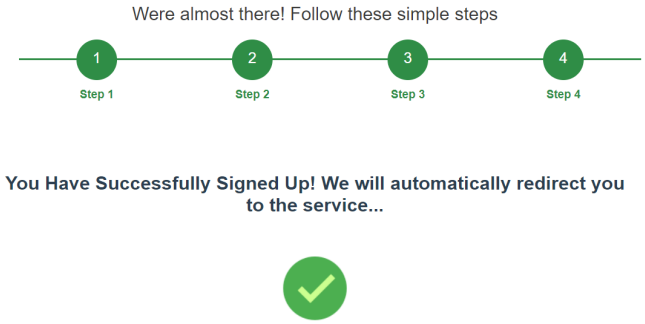

Figure 10. Completion of the interaction with LA.

On SP (Figure 11), after another challenge–response mechanism, the user can subscribe by entering a username and a password that will be memorized by SP along with the address Add_2^U.

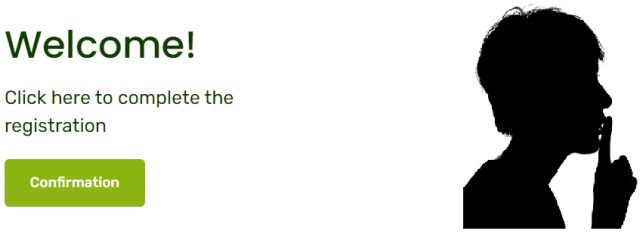

Figure 11. Subscription to SP.

This address is also used in the case where the user forgets their login credentials. Again, the procedure used to recover them consists of the application of the challenge–response mechanism. We recall that the user needs to select the right address to invoke the `solveChallenge` function of the smart contract. To aid the user in the selection of the right address, the client-side module of SP implements the same mechanism seen in the first step of the procedure executed with LA. We highlight that this design choice has two main benefits: (i) the user does not have to memorize Add_2^U and, (ii) it being a procedure performed exclusively client-side, the user's other addresses are not disclosed to the server.

6.3. Time-Cost Analysis

We analyzed our implementation by measuring the time taken to perform the operations required by our solution. The measurements were performed by using a personal computer equipped with a 2.8 GHz Intel i7-1165G7 CPU and 16 GB of RAM. The obtained results are reported in Figures 12 and 13.

In Figure 12, the interaction between the user, IG, and blockchain is represented. In Figure 13, the interaction between the user, LA, SP, and blockchain is reported.

We distinguished between the tasks the executed client-side (marked by CC) and the tasks executed server-side (marked by SC). The time intervals necessary to perform these two kinds of tasks are represented with a solid grey line. For each of these intervals, the time needed to complete it is reported in the figures.

We also represent the user-dependent time intervals (marked by UDT) with a grey dashed line. Since the duration of these time intervals depends on the time the users need to click the buttons, we assumed realistic average times of 2–3 s.

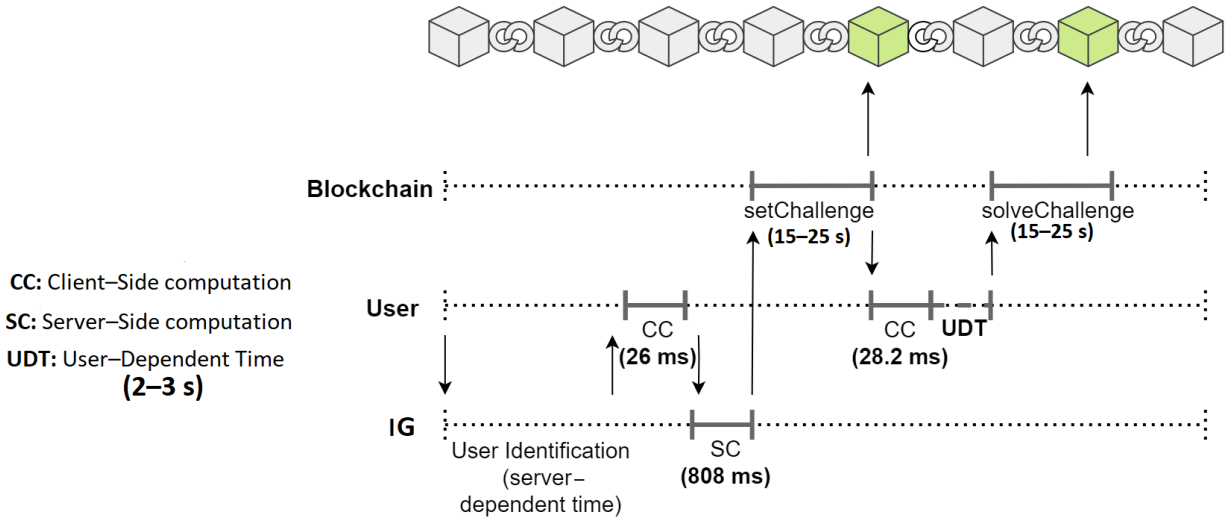

Figure 12. Timeline of the interactions between user, *IG*, and blockchain.

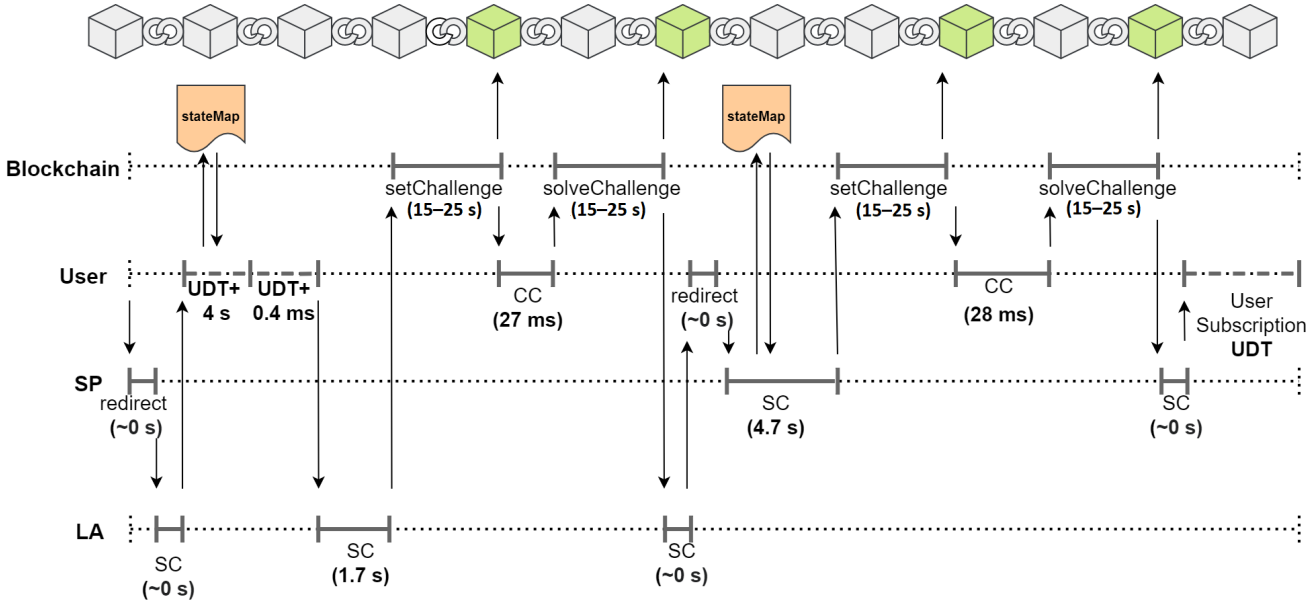

Figure 13. Timeline of the interactions between user, *LA*, *SP*, and blockchain.

Finally, the measurements of the time intervals between sending a transaction to the blockchain and confirming the transaction itself are reported in the two figures. To perform these measurements, we chose the *Ropsten* test network [44], since it is able to reproduce the network conditions of the live Ethereum MainNet.

Our measurements show that the time intervals pertaining to the blockchain timeline lasted 15–25 s. From our experiments, these values, compared to the duration of the other measured time intervals, appear to be predominant but still acceptable.

A factor influencing the duration of the time intervals pertaining to the blockchain is the selected gas price. Indeed, a high gas price causes a transaction to be processed faster, at the cost of greater transaction fees. For our experiments, we adopted the default gas price suggested by Metamask.

To summarize the results, we obtained that the entire registration procedure with *IG* requires less than 1 min, whereas the registration procedure with *SP* (including the interaction with *LA*) requires less than 2 min.

By using the default gas price suggested by Metamask, we also measured the costs, in terms of Ethers, resulting from the following operations: smart contract deployment, the execution of the `setChallenge` function, and the execution of the `solveChallenge` function.

In the following, we report the resulting costs in terms of Ethers (ETH) and US dollars in June 2022:

- Smart contract deployment: 0.00296997 ETH/USD 5.34;
- `setChallenge` execution: 0.00029962 ETH/USD 0.54;
- `solveChallenge` execution: 0.00013468 ETH/USD 0.24.

We stress that, although the smart contract deployment has a relatively high cost, it is a one-time operation performed by SP, LA, and IG, and therefore its cost is sustainable.

From the above results, we can estimate the overall cost of a user registration, which is approximately USD 2.34. In a real business model, this cost can be charged to the user willing to use the anonymous service. Therefore, our solution does not result in significant costs, neither for the user nor for the service providers.

7. Security Analysis

In this section, we discuss the security guarantees offered by our solution.

We analyze the following two *compromises* possibly affecting our protocol.

- C1: an adversary, not including the collaboration of IP, SP, and LA simultaneously, is able to discover the link between the username and the real identity of a user.
- C2: an adversary is able to forge a fake link, publicly verifiable on the blockchain, between the username and the real identity of a user.

Regarding C1, it represents a privacy compromise. As already discussed, our protocol allows for the de-anonymization of a username only when IP, SP, and LA collaborate. In the other cases, the real identity–username link should not be disclosed to anyone.

Concerning C2, we refer to attacks in which the adversary attempts to associate a username $user_U$ of a user U with an identity $ID_{\bar{U}}$ of another user $\bar{U}$ to falsely accuse the latter of an illegal behavior performed by U.

Regarding the trust required to the involved actors, we only make the following *assumption*.

- A: IP is *honest but curious*, in the sense that it legally performs the steps of the protocols but attempts to cause the two compromises C1 and C2.

In other words, A requires that IP does not swap the real identities of two users. This is a standard assumption of all of the identity management systems. Indeed, the role of an identity provider is just to certify the real identity of users. However, in our threat model, we allow IP to disclose the identities that it knows and show that this does not affect the privacy of the users.

No assumption is made on IG, LA, and SP. They can be considered fully malicious with respect to the considered compromises.

Now, we discuss in detail the two compromises and show how our protocol prevents them.

Compromise C1. This compromise occurs when the adversary identifies the real identity ID_U of a user U associated with a given username $user_U$.

Since external users just see the transactions originating from some Ethereum addresses on the blockchain, we consider SP, IG, IP, and LA as adversaries. They can access the blockchain too; then, they have at least the knowledge of all external users. By excluding the collaboration of IP, SP, and LA (by hypothesis), we consider the collaboration of the three tuples (IP, IG, LA), (IG, LA, SP), and (IP, IG, SP) as an adversary and show that C1 does not occur.

We start by considering the collaboration of IP, IG, and LA. Given the user U, IP knows the real identity ID_U of U. Through the collaboration with IG, IP links ID_U with the Ethereum address Add_1^U. Then, through the collaboration with LA, IP can discover the Ethereum address Add_2^U linked with Add_1^U. However, no information about the username $user_U$ of U is available to IP, IG, and LA. Indeed, the only public information containing $user_U$ (in obfuscated form) is the transaction T_6 generated by U with the address Add_2^U. This transaction contains the solution to the challenge $sol = Keccak256(\hat{R} \| user_U)$. Without

$\hat{R}$ (maintained by SP), it is not possible to also reverse the hash function `Keccak256` through dictionary attacks on all of the possible usernames.

A similar reasoning applies when considering the collaboration of IG, LA, and SP as an adversary. Their collaboration just allows for the linkage of Add_1^U, Add_2^U, and $user_U$, but no information is available to IG, LA, and SP about ID_U.

Indeed, the only public information containing ID_U (in obfuscated form) is the transaction T_2 generated by U with the address Add_1^U. This transaction contains the solution to the challenge $sol = Keccak256(R \| ID_U)$. Without R (maintained by IP), it is not possible to reverse the hash function `Keccak256`. $Keccak256(R \| ID_U)$ is the only information received by IG from IP, since the assertion does not contain any information linked to ID_U.

Finally, consider the adversary composed of the collaboration of IP, IG, and SP. IP, through IG, knows the mapping $Add_1^U - ID_U$, whereas SP maintains the mapping $Add_2^U - user_U$. However, without the mapping $Add_1^U - Add_2^U$ (maintained by LA), they are not able to link ID_U with $user_U$. The only public information containing this mapping (in obfuscated form) is the transaction T_4 generated by U. This contains the solution to the challenge $sol = Keccak256(\bar{R} \| Add_1^U \| Add_2^U \| \sigma)$. Again, without $\bar{R}$ and σ (maintained by LA), it is not possible to reverse the hash function `Keccak256`.

Compromise C2. This compromise occurs when an adversary forges a valid link (publicly verifiable on the blockchain) between the real identity $ID_{\bar{U}}$ of a user $\bar{U}$ and a username $user_U$ of a different user U.

To forge a valid link, $ID_{\bar{U}}$ has to be present (in obfuscated form) in a transaction intended for the instance C^{IG}. By Assumption A, IP legally performs the steps of the protocol and does not provide a tampered digest $Keccak256(R||\bar{U})$ associated with the real identity of a user $\bar{U}$ not requiring the anonymous service. Therefore, we consider $\bar{U}$ as a user with Ethereum address $Add_1^{\bar{U}}$ such that the link between $ID_{\bar{U}}$ and $Add_1^{\bar{U}}$ can be verified through the instance C^{IG} and the random R provided by IP.

To perform the compromise C2, the attacker has two possibilities. The first possibility is to forge a fake link between $Add_1^{\bar{U}}$ and Add_2^U verifiable through the instance C^{LA} such that Add_2^U is associated with $user_U$ by SP. To achieve this, the blockchain would have a transaction originated by Add^{LA} including the challenge $ch = Keccak256(Keccak256(\bar{R} \| Add_1^{\bar{U}} \| Add_2^U \| \bar{\sigma}))$, where $\bar{\sigma}$ is a signature obtained from the private key associated with the address $Add_1^{\bar{U}}$. Moreover, another transaction originated by Add_2^U, including the solution to the challenge, would be present on the blockchain. Even though the attacker coincides with the collaboration of LA and U (the only parties that can generate these two transactions), it is not able to forge the signature $\bar{\sigma}$ without the collaboration of $\bar{U}$. Therefore, this first case cannot occur.

The second possibility is that a valid link between $Add_1^{\bar{U}}$ and $Add_2^{\bar{U}}$ (both belonging to $\bar{U}$) exists and the attacker attempts to forge a fake link between $Add_2^{\bar{U}}$ and $user_U$. However, similar to the previous case, in order to accomplish this, the blockchain would have a transaction originated by Add^{SP}, including the challenge $ch = Keccak256(Keccak256(\hat{R} \| user_U))$, and a transaction originated by $Add_2^{\bar{U}}$, including the solution to the challenge. Even though the attacker coincides with the collaboration of SP and U, it is able to forge the first transaction but not the second. Therefore, this possibility also cannot occur.

This concludes the security analysis.

8. Conclusions and Discussion

Anonymous services have shown increasing popularity over the years. However, for most of these services, it is crucial to not just provide anonymity to the user but also to implement an accountability mechanism.

In this paper, we propose a blockchain-based approach as a solution to the accountability issue in anonymous services. Our solution requires that only the co-operation of three independent parties (IG, LA, and SP) can de-anonymize a user who is performing illegally on the SP platform. This fact represents the main advantage of our solution since it makes an illegitimate attempt to de-anonymize the user unlikely.

To prove that our solution can be applied in real-life contexts, we provide a prototype that takes into account usability aspects, as it aims to reduce the number of operations performed client-side. Moreover we provide a time and cost analysis of our solution in order to demonstrate its feasibility.

According to our cost analysis, the most expensive operation consists in the smart contract deployment in the Ethereum network (approximately USD 5). However, such an operation is meant to be performed only once by IG, LA, and SP. Hence, it is safe to assume that such a cost is sustainable. The other operation that they have to perform is `setChallenge`, whose cost is estimated to be USD 0.54. Even though this operation has to be performed for each user willing to subscribe to an anonymous service, its results are acceptable for the service providers. However, in a possible business model, the provider can charge the user for this cost. On the other hand, the cost to perform the `solveChallenge` operation is an equally low price (USD 0.24) and is meant to be paid directly by the user.

Considering instead the cost of a complete user registration (which requires the execution of both `setChallenge` and `solveChallenge`), it turns out to be very cheap (around USD 2). It is therefore realistic to assume that this cost can be charged to the user in order to not burden the providers.

Regarding the time analysis that we conducted, our study shows that the registration procedure with IG requires less than 1 min, whereas the registration procedure with SP and LA requires less than 2 min. It is worth noting that the time intervals related to the operations involving the blockchain are predominant (ranging from 15 to 25 s). However, despite this tolerable drawback, relying on the blockchain for these operations has the advantage of making them immutable and publicly verifiable. Therefore, untrusted parties (IG, LA, and SP) cannot hold other users accountable instead of the real misbehaving user. Furthermore, as the entire registration process is one time, the overall time taken to perform a complete registration is reasonable.

To conclude, our solution is shown to be usable and reasonably time and cost-efficient. As for the cost, we may argue that honest users might be encouraged to pay for a service that, in addition to anonymity, would guarantee them the possibility of reporting dishonest users who might take advantage of the anonymous service to threaten them. Moreover, the anonymous service provider itself might be encouraged to employ our solution since accountability guarantees can discourage users from engaging in illegal behaviour, thus improving the overall reputation of the anonymous service provider.

As a future work, we plan to further increase the technology readiness level of our research by proposing its application to real-life environments, possibly with industrial partners.

Author Contributions: Conceptualization, F.B., V.D.A. and S.L.; methodology, F.B., V.D.A. and S.L.; software, V.D.A. and S.L.; validation, F.B., V.D.A. and S.L.; formal analysis, F.B., V.D.A. and S.L.; investigation, F.B., V.D.A. and S.L.; resources, V.D.A. and S.L.; data curation, V.D.A. and S.L.; writing—original draft preparation, F.B., V.D.A. and S.L.; writing—review and editing, F.B., V.D.A. and S.L.; visualization, V.D.A. and S.L.; supervision, F.B.; project administration, F.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: This work has been developed in coordination with the activities of "iCARE" project (CUP J39J14001400007)-action 10.5.12.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

EOA	External Owned Account
U	User
SP	Service Provider
IG	Identity Gateway
LA	Linkage Agency
ECDSA	Elliptic Curve Digital Signature Algorithm
VM	Virtual Machine
UDT	User-Dependent Time
CC	Client-side Computation
SC	Server-side Computation
ETH	Ether
P2P	Peer-to-Peer
MVC	Model View Controller
JSP	JavaServer Page
AJAX	Asynchronous JavaScript and XML

References

1. Harkavy, M.; Tygar, J.D.; Kikuchi, H. Electronic auctions with private bids. In Proceedings of the USENIX Workshop on Electronic Commerce, Boston, MA, USA, 31 August–3 September 1998.
2. Liu, B.; Xie, S.; Yang, Y.; Wang, R.; Hong, Y. Privacy preserving divisible double auction with a hybridized TEE-blockchain system. *Cybersecurity* **2021**, *4*, 1–14. [CrossRef]
3. Hohenberger, S.; Myers, S.; Pass, R. ANONIZE: A large-scale anonymous survey system. In Proceedings of the 2014 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 18–21 May 2014; pp. 375–389.
4. Carnley, R.; Bagui, S. A Public Infrastructure for a Trusted Wireless World. *Future Internet* **2022**, *14*, 200. [CrossRef]
5. Wang, G.; Wang, B.; Wang, T.; Nika, A.; Zheng, H.; Zhao, B.Y. Whispers in the Dark: Analysis of an Anonymous Social Network. In Proceedings of the 2014 Conference on Internet Measurement Conference (IMC '14), Vancouver, BC, Canada, 5–7 November 2014; Association for Computing Machinery: New York, NY, USA, 2014; pp. 137–150. [CrossRef]
6. Gerhart, N.; Koohikamali, M. Social network migration and anonymity expectations: What anonymous social network apps offer. *Comput. Hum. Behav.* **2019**, *95*, 101–113. [CrossRef]
7. Hosseinmardi, H.; Han, R.; Lv, Q.; Mishra, S.; Ghasemianlangroodi, A. Analyzing negative user behavior in a semi-anonymous social network. *arXiv* **2014**, arXiv:1404.3839v1.
8. Farkas, C.; Ziegler, G.; Meretei, A.; Lörincz, A. Anonymity and accountability in self-organizing electronic communities. In Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society, Washington, DC, USA, 21 November 2002; pp. 81–90.
9. Buccafurri, F.; Lax, G.; Nicolazzo, S.; Nocera, A. Accountability-preserving anonymous delivery of cloud services. In *International Conference on Trust and Privacy in Digital Business*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 124–135.
10. Buccafurri, F.; De Angelis, V.; Lax, G.; Musarella, L.; Russo, A. An Attribute-Based Privacy-Preserving Ethereum Solution for Service Delivery with Accountability Requirements. In *ARES '19: Proceedings of the 14th International Conference on Availability, Reliability and Security*; Association for Computing Machinery: New York, NY, USA, 2019. [CrossRef]
11. Russo, A.; Lax, G.; Dromard, B.; Mezred, M. A System to Access Online Services with Minimal Personal Information Disclosure. *Inf. Syst. Front.* **2021**, 1–13. [CrossRef]
12. Camenisch, J.; Lysyanskaya, A. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 93–118.
13. Backes, M.; Camenisch, J.; Sommer, D. Anonymous yet accountable access control. In Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, Alexandria, VA, USA, 7 November 2005; pp. 40–46.
14. Teranishi, I.; Sako, K. K-times anonymous authentication with a constant proving cost. In *International Workshop on Public Key Cryptography*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 525–542.
15. Lysyanskaya, A.; Rivest, R.L.; Sahai, A.; Wolf, S. Pseudonym systems. In *International Workshop on Selected Areas in Cryptography*; Springer: Berlin/Heidelberg, Germany, 1999; pp. 184–199.
16. Damgård, I.B. Payment systems and credential mechanisms with provable security against abuse by individuals. In *Conference on the Theory and Application of Cryptography*; Springer: Berlin/Heidelberg, Germany, 1988; pp. 328–335.
17. Camenisch, J.; Van Herreweghen, E. Design and implementation of the idemix anonymous credential system. In Proceedings of the 9th ACM Conference on Computer and Communications Security, Washington, DC, USA, 18–22 November 2002; pp. 21–30.
18. Tsang, P.P.; Smith, S.W. PPAA: Peer-to-peer anonymous authentication. In *International Conference on Applied Cryptography and Network Security*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 55–74.

19. Dingledine, R.; Freedman, M.J.; Molnar, D. The free haven project: Distributed anonymous storage service. In *Designing Privacy Enhancing Technologies*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 67–95.
20. Durahim, A.O.; Savaş, E. A-MAKE: An efficient, anonymous and accountable authentication framework for WMNs. In Proceedings of the 2010 Fifth International Conference on Internet Monitoring and Protection, Barcelona, Spain, 9–15 May 2010; pp. 54–59.
21. Backes, M.; Clark, J.; Kate, A.; Simeonovski, M.; Druschel, P. BackRef: Accountability in anonymous communication networks. In *International Conference on Applied Cryptography and Network Security*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 380–400.
22. Diaz, C.; Preneel, B. Accountable anonymous communication. In *Security, Privacy, and Trust in Modern Data Management*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 239–253.
23. Niu, Y.; Wei, L.; Zhang, C.; Liu, J.; Fang, Y. An anonymous and accountable authentication scheme for Wi-Fi hotspot access with the Bitcoin blockchain. In Proceedings of the 2017 IEEE/CIC International Conference on Communications in China (ICCC), Qingdao, China, 22–24 October 2017; pp. 1–6.
24. Tsang, P.P.; Au, M.H.; Kapadia, A.; Smith, S.W. PEREA: Towards practical TTP-free revocation in anonymous authentication. In Proceedings of the 15th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 27–31 October 2008; pp. 333–344.
25. Maria, A.; Rajasekaran, A.S.; Al-Turjman, F.; Altrjman, C.; Mostarda, L. Baiv: An efficient blockchain-based anonymous authentication and Integrity Preservation Scheme for secure communication in VANETs. *Electronics* **2022**, *11*, 488. [CrossRef]
26. Black, E.W.; Mezzina, K.; Thompson, L.A. Anonymous social media–Understanding the content and context of Yik Yak. *Comput. Hum. Behav.* **2016**, *57*, 17–22. [CrossRef]
27. Chatzistefanou, V.; Limniotis, K. On the (non-) anonymity of anonymous social networks. In *International Conference on e-Democracy*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 153–168.
28. Bernabe, J.B.; Canovas, J.L.; Hernandez-Ramos, J.L.; Moreno, R.T.; Skarmeta, A. Privacy-preserving solutions for blockchain: Review and challenges. *IEEE Access* **2019**, *7*, 164908–164940. [CrossRef]
29. Treiblmaier, H. What Is Coming across the Horizon and How Can We Handle It? Bitcoin Scenarios as a Starting Point for Rigorous and Relevant Research. *Future Internet* **2022**, *14*, 162. [CrossRef]
30. Möser, M.; Soska, K.; Heilman, E.; Lee, K.; Heffan, H.; Srivastava, S.; Hogan, K.; Hennessey, J.; Miller, A.; Narayanan, A.; et al. An empirical analysis of traceability in the monero blockchain. *arXiv* **2017**, arXiv:1704.04299.
31. Hopwood, D.; Bowe, S.; Hornby, T.; Wilcox, N. *Zcash Protocol Specification*; GitHub: San Francisco, CA, USA, 2016; p. 1.
32. Johnson, D.; Menezes, A.; Vanstone, S. The elliptic curve digital signature algorithm (ECDSA). *Int. J. Inf. Secur.* **2001**, *1*, 36–63. [CrossRef]
33. Bertoni, G.; Daemen, J.; Peeters, M.; Assche, G.V. Keccak. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 313–314.
34. Pierro, G.A.; Tonelli, R.; Marchesi, M. An organized repository of ethereum smart contracts' source codes and metrics. *Future Internet* **2020**, *12*, 197. [CrossRef]
35. European Union. Regulation EU No 910/2014 of the European Parliament and of the Council. 23 July 2014. Available online: http://eur-lex.europa.eu/legal-content/EN/TXT\/HTML/?uri=CELEX%3A32014R0910&from=EN (accessed on 18 August 2022).
36. Boehm, O.; Caumanns, J.; Franke, M.; Pfaff, O. Federated authentication and authorization: A case study. In Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference, Munich, Germany, 15–19 September 2008; pp. 356–362.
37. Dingledine, R.; Mathewson, N.; Syverson, P. *Tor: The Second-Generation Onion Router*; Technical Report; Naval Research Laboratory: Washington, DC, USA, 2004.
38. Rossberg, M.; Schaefer, G. A survey on automatic configuration of virtual private networks. *Comput. Netw.* **2011**, *55*, 1684–1699. [CrossRef]
39. Roughley, I. Starting Struts 2. 2007. Available online: https://www.infoq.com/minibooks/starting-struts2/ (accessed on 18 August 2022).
40. Kurniawan, B. *Java for the Web with Servlets, JSP, and EJB*; Sams: Carmel, IN, USA, 2002.
41. Garrett, J.J. Ajax: A New Approach to Web Applications. 2005. Available online: https://immagic.com/eLibrary/ARCHIVES/GENERAL/ADTVPATH/A050218G.pdf (accessed on 18 August 2022).
42. Infura The World's Most Powerful Blockchain Development Suite. Available online: https://infura.io/ (accessed on 18 August 2022).
43. MetaMask. A Crypto Wallet & Gateway to Blockchain Apps. Available online: https://metamask.io/ (accessed on 18 August 2022).
44. Ropsten Testnet Explorer, 2006. Available online: https://ropsten.etherscan.io (accessed on 18 August 2022).