Recoverable Token: Recovering from Intrusions against Digital Assets in Ethereum

Filipe F. Martins, David R. Matos, Miguel L. Pardal, Miguel Correia
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
{filipe.f.martins,david.r.matos,miguel.pardal,miguel.p.correia}@tecnico.ulisboa.pt

***Abstract*—Blockchain systems allow storing digital assets in a tamper-proof, consensus-based, append-only ledger in a decentralized fashion, where no single party has full control. A blockchain is an immutable, append-only, log of transactions. Unfortunately, in some cases there is the need to undo transactions that result from intrusions, e.g., when the private keys of a wallet are stolen, when one of the transaction participants does not comply with what was agreed upon, or when smart contract vulnerabilities are exploited by attackers. There are also accidental scenarios, e.g., when private keys are lost leaving the associated digital assets inaccessible. Although there have been a few proposals which allow modifications to the blockchain, they break the basic guarantees they are supposed to provide. We propose an approach for wallet owners to recover from attacks against their digital assets and accidental loss, while still assuring fundamental properties of the blockchain technology. We implemented the mechanism for Ethereum / EVM.**

***Index Terms*—Intrusion Recovery, Blockchain, Digital Assets, Tokens, Ethereum**

I. Introduction

Blockchain technology has been gaining popularity in the last decade with the rise of interest in cryptocurrencies such as *bitcoin* [1] and *ether* [2]. The initial goal was to have fast and cheap monetary transactions without involving a trusted third party, a role that is currently performed by banks and other financial institutions. As the technology matured, more use cases were discovered in several fields such as healthcare [3], business processes [4], and educational certificates [5]. However the learning curve for using the technology is still relatively high for non tech-savvy users. A minimum requirement is to have a set of asymmetric key pairs which are used to sign transactions that are then submitted into the blockchain. Handling such keys may be complicated but wallet software is improving daily, so is accessibility and usability. This allows more and more users to make use of blockchain to perform money transfers and store both information and value.

A. Ethereum and Tokens

Ethereum [2] is a public blockchain that was announced in 2014. The main goal of Ethereum is to provide an open-ended decentralized platform that enables the development and use of *smart contracts* and *decentralized applications* with built-in economic functions. In contrast to Bitcoin which has a limited scripting language, Ethereum is designed to be a *programmable blockchain* that runs a virtual machine – the *Ethereum Virtual Machine* (EVM) – that is Turing complete. The EVM allows running low-level machine code in the form of EVM bytecode. Developers can program *smart contracts* using high-level languages such as Solidity[1] or Vyper[2], compile them into bytecode and deploy them on the Ethereum blockchain. These smart contracts are analogous to objects in object-oriented programming, as they have attributes that define their state and methods that allow changing that state. Essentially they are immutable programs that run deterministically in an EVM context and their execution is triggered through *transactions* (akin to method calls).

In the *Ethereum* blockchain, wallets store keys that provide access to accounts. These accounts are associated with *ether*, Ethereum's intrinsic cryptocurrency, handled at the protocol level, and optionally to *tokens* [6]–[8] that are handled at the smart contract level. Tokens are handled by smart contracts that are owned by one account, so they encompass a form of centralization.

Tokens are frequently used to represent private currencies or value (e.g., capital stock), although they may also serve other purposes, e.g., representing voting rights, collectibles, identity, ownership of resources or other types of digital assets. As of September 2020, the top 10 tokens implemented over the Ethereum blockchain hold a market capitalization of over $27 billion.[3]. With such large amounts of value being exchanged, security and recovery mechanisms are indispensable. There are a set of standard tokens that can be used in many contexts, e.g., ERC20 and ERC721 [6], [7].

When a wallet is used to create an account, it generates an asymmetric key pair and derives the account address from the generated public key. This address is what uniquely identifies the wallet owner in the Ethereum blockchain. In order to interact with the Ethereum network, the *private key* is used to digitally *sign* transactions and the corresponding *public key* is used to *verify* the integrity and authenticity of those transactions. Unfortunately, if the *private key* is *lost* then it is no longer possible to interact with the network using that specific account and all the resources linked to it such as *ether* or *tokens* become *permanently inaccessible*. This may happen for several reasons, from laptop/smartphone loss or theft, to ransomware that denies access to the user files. Most wallet software generates deterministic wallets [9] meaning that the

[1] Solidity documentation – https://solidity.readthedocs.io/en/latest/

[2] Vyper documentation – https://vyper.readthedocs.io/en/latest

[3] CoinMarketCap Top 100 Tokens by Market Capitalization – https://coinmarketcap.com/tokens/

978-1-7281-8326-8/20/$31.00 ©2020 IEEE

key pairs are derived from *seed phrases*: lists of 12 to 24 words that allow users to recreate both the public and private keys. However, the user may never store their seed phrases in paper or digitally, so they may be unavailable when they are needed for recovery purposes.

B. Blockchain Recovery

There is a considerable corpus of research on *intrusion recovery*, i.e., on undoing the effects of intrusions from the perspective of the user [10]–[16]. These works focus on removing the effects of an intrusion from the state of a system: mail server, web application, file system, etc.

In the blockchain domain, instead, *recovery mechanisms* are still scarce, something that may be an obstacle for the wider adoption of this technology. Recovery mechanisms in this context fall in two categories: *data redaction* [17], [18] and *transaction reversion* [19], [20]. *Data redaction* refers to removing data stored on the blockchain. This is useful in scenarios where data needs to be hidden or removed from the blockchain, e.g., references to illegal pornography or sensitive and private information. Nevertheless, this is not the approach we are interested in this work.

Transaction reversion is a form of *blockchain rollback*, where the goal is to either undo specific actions or moving the state of the blockchain to a previous point in time. Transaction reversion mechanisms allow for example the rightful owner to recover his funds in case an address has been compromised, e.g., because an attacker obtains access to the private key and proceeds to transfer all the funds to an address he owns.

When addressing issues such as losing access to a private key of an account, none of the previous works is fit to solve them without breaking fundamental properties of blockchains, specifically their *immutability*. This property means that it is possible to append blocks to the blockchain, but not to modify the blocks that are already part of the chain.

C. Our Approach

We present the first full solution for *recovering tokens* implemented in Ethereum. Ethereum smart contracts allow implementing arbitrary applications that may have different forms of operating and interacting with the external world (e.g., using clients that are not wallets [5]). Therefore, we do not aim to recover arbitrary applications, but tokens. Notice that although we often refer to Ethereum, our solution applies to other blockchains that run EVM (e.g., Ethereum Classic, TRON, Cardano, Ropsten) and private blockchains based on clients that run EVM (e.g., Quorum, Hyperledger Besu, Pantheon). Our intrusion recovery approach is even more generic and applies to many other blockchains.

Our approach involves a blockchain-based *dispute resolution mechanism* used to determine if a recovery request should be executed. To perform a *recovery action*, a claimant first has to submit a claim that becomes a dispute. If the claim is supported, the recovery action is executed. Our solution does not require any changes to the underlying blockchain protocol; it does not involve changes to the chain of blocks, so it does not break immutability. Instead, the recovery happens in a smart contract that contains the balance of tokens of each account.

The *Recoverable Token* we propose allows users to recover their Ethereum tokens in the following scenarios:

- *account loss* – user lost the access to the private key of an account and/or corresponding seed phrases and can no longer recover them;
- *account theft* – there is reasonable proof to believe that an account has been compromised;
- *chargebacks* – payment is made for any good or service and the payer believes that it did not receive what was agreed upon.

We implemented our approach in Solidity, the most used high-level EVM language, and thoroughly evaluated its performance on the Ropsten testnet. Our evaluation has shown that the mechanism allows doing recovery in a reasonable amount of time and at a reasonable cost, given the benefits of being able to do such an operation that is currently not supported in Ethereum or related blockchains.

The remainder of the paper is organized as follows. In Section II we give an overview of the architecture of the solution, describe its components and explain the recovery mechanism. Section III details the implementation and Section IV its evaluation. Finally in Section V we discuss related work and in Section VI we state our conclusions.

II. Recoverable Token

This section presents our approach in detail.

A. Recoverable Token Architecture

Figure 1 shows the architecture of the system, i.e., of the Ethereum blockchain with the Recoverable Token smart contracts and client (RCV App), users, and a storage service (IPFS). To interact with the system, users need *private keys*. The tokens owned by an account are stored and managed by the smart contracts deployed on the blockchain. These smart contracts are extended with the functionality of those that implement our recovery mechanism: *RCVToken*, *Claims*, and *Profiles*. Users also need to have access to an Ethereum node to send transactions (value transfers or smart contract calls) and propagate them to other nodes so that they may be validated and appended to the chain. The system will also make use of a decentralized, tamper-resistant, content-addressable, peer-to-peer storage network.

B. Attack Model

A *user* can be a *regular* user or an *arbitrator*. The regular users (that we often designate simply as users) are those entities that use the tokens provided by smart contracts. They own the tokens and can perform any actions just as they would if there was no recovery mechanism in the system. The recovery mechanism allows these users to submit claims that may escalate to disputes and in turn lead to account recoveries being performed. Arbitrators are special users who have permission to rule disputes.

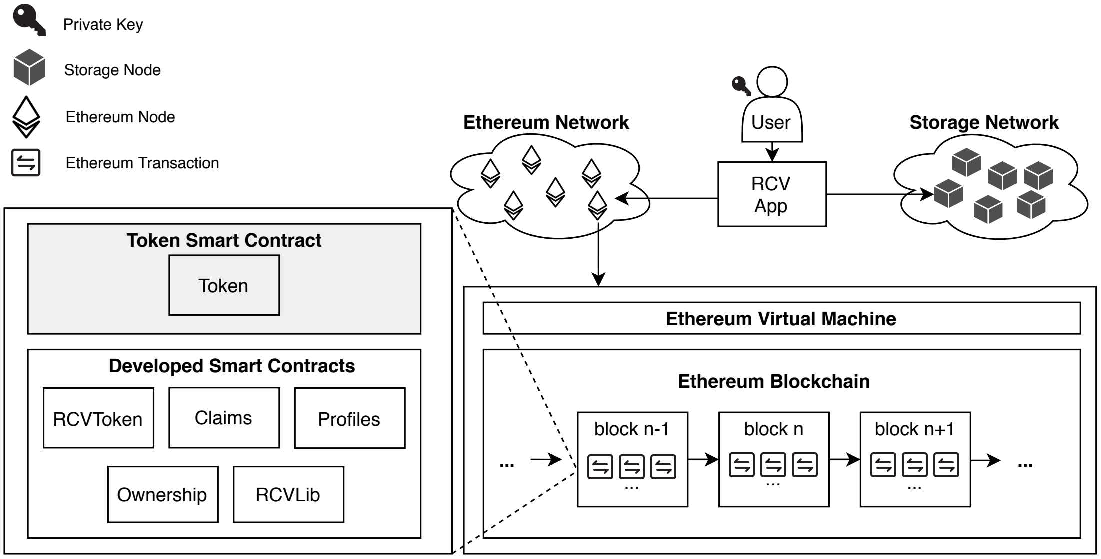

Fig. 1: Recoverable Token system architecture. A user which owns a private key is able to use the RCVApp to connect to a node of the Ethereum network and interact with the storage network. Each Ethereum node runs a local copy of an EVM and maintains a copy of the chain. The smart contracts that comprise the token recovery mechanism and the token which is to be recovered (shown in a grey background) are deployed on the blockchain.

Both regular users and arbitrators may positively contribute to the system or play against it. A user or arbitrator that does what it is supposed to is said to be *correct*, whereas one that deviates from that behavior is said to be *malicious*. Some possible attacks that users may try to perform against our mechanism include: submitting false claims; exploiting bugs; colluding with arbitrators. Arbitrators have the ability to rule on disputes, they may give dishonest rulings or none at all. We assume that less than a third (f) of the total amount of arbitrators (n) participating in a dispute are malicious, i.e., $n \geq 3f + 1$ (the same proportion as in common Byzantine fault-tolerant consensus algorithms [21]).

We do the usual assumptions that Ethereum works as expected and that cryptography is not compromised (e.g., no transactions can be issued on behalf of a user without his private keys).

C. Building Blocks

New features and standards for the Ethereum blockchain can be proposed through the submission of Ethereum Improvement Proposals (EIPs). These documents provide a technical specification and a rationale of the proposal. Our approach leverages some of these documents, although they are far from providing a full solution for our problem.

EIP1080 is a proposal for a standard for an interface used to implement a recoverable token [22]. At the time of this writing, the proposal is still up for discussion and no implementations exist.

EIP792 [23] and EIP1497 [24] propose interfaces for arbitration and evidence submission, where a group of chosen arbitrators make rulings on disputes. The claimant is also able to submit evidence to a decentralized storage such as IPFS [25] to support his claim and then link the evidence to the dispute.

D. Recoverable Token Dispute Resolution

In order to perform any recovery actions, first a *claim* has to be submitted, then escalated to a *dispute*, finally approved (or denied). This approval is the result of the dispute resolution mechanism. In the legal system there are several types of dispute resolution mechanisms such as lawsuits, mediation and arbitration. In the context of this paper, the one we will be focusing on is *arbitration*.

Arbitration is a method of resolving disputes outside the court of law, commonly used in commercial and consumer disputes [26], [27]. Generically the process consists in two parties that have a dispute to agree upon a group of arbitrators who ideally constitutes, as a group, an unbiased third party. Then the arbitrators, after analyzing the arguments and evidence provided by both parties, will make a decision in favor of one party or neither. A dispute involves the payment of a *fee* that is used to reward the arbitrators for their work, i.e., as an incentive for them to do their job.

The method we decided to use to choose arbitrators relies on address whitelisting. This means that there is a known group of arbitrators who are trusted by the community to resolve disputes by voting. A new arbitrator is able to join the group if all the current members agree on it and the same process is used to remove a member. There are more decentralized alternative methods to select arbitrators such as the one used by both Kleros [28] and Aragon [29] which essentially require users to stake their tokens so that they are allowed to arbitrate.

This mechanism is supported by an evidence submission standard proposal [24] that enables linking evidence to dis-

TABLE I: Public methods of the three contracts

Method Name	Description
RCVToken contract	
claimLost(lostAccount)	Reports the *lostAccount* address as being lost
cancelLostClaim()	Reports the current address as not being lost
reportStolen()	Reports the current address as being stolen. If successful the sender's tokens are frozen
chargeback(pendingTransferNumber)	Requests a reversal of the transfer number *pendingTransferNumber* on behalf of the sender
get/setPendingTransferTime(account)	Get/Set the time an account has to chargeback a transfer
get/setLostAccountRecoveryTime(account)	Get/Set the time account has to wait before a lost account dispute can start
submitMetaEvidence(claimID, metaEvidence)	Link a meta evidence URI to a claim
submitEvidence(claimID, evidence)	Link an evidence URI to a claim
signUp(transferTime, recoveryTime)	Sign up an account to allow recovery actions
addRecoveryInfo(recoveryAccount, proof, identity)	Submit the proof of ownership of the current account
Claims contract	
createLostClaim(claimant, lostAccount)	Create a new lost claim
createStolenClaim(claimant)	Create a new stolen claim
createChargebackClaim(claimant, pendingTransferID)	Create a new chargeback claim
voteOnClaim(claimID, vote)	Vote on claim number *claimID*
rule(claimID, ruling)	Enforce *ruling* on claim number *claimID*
Profiles contract	
appeal(disputeID, extraData)	Request an appeal for a dispute
appealCost()	Return the cost of requesting an appeal
appealPeriod()	Return the time window for appealing a ruling
arbitrationCost(extraData)	Return the cost of submitting a claim
currentRuling(disputeID)	Return the current ruling of a dispute
disputeStatus(disputeID)	Return the status of a dispute
isPendingTransfer(transferID)	Check whether transfer status is pending

putes. In our case, evidence is an URI that links to a file hosted on IPFS [25]. That file contains some form of argument towards the resolution of the dispute, i.e., towards the recovery being accepted. The processing of the documents is done by the human arbitrators, so the format of the files is opaque to the system. The linking between a claim and the file is then performed by executing a smart contract call such as *submitMetaEvidence* which will end up emitting an *event* that, in turn, is stored on the event log of the resulting transaction. Users are then able to query the blockchain for events emitted by the smart contract and retrieve all evidence related to a particular dispute.

Our dispute resolution mechanism starts with the creation of a new dispute by calling one of the *createLostClaim*, *createStolenClaim* or *createChargebackClaim* methods. Afterwards, arbitrators analyze the submitted evidence and commit to a ruling decision, or in other words, vote. Then, after at least two thirds plus one arbitrators ($2f + 1$) have voted, the *rule* method is called to get the final ruling based on the decisions of the participating arbitrators. This is the same proportion as in common Byzantine fault-tolerant consensus algorithms [21]; it ensures that the process does not stall waiting for malicious arbitrators (there are at most f malicious and $n \geq 3f + 1$) and a majority of the $2f + 1$ arbitrators are not malicious.

Next, there is another time window (known by calling *appealPeriod*) in which the claimant may appeal the decision. This can be done by calling the *appeal* method, which requires a fee known by calling *appealCost*. If no appeal is started or the appeal period is over, then the appropriate recovery action is performed according to the given ruling and a fee is split between the participating arbitrators.

Note that the dispute resolution and the evidence submission mechanisms adhere to the EIP792 and EIP1497 interfaces respectively, but their functionality goes beyond what these documents propose (they are not about recovery).

E. Intrusion Recovery

Now that we understand the components that make up the recovery mechanism we can start to describe how it can be used to address three problematic scenarios. Consider that Alice is the owner of a wallet – or individual address – that contains digital assets in the form of Ethereum tokens and Eve is an ill-intentioned user. The scenarios are:

- *S1*: Alice misplaced the seed phrase for her wallet and therefore has lost access to all the addresses stored within.
- *S2*: Eve obtains access to the private key of an Alice's address.
- *S3*: Alice pays Eve for a good or a service but Alice does not receive what she had paid for.

S1, S2 and S3 correspond respectively to account *loss*, account *theft* and *chargeback* cases (Section I).

In scenarios S1 and S2 the recovery mechanism requires the use of a *recovery account*. This account is set up by the user and it has the important role of receiving the lost or stolen tokens.

For *account loss* scenarios (S1), the protocol executed by the participating entities is shown in Figure 2. The process begins with the user submitting a loss claim (calling *claimLost*) to the RCVToken contract. This requires a fee to be paid; the value of the fee can be discovered by calling the *arbitrationCost* method in the Profiles contract (not shown in the figure).

```
/* MetaEvidence.json */
{
  fileURI:
    "ipfs://QmWRUgLu9iRk...",
  fileHash:
    "QmWRUgLu9iRk...",
  fileTypeExtension: ".txt",
  category: "Lost Claim",
  description: "I lost access to my address.",
  question:
   "Should the tokens be transferred
   to the specified recovery account?",
  rulingOptions: {
   type: "single-select",
   titles: ["Yes", "No"],
   descriptions: [
    "The account is indeed lost.
    Tokens will be transferred
    to the specified recovery account",
    "There is not enough proof to conclude
    that the account is lost.
    Tokens will remain in the account.",
   ],
  },
 }
```

Listing 1: Example meta evidence file

Moreover, it will call the *createLostClaim* method which creates a new loss claim which in turn will call *addNewClaim* which links it to the claimant's profile. Then, depending on the specific account configuration there is a time window (set with *setLostAccountRecoveryTime*), in which this claim can be cancelled (by calling the *cancelLostClaim* method using the account that is claimed to be lost). This is necessary in case of someone falsely claiming that a specific account is lost, by gaining access to a recovery account instead of the main account. This time window allows the owner of the address to deny the claim therefore proving that the account was in fact not lost. This is also not shown in the figure, as it considers the case in which the account recovery process is successful.

During this period the claimant has to submit a meta evidence file (calling *submitMetaEvidence*) that contains information regarding the context of the dispute and references an URI to a file which is the basis of the dispute (an example of the evidence file is in Listing 1). Without this the claim is unable to escalate to a dispute.

If the time window passes and the claim is not cancelled, then the dispute resolution mechanism begins. In a nutshell, dispute resolution will decide if the tokens should be transferred to a recovery account. This means that arbitrators can now vote on the dispute using the *voteOnClaim* method. The status of the claim is updated on each new vote. When the voting period ends, if two thirds of the arbitrators have voted one of them may call *giveRuling* to enforce the final ruling of the dispute and perform the recovery action which in this case is to transfer the tokens to the recovery account, otherwise the claim is cancelled. Although not shown in the figure, a subset of the voting period is reserved for appeals where the claimant is able to appeal the ruling before the arbitrators are capable of enforcing it.

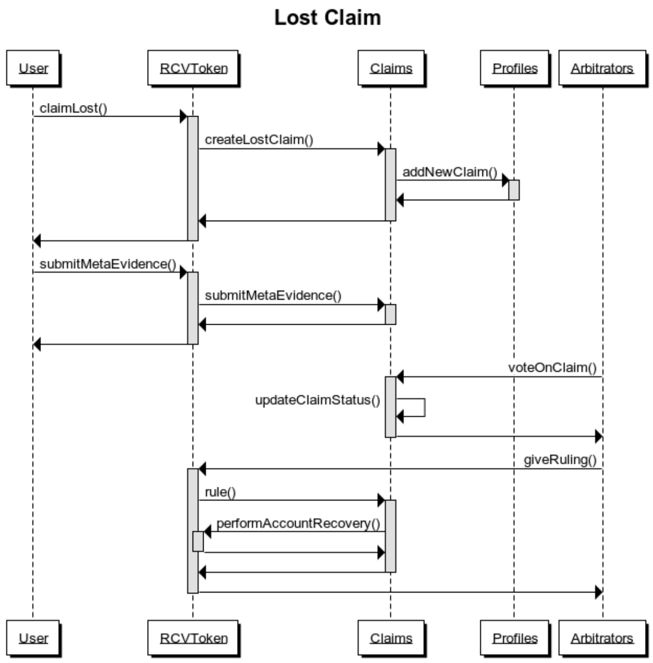

Fig. 2: Example of a successful lost claim (without an appeal)

In case of *account theft* (S2), the process is very similar with only some slight differences. In fact we do not present a diagram like Figure 2 because it would be very similar. First of all, only accounts that have a *proof of ownership* linked to them can be reported as stolen. The proof of ownership is the *digital signature* of a message that contains three elements: an *identity* and the *addresses* of the *account* and its corresponding *recovery account*. One difference when compared to the *account loss* scenario is that there is no time window in which the claim can be cancelled but instead the tokens owned by the account are *frozen*. After the claim is submitted and the meta evidence file is provided, the dispute resolution mechanism starts. Similarly to S1, the tokens are transferred to a recovery account linked to the one claimed to be stolen.

Finally, when dealing with *chargeback* scenarios (S3), a request is submitted to chargeback a *pending transfer*. Again we do not present a figure due to the similarity. A transfer is considered pending when the amount of time specified by the account configuration has not yet passed since the transfer was performed. As is usual, when the chargeback claim is submitted and the meta evidence is provided, the dispute resolution mechanism starts. In this case the tokens are returned to the account that first performed the transfer, i.e., the claimant, not the recovery account that does not need to exist.

The time complexity of all protocols is $O(1)$, i.e., there is a constant bound on the number of communication steps. The message complexity is $O(n)$, where n is the number of arbitrators.

F. Recoverable Token Application

As user interface (UI), we designed an application tailored to both regular users and arbitrators (RCV App in Figure 1). For a *regular user*, the application allows: signing up to the Recoverable Token system; generating and submitting proof of ownership; submitting new claims; tracking on-going claims and disputes; tracking pending transfers; submitting evidence; appeal the rulings given to disputes. For *arbitrators* the applications allows them to: view claims and disputes; access the evidence linked to disputes; rule (or vote) on disputes.

III. Implementation

In this section we delve deeper into details regarding the implementation.

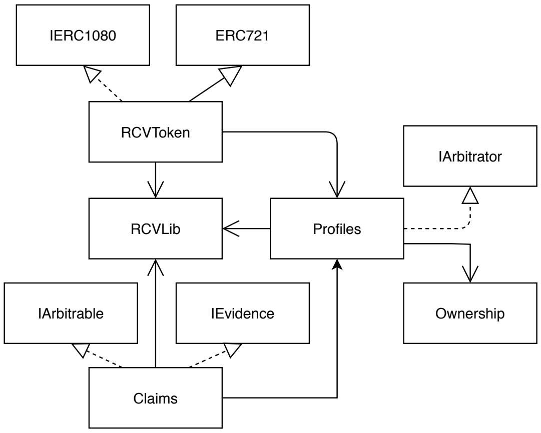

Fig. 3: Smart contracts used with inheritance relations

To implement the bulk of the Recoverable Token functionalities we developed a set of smart contracts written in the Solidity programming language. Figure 3 shows the inheritance relationships between the main types of contracts. *Profiles* is the smart contract that holds information about all the addresses that signed up to use the application.

It implements the IArbitrator interface of the EIP 792 [23]. Additionally, it references the *Ownership* library which has the necessary functions to validate the *proof of ownership* of an account.

The role of the *Claims* contract is to hold the information related to any sort of dispute, i.e., lost, stolen or chargeback claims. Through it arbitrators can vote on claims and enforce rulings by calling the methods shown in Table I.

Finally, the *RCVToken* contract implements the Recoverable Token interface and extends the token that is the target of recovery, e.g., an ERC20 or ERC721 token. A non-arbitrator user may perform the necessary actions to recover the tokens belonging to an account via the methods described in Table I. Depending on the type of token being extended, there is a need to modify the token transfer function so that the account freezing functionality may be added. An example of what modifications are required is seen in Listing 2.

```
/* RCVToken.sol */
function transferFrom(
    address from,
    address to,
    uint256 tokenID
) public override {
    RCVLib.Profile memory profile =
        _profiles.getAccountProfile(from);
    require(!profile.isClaimedStolen, "FROZEN");
    super.transferFrom(from, to, tokenID);
    uint256 transferNumber =
        _profiles.addTransfer(from, to, tokenID);
    emit PendingTransfer(
        from, to, tokenID, transferNumber
    );
}
```

Listing 2: Changes to transferFrom function

All the mentioned contracts share a library – *RCVLib* – that stores definitions of structures used across them. As for the UI (Section II-F), it is essentially a command line tool that allows interacting with the RCVToken contract.

TABLE II: Recoverable Token contracts' source code metrics

Contract	Deployed Bytecode	Gas	LoC
RCVToken	23919 bytes	5412412	186 lines
Claims	23411 bytes	5203042	372 lines
Profiles	13250 bytes	2948864	266 lines

We obtained source code metrics of the deployed smart contracts: *deployed bytecode*, which is the size of bytecode that is stored on-chain; *gas*, the amount of gas used to deploy the contract; *lines of code*, number of source code lines – excluding comments – after running a code formatter. Looking at Table II, we notice that the RCVToken contract has the highest size per lines of code ratio. This is due to the fact that it extends the contract that implements the ERC721 token which already has a size of 13867 bytes, therefore having an overhead of 10052 bytes. Furthermore, as is to be expected, the amount of gas increases linearly with the size of the deployed bytecode. We also recognized that the deployed bytecode size of the RCVToken contract needs to be addressed. The reason being that the hard cap put on the size of the objects that can be stored on-chain is equal to 24576 bytes according to EIP170 [30]. The security of smart contracts is an important concern today [31], [32], but our smart contracts are small. Moreover, the code can be checked using a code verification tool [33].

IV. Experimental Evaluation

To perform our evaluation we deployed the smart contracts on Ropsten, a public test network for Ethereum that to most extent mimics the Ethereum main network. This allowed us to request ether from publicly available faucets for free.

In the evaluation, we assess the cost in terms of both *time* (Section IV-A) and *gas* (Section IV-B) for all three main use cases of the system: trying to recover from the scenarios *S1*, *S2* and *S3* (Section I). All the metrics represented are the result of calculating the average of 20 claims performed in the Ropsten testnet. In every scenario the account to be recovered only holds one token and there are three arbitrators ruling on the claim.

A. Time consumption

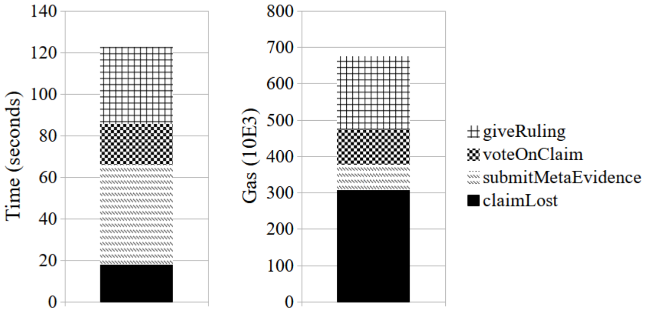

Fig. 4: Breakdown of lost claim dispute resolution

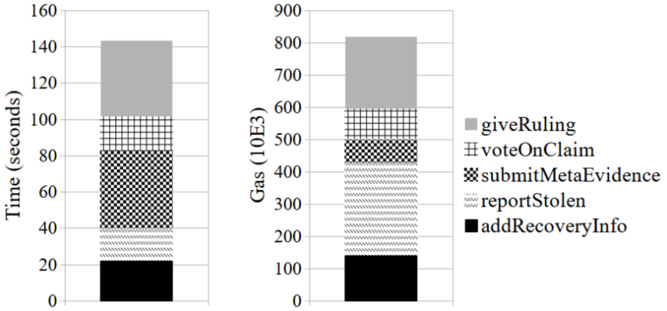

Fig. 5: Breakdown of stolen claim dispute resolution

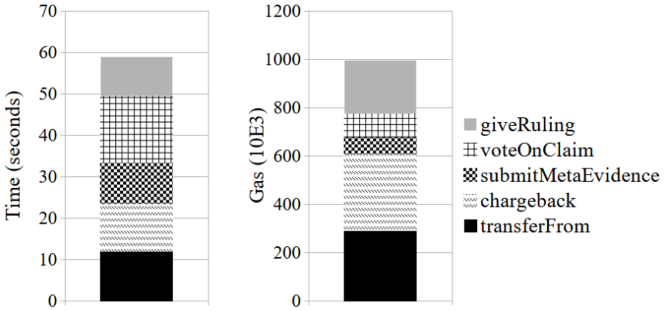

Fig. 6: Breakdown of chargeback claim dispute resolution

TABLE III: Time breakdown for the 3 dispute resolution cases

Action	Time (s)	Time (%)	σ
Lost claim dispute resolution			
claimLost	18.40	15.04	14.37
submitMetaEvidence	47.80	39.09	39.66
voteOnClaim	20.04	16.38	11.93
giveRuling	36.05	29.48	39.19
Stolen claim dispute resolution			
addRecoveryInfo	22.52	15.70	20.63
reportStolen	17.40	12.14	11.07
submitMetaEvidence	43.39	30.26	27.81
voteOnClaim	18.44	12.86	10.90
giveRuling	41.65	29.04	30.80
Chargeback claim dispute resolution			
transfer	12.03	20.42	7.44
chargeback	11.53	19.58	6.25
submitMetaEvidence	9.99	16.96	6.16
voteOnClaim	15.96	27.09	14.98
giveRuling	9.40	15.96	4.83

Figures 4, 5 and 6 include charts of the time data shown in Table III. In both the lost and stolen claim scenarios we notice that the *submitMetaEvidence* occupies the largest portion of the total time to complete each one. The reason for this is not that it is the most computationally expensive operation, quite the opposite as shown by its gas consumption charts which are also represented in the respective figures. Since its gas consumption is so low, miners do not prioritize it in the blocks they are trying to solve. This is a consequence of the proof-of-work consensus algorithm used in the Ropsten testnet. All the other actions take approximately 20 seconds to perform. In regards to the chargeback claim scenario, the network activity during the execution of the tests was much higher and therefore blocks were added to the chain at a faster rate (much closer to every 10 seconds). The calculated standard deviation supports this claim since as the network activity grows so does its stability. This is the reason for the standard deviation values in the chargeback scenario not resembling the ones in the other two. Additionally, note that these results are coherent with the fact that blocks in the Ropsten testnet are mined approximately every 10 to 20 seconds.

B. Gas consumption

Any transaction requires *gas* to be executed. Gas is a unit used in Ethereum to measure the computational effort of executing a transaction. When creating a transaction it is necessary to pay for the amount of gas that it will consume using ether. The sender of the transaction offers a value for each unit of gas. It is possible to estimate how much gas a transaction will spend since every instruction has a set gas cost.[4] While the amount of gas a transaction will require is mostly predictable, the price to pay for each unit of gas is not. It depends on different factors such as the number of pending transactions and the number of active miners and how fast you want it to be confirmed in the blockchain [34]. To determine how much the price will be in terms of a fiat currency (e.g., euros or dollars) the formula is:

$$gasPrice \times gasCost \times etherCost$$

where $gasPrice$ is the price of each unit of gas in ether, $gasCost$ is the amount of gas the transaction requires and $etherCost$ is the conversion rate from ether to the desired fiat currency. As the gas price offered by the sender increases so does the likelihood of a miner adding that specific transaction to the block it is mining because the reward he gets from doing so also increases.

[4] https://github.com/crytic/evm-opcodes

In relation to gas, we also notice that in each scenario the most computationally expensive operation is the one that is responsible for submitting the claim as shown in Figures 4, 5 and 6, as well and in Table IV. This corresponds to *claimLost* for lost claims, *reportStolen* for stolen claims and *chargeback* for chargeback claims. These operations are among the ones which do not have a constant gas cost, i.e. a non-zero standard deviation, which we believe is a result of the initialization and iteration of the data structures that store information about the different types of claims.

TABLE IV: Gas breakdown for the 3 dispute resolution cases

Action	Gas	Gas (%)	σ
Lost claim dispute resolution			
claimLost	310725	46.12	0
submitMetaEvidence	69514	10.32	778
voteOnClaim	95705	14.20	0
giveRuling	197834	29.36	0
Stolen claim dispute resolution			
addRecoveryInfo	144544	17.65	3
reportStolen	286530	35.00	3354
submitMetaEvidence	70107	8.56	0
voteOnClaim	94266	11.51	0
giveRuling	223371	27.28	0
Chargeback claim dispute resolution			
transferFrom	292092	29.34	18880
chargeback	315802	31.72	14300
submitMetaEvidence	71089	7.14	0
voteOnClaim	96278	9.67	215
giveRuling	220315	22.13	0

V. Related Work

A. Intrusion Recovery

The problem of intrusion recovery has been studied for different kinds of systems. In [10] the authors present a generic algorithm to recover from intrusions that works in three steps: a rewind step that rolls back the system to a point in time prior to the attack, a repair step in which the faulty operations are erased from the log, and a replay step to re-execute every operation in log. By the end of the third step the system no longer has the effects of the attack but it keeps every legitimate operation that occurred. This three-step algorithm was adopted in other works [11]–[16]. Our work aims to solve a similar problem by reverting the effects of undesired operations. However, the approach we use is more in line with the execution of compensating transactions [35], [36]. These kind of transactions aim to revert the effects of the intrusion without requiring the system to be rolled back to a previous point in time allowing the system to be recovered without shutting it down. A compensating transaction can be thought of as an inverse operation of the intrusion. Some systems that use this method of recovery are [13], [37]–[42].

B. Blockchain Recovery

Forks which are an expected occurrence as a result of the design of the blockchain may also be leveraged to perform blockchain recovery. One instance where a fork was used as a recovery mechanism was the response to The DAO hack [43]. Essentially a smart contract that implemented it had a vulnerability which allowed the attacker to steal over $50M USD worth of ether at the time. After a number of proposals the community as a whole voted for forking the chain to a state before the hack ever happened. As a result some community members which did not agree with the decision as they argued that it put into question the ledger immutability attribute of blockchains decided to continue with the original Ethereum chain which is now Ethereum Classic.

Removing or editing data [17], [18] from the blockchain has also been a topic of research. The general idea is to be able to modify confirmed blocks without breaking the links between them. This may be useful when dealing with removing references to illegal or unwanted information that was submitted to the blockchain, or to work towards making the blockchain GDPR [44] compliant.

For cases where a private key may have been stolen and transactions performed without the consent of the rightful owner, transaction reversion mechanisms such as Reversecoin [19] and more recently Blockd [20] have been proposed. Usually these types of work rely on replacing or cancelling transactions while they have not yet been submitted to or confirmed in the blockchain.

C. Dispute resolution

The idea to bring dispute resolution to the blockchain is being explored. Kleros [28] has been working on a decentralized arbitration application in which crowdsourced arbitrators give rulings on disputes. The arbitrators are expected to rule correctly and fairly as a result of game theoretical incentives.

Aragon [29], a software used to create and govern organizations on the Ethereum blockchain, has a component named Aragon Court that works similarly to Kleros but is limited to organizations in the Aragon Network whereas Kleros does not have that restriction.

Mattereum [45] is working on creating an infrastructure to build a layer to manage property or assets on-chain. As it is the case with several types of transfers, disputes may arise, thus a dispute resolution mechanism had to be developed. The approach taken was akin to the standard used in common arbitration courts. When a dispute is raised, either both parties had a predetermined agreement where the arbitrator had already been chosen, or alternatively and by default, an arbitrator is appointed from a panel of arbitrators.

VI. Conclusion

This paper presented Recoverable Token, a system that combines several standards in order to provide an opportunity for recovering digital assets stored on the Ethereum blockchain without modifying its fundamental properties.

To evaluate our system we applied it to an application that implements an ERC721 token and demonstrated that it is possible to recover the tokens in a variety of scenarios.

We concluded by discussing work related to intrusion recovery, blockchain recovery and dispute resolution on the blockchain.

Acknowledgements This research was supported by the European Commission under grant agreement number 822404 (QualiChain) and by national funds through Fundação para a Ciência e Tecnologia (FCT) with reference UIDB/50021/2020 (INESC-ID).

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, online.

[2] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, 2014.

[3] C. McFarlane, M. Beer, J. Brown, and N. Prendergast, "Patientory: A healthcare peer-to-peer EMR storage network v1," *Entrust Inc.*, 2017.

[4] P. Snow, B. Deery, J. Lu, D. Johnston, and P. Kirby, "Factom: Business processes secured by immutable audit trails on the blockchain," *Whitepaper*, Nov. 2014.

[5] D. Serranito, A. Vasconcelos, S. Guerreiro, and M. Correia, "Blockchain ecosystem for verifiable qualifications," in *Proceedings of the 2nd IEEE Conference on Blockchain Research & Applications for Innovative Networks and Services*, Sep. 2020.

[6] F. Vogelsteller and V. Buterin, "EIP 20: ERC-20 Token Standard," last accessed: 2020-06-16. [Online]. Available: https://eips.ethereum.org/EIPS/eip-20

[7] W. Entriken, D. Shirley, J. Evans, and N. Sachs, "EIP 721: ERC-721 Non-Fungible Token Standard," last accessed: 2020-06-13. [Online]. Available: https://eips.ethereum.org/EIPS/eip-721

[8] A. M. Antonopoulos and G. Wood, *Mastering Ethereum: building smart contracts and dapps*. O'Reilly Media, 2018.

[9] V. Buterin. (2013) Deterministic wallets, their advantages and their understated flaws. Last accessed: 2020-06-13. [Online]. Available: https://bitcoinmagazine.com/articles/deterministic-wallets-advantages-flaw-1385450276

[10] A. B. Brown and D. A. Patterson, "Rewind, repair, replay: three r's to dependability," in *Proceedings of the 10th ACM SIGOPS European Workshop*, 2002, pp. 70–77.

[11] A. B. Brown and D. A. Patterson, "Undo for operators: Building an undoable e-mail store," in *USENIX Annual Technical Conference*, 2003, pp. 1–14.

[12] D. Nascimento and M. Correia, "Shuttle: Intrusion recovery for PAAS," in *IEEE 35th International Conference on Distributed Computing Systems*, 2015, pp. 653–663.

[13] D. Matos and M. Correia, "Nosql undo: Recovering NoSQL databases by undoing operations," in *IEEE 15th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2016, pp. 191–198.

[14] A. Goel, K. Po, K. Farhadi, Z. Li, and E. De Lara, "The Taser intrusion recovery system," in *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, 2005, pp. 163–176.

[15] T. Kim, X. Wang, N. Zeldovich, and M. F. Kaashoek, "Intrusion recovery using selective re-execution," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, 2010, pp. 89–104.

[16] F. Hsu, H. Chen, T. Ristenpart, J. Li, and Z. Su, "Back to the future: A framework for automatic malware removal and system repair," in *IEEE 22nd Annual Computer Security Applications Conference*, 2006, pp. 257–268.

[17] G. Ateniese, B. Magri, D. Venturi, and E. Andrade, "Redactable blockchain–or–rewriting history in bitcoin and friends," in *2017 IEEE European Symposium on Security and Privacy*, 2017, pp. 111–126.

[18] D. Deuber, B. Magri, and S. A. K. Thyagarajan, "Redactable blockchain in the permissionless setting," in *2019 IEEE Symposium on Security and Privacy*, 2019, pp. 124–138.

[19] O. N. Challa. (2014) Reversecoin: Worlds First Cryptocurrency With Reversible Transactions. Last accessed: 2020-06-13. [Online]. Available: https://docs.google.com/document/d/1hMCkEQUYm9oFCQpxtIWFqVpt66pTQn1zCDW8WX0b7hw/

[20] R. M. Forster. (2019) Blockd.co In-Depth: Features and Future. Last accessed: 2020-05-17. [Online]. Available: https://medium.com/blockd/blockd-co-in-depth-under-the-hood-and-into-the-future-c142d6d54777

[21] M. Correia, "From byzantine consensus to blockchain consensus," in *Essentials of Blockchain Technology*. CRC Press, 2019, ch. 3.

[22] B. Leatherwood, "Ethereum Improvement Proposal 1080: Recoverable Token," last accessed: 2020-06-13. [Online]. Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1080.md

[23] C. Lesaege, "EIP 792: ERC-792 Arbitration Standard," last accessed: 2020-06-13. [Online]. Available: https://github.com/ethereum/EIPs/issues/792

[24] S. Vitello, C. Lesaege, and E. Piqueras, "EIP 1497: ERC-1497 Evidence Standard," last accessed: 2020-06-13. [Online]. Available: https://github.com/ethereum/EIPs/issues/1497

[25] J. Benet, "IPFS-content addressed, versioned, P2P file system," *arXiv preprint arXiv:1407.3561*, 2014.

[26] F. Elkouri, E. A. Elkouri, and K. May, *How arbitration works*. Bureau of National Affairs Washington, DC, 1973.

[27] S. Mentschikoff, "Commercial arbitration," *Columbia Law Review*, vol. 61, no. 5, pp. 846–869, 1961.

[28] C. Lesaege and F. Ast, "Kleroterion, a decentralized court for the internet," Jun. 2017.

[29] L. Cuende and J. Izquierdo, "Aragon network: A decentralized infrastructure for value exchange," *Whitepaper*, vol. 24, p. 2018, 2017.

[30] V. Buterin, "EIP 170: Contract Size Limit." [Online]. Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-170.md

[31] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 254–269.

[32] A. Mavridou, A. Laszka, E. Stachtiari, and A. Dubey, "VeriSolid: Correct-by-design smart contracts for Ethereum," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 446–465.

[33] T. Durieux, J. F. Ferreira, R. Abreu, and A. P. C. Monteiro, "Empirical review of automated analysis tools on 47,587 Ethereum smart contracts," in *42nd International Conference on Software Engineering (ICSE 2020)*, Dec. 2019.

[34] G. A. Pierro and H. Rocha, "The influence factors on ethereum transaction fees," in *IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain*, 2019, pp. 24–31.

[35] H. F. Korth, E. Levy, and A. Silberschatz, *A formal approach to recovery by compensating transactions*. University of Texas at Austin, Department of Computer Sciences, 1990.

[36] P. Liu, P. Ammann, and S. Jajodia, *Rewriting Histories: Recovering from Malicious Transactions*. Springer US, 2000, pp. 7–40.

[37] X. Xiong, X. Jia, and P. Liu, "Shelf: Preserving business continuity and availability in an intrusion recovery system," in *Proceedings of the Annual Computer Security Applications Conference*, 2009, pp. 484–493.

[38] İ. E. Akkuş and A. Goel, "Data recovery for web applications," in *Proceedings of the 40th IEEE/IFIP International Conference on Dependable Systems and Networks*, 2010, pp. 81–90.

[39] D. R. Matos, M. L. Pardal, and M. Correia, "Rectify: black-box intrusion recovery in paas clouds," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. ACM, 2017, pp. 209–221.

[40] D. R. Matos, M. L. Pardal, and M. Correia, "RockFS: Cloud-backed file system resilience to client-side," in *Proceedings of the 2018 ACM/IFIP/USENIX International Middleware Conference*, 2018.

[41] P. Ammann, S. Jajodia, and P. Liu, "Recovery from malicious transactions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 5, pp. 1167–1185, 2002.

[42] R. Chandra, T. Kim, M. Shah, N. Narula, and N. Zeldovich, "Intrusion recovery for database-backed web applications," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, 2011, pp. 101–114.

[43] C. Jentzsch, "Decentralized autonomous organization to automate governance," *Whitepaper*, Nov. 2016.

[44] R. Viorescu *et al.*, "2018 reform of eu data protection rules," *European Journal of Law and Public Administration*, vol. 4, no. 2, pp. 27–39, 2017.

[45] The Mattereum Team, "Mattereum protocol: Turning code into law," 2018. [Online]. Available: https://mattereum.com/wp-content/uploads/2020/02/mattereum-summary_white_paper.pdf