CIRCLE RESEARCH

Recoverable Wrapper Tokens

This paper introduces the recoverable wrapper token as a configurable mechanism to protect any ERC20 token from thefts, hacks, and accidental transactions.

Authors
Kaili Wang, Erik Tierney, & Gordon Liao

November 2023

Contents

Abstract

The recoverable wrapper token (RWT) can wrap around ERC-20 tokens to support asset recovery within a limited time window after an asset transfer.

It is designed to reduce theft and losses on the blockchain by allowing a victim to recover their stolen or lost assets during a recovery window (e.g., 24 hours). When an honest recipient receives the RWT, they must wait until the recovery window elapses, before they can unwrap the RWT back to its base ERC20 form. Our work builds upon prior research from Stanford on recoverable tokens. Here we describe a different version, its configurable features, the interface for interoperability, and potential forms of use, alongside a reference implementation.

Introduction

Annual losses due to token theft and accidental losses are in the billions of dollars: in 2021, $3.3 billion was stolen in crypto hacks, and that number jumped to $3.8 billion in 2022.

Funds lost or stolen are often irrecoverable due to the irreversible nature of the blockchain. This is a major reason why many retail users are hesitant to use blockchain systems today. Institutional adoption and regulatory support for the use of cryptographically-secured tokens as alternative payment instruments also hinge on the ability to reduce cyber and operational risks, as highlighted by a recent report by an international payment standard setting body.

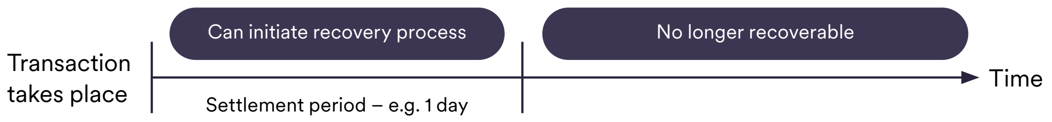
One way to protect assets from theft is to strengthen the security of asset keys and to improve the quality of Web3 code. A complementary approach, explored in depth by Wang, Wang and Boneh last year, is to extend the ERC-20 interface to support asset recovery with the "ERC-20R" framework. This paper focuses on a token wrapper approach that reuses several features from the ERC-20R proposal. We describe this token wrapper as a configurable and programmable recovery mechanism for developers and users.

ERC-20R Fundamentals

First, we review the premise of ERC-20R, which we build upon in our research. Recoverable ERC-20, called ERC-20R, gives the victim of an illegitimate transaction a short time window, e.g. 24 hours, to initiate a recovery process. Every transfer of the ERC-20R token is subject to a time window. The recoverable period is the one non-optional aspect of the ERC-20R token.

The length of the recovery window, however, is configurable. The developer can set this as a variable:

recoverableWindow: the time window during which a transferred token is recoverable (e.g. 24 hours).



If, say, Alice believes that a transaction transferring recoverable tokens from her account was illegitimate, she has 24 hours to initiate the recovery process (more details on this in the next section.) The recovery process can be set by the recoverable token issuer, and we provide some examples later in this paper.

An interesting consequence of the recovery window is that Alice's ERC-20R balance is split in two: settled tokens S and unsettled tokens U. Her balance in the ERC-20R contract is the sum of S and U. When a transaction deposits funds into Alice's account, the funds are initially deposited into her unsettled balance U. During the recovery window (say, 24 hours) these funds are subject to a potential recovery. Once the window elapses, the funds are irrecoverable, and they are transferred from Alice's unsettled balance U to her settled balance S. We stress that settled tokens cannot be clawed back from Alice by a recovery process. This is important because in many cases, the receiver requires a strong guarantee at some point in time (e.g. for bridging, redemptions, etc.).

It should be noted that there are two stages of settlement finality: settlement as a U token, then settlement as an S token. When settled as a U token, it has chain finality in that the transfer cannot be revoked via chain consensus, but it is still revocable by the governance. When settled as an S token, it has both chain finality and governance finality. For simplicity, we generally use "settlement" in this work to refer to the second stage.

Since ERC-20R was introduced, several projects have been built on it. A few examples include the Novospace project, the Resolve project, as well as some recovery UX proposals.

The Recoverable Wrapper Token (RWT)

The core idea underlying ERC-20R works best as a wrapper, rather than a standalone token.

First, it makes recoverability an optional feature to add to existing assets. Wrapping can be thought of as “protecting” the asset. Second, for a given base asset, different parties may need different configurations. Some may want a longer or shorter recoverable period. Similarly, different parties may want different arbitration systems for the same asset. Third, wrapping provides a much easier deployment path as there is no need to change the base asset.

Wrapping and unwrapping

When a base ERC-20 asset gets wrapped, a settled RWT is minted while the base asset is locked in the ERC-20 contract. Going the other way, anyone can unwrap a settled RWT with no delay, and this transfers a base ERC-20 token to the caller. An unsettled RWT cannot be unwrapped until it becomes settled.

Why is there no delay when unwrapping a settled RWT? Recall that settled RWTs cannot be clawed back from their owner. Hence, the only risk is that the current owner of the tokens has been compromised, and the attacker is trying to evade recovery by unwrapping the tokens while they are in the owner’s possession and then transferring the unwrapped tokens to the attacker’s address. If the owner is a DeFi contract, then the attacker would need to cause the contract to call the *unwrap* function, and this is typically not possible when a bug is being exploited or an oracle is being manipulated. Moreover, since most DeFi contracts do not ever need to unwrap tokens themselves (they would transfer them to others in wrapped form), a newly created contract can call the *disableUnwrap* function to prevent any future unwraps by the contract. Another argument for immediate unwrap of unsettled tokens, unrelated to DeFi use cases, is the far better user experience compared to a 24-hour delayed unwrap.

The Recoverable Wrapper Interface

Some level of composability is needed for easier adoption of RWT, so an interface is needed. The interface has the same six functions as the IERC-20 interface, except that most of these functions take an additional boolean flag as input that indicates whether the action should be applied to only the settled portion of the balance or any portion of the balance. For example, here is the interface for two functions:

```
function balanceOf(address account, bool includeUnsettled) external
view returns (uint256);

function transfer(address to, uint256 amount, bool includeUnsettled)
external returns (bool);
```

The recoverable wrapper interface is composable with ERC20 in that one can still call these functions without the boolean parameter; it would effectively assume that `includeUnsettled` is false. This way, anyone can call it as if it's simply an ERC20, and the function would only touch their settled balance. This makes RWTs interoperable with applications and wallets that don't specially support them.

One interesting note is that if wallets were to use RWTs and their UX does not support calling functions with the extra parameter, the RWT acts as a slow transfer. Users can only spend their settled funds, and sent funds don't show up in the recipient's balance until after the settlement window. One could also easily implement the RWT with these rules in the contract instead of relying on wallet UX patterns, as we will see in the "Configurable Features" section.

The interface also supports the following additional functions:

`function baseToken() external view returns (address);`
Returns the address of the base ERC-20 contract.

`function wrap(uint256 amount) external;`
Withdraws base ERC20 tokens from the caller's address and adds the same number of tokens to their settled RWT balance.

`function unwrap(uint256 amount) external;`
Removes tokens from the caller's settled RWT balance and sends them the same number of base ERC20 tokens.

`function unwrapTo(uint256 amount, address to) external;`
Removes tokens from the caller's settled RWT balance and sends the same number of base ERC20 tokens to the specified address.

`function nonce(address account) external view returns (uint256);`
Returns the current nonce associated with an account, which increments by one every time the account receives or sends this RWT. This may be useful for other contracts that use RWTs, such as the decentralized insurance-like R-Pool described in our next paper.

`function disableUnwrap() external;`
If implemented, this causes all subsequent unwrap requests from the caller's account to fail. Once unwraps for an account are disabled, it cannot be re-enabled. This is an optional security measure for addresses (e.g. contracts) that only send out wrapped tokens and do not need to unwrap their RWTs themselves. Note that if they need to have their RWTs unwrapped in the future, they can send them to a different address to unwrap.

By definition, the RWT would also support functions that allow recovery; however, we do not list them as part of the interface because recovery methods can vary widely, and they are not needed to allow composability between tokens.

04

Configurable Features

A key point is that the RWT can be designed in whatever way fits the developer's needs, based on its implementation of the interface. Listed below are a few components the developer could configure. Keep in mind there are many more ways to tailor the RWT contract to specific needs, as long as it can be implemented as a smart contract.

Length of recoverable window

The recoverable window length is configurable. Each transfer could just be recoverable for as little as the block time (~12 seconds) or as long as multiple days, depending on the use case.

Base token

The base token that is being wrapped can be any ERC20 token.

Transferability of unsettled tokens

Unsettled RWTs can be spendable or unspendable. Let us consider each scenario.

Option 1: Unsettled tokens are transferable
When Alice sends RWTs to Bob, she can choose to send them from either her settled balance or her unsettled balance. Either way the tokens are deposited into Bob's unsettled balance. Bob can choose to send the just-received unsettled RWTs to Carol, then Carol can send them to David, and so on. If Alice initiates a recovery process within 24 hours of sending the tokens to Bob, the funds would be traced to the current owners of the stolen funds (say, David). The original paper describes an on-chain discovery algorithm implemented within the freeze function, but this could also be done off-chain or manually, according to the rules set by the RWT issuer.

Option 2: Unsettled tokens are non-transferable
Alice can only spend from her settled balance, not her unsettled balance. Once she sends them to Bob, Bob must wait for the recovery window to elapse before spending or unwrapping them. The recovery process is simpler than in Option 1, because the funds would only be recovered directly from the initial recipient.

Each of these options has pros and cons. Option 1 gives receivers more instant utility to their tokens which is necessary in DeFi, but that utility is limited by others who may not be willing to take unsettled funds, and the recovery process is more complicated. Option 2 is simpler in the recovery process, though also slowing down the transfer, perhaps making it more fitting for retail users.

Recovery process mechanics

In the original ERC-20R paper, the recovery mechanics involved two steps: 1) Freeze, 2) Recover. The freezing buys time to gather evidence, evaluate whether the recovery should take place, without the fear of the funds passing through too many accounts (assuming unsettled tokens are transferable). Alternatively, a recoverable token could leave out the freeze step and just allow for the recovery step.

If unsettled tokens are transferable, there are several ways to fairly trace down the path of stolen funds. The original paper lists one such algorithm that is entirely on-chain; however, one could imagine an algorithm executed off chain and then verified on-chain.

Arbitration: Self vs Third-party

Who is responsible for deciding whether to recover or freeze? A few potential configurations:

- A designated address, which represents a smart contract or a centralized account. This address is the only one that can recover or freeze funds. A smart contract could represent a decentralized governing power; a centralized authority could be the RWT issuer, or third-party entity (e.g. Visa). The contract effectively allows the third party to provide an arbitration service within the defined period of recoverability.
- The sender of the contested transaction. This can be seen as more of a "clawback" or "undo button." This configuration can be useful for reversing fat finger transactions shortly after initiation.
- A multisig agreement between the sender and the receiver. This could be used as an escrow situation, if funds need to be set aside in a neutral place until an agreement is reached between both parties.

05

Examples of Configuration Sets

Using the above configurations, developers can create different templates of RWTs that serve different purposes. Below are two examples, and we encourage the reader to think of other configurations and use cases.

#1: Arbitrated wrapper designated for a pre-defined ecosystem

Recovery window (suggested)	Unsettled tokens are transferable?	Recovery mechanics	Arbitration
~1 day	yes	Freeze - recover	governanceAddress (contract or EOA)

This configuration allows recipients of RWTs to be able to spend them immediately without waiting for them to settle, without sacrificing the sender's ability to recover the funds. However, since illicit funds could theoretically pass through an exponential number of parties (Bob sends to 5 of his friends, who each send to 5 more of their own friends), self-arbitration is not an option. Arbitration should be done by some governing power, which depends on the use case.

Use cases
Any designated ecosystem where senders may not trust the self-custodied receiver, or parties are unidentified. The RWT must be used within a specific ecosystem since it is arbitrated by an implicitly designated party when users opted into the RWT.

Examples:

- **DeFi protocol protecting itself from hacks.** When a DeFi service receives ERC-20 tokens, it would be wise to immediately wrap the tokens in a RWT that they themselves issue and govern. This protects the service's token pool from bugs and hacks that could drain the pool. In case of a hack, the service operator could request a freeze and begin the recovery process. When the DeFi service sends tokens from its pool to another account, it sends them as RWTs. If the transfer was caused by a hack, it could get the tokens back through a recovery process. In other words, DeFi services that use RWTs introduce an asymmetry: they only accept base ERC-20 tokens, but send out wrapped tokens. In expectation of recoverability, the DeFi protocol would also be less likely a target of hacks and exploits as recoverability deters exploit attempts. In our next paper on R-Pools, we discuss how they could unwrap them immediately for a price.

- **A P2P ecosystem or two-sided marketplace operated by a trusted entity,** such as Uber, Ebay, a Visa-arbitrated self-custodial payments network, etc. If a sender believes they have been scammed, they initiate a recovery request to the trusted operator with any additional evidence. The operator could then freeze the funds, buying time to adjudicate over the case. The sender and receiver may not trust each other, but they trust the operator for fair dispute resolution.

This type of wrapper is implemented here as an example.

#2: Cancellable-by-sender delayed transfer

Recovery window (suggested)	Unsettled tokens are transferable?	Recovery mechanics	Arbitration
A few minutes or hours	no	Recover (1 step)	Self (sender)

With this configuration, Alice's transfer to Bob takes a configurable amount of time to settle, during which Alice could unilaterally choose to cancel the transfer, akin to an "undo send" feature which is only available for a short period of time. Due to this, Bob would likely not consider himself officially paid until the settlement window has elapsed. There is no need to trust any arbitration system or body, which makes the RWT an unbiased utility token. Recovery is simple and inexpensive.

Use cases
Generic payment token for users with a lower risk tolerance, such as retail users. This could prevent phishing, thefts, scams, or mistakes of sending to the wrong address.

One specific example is when establishing a payment connection for the first time. Banks today typically send an initial transaction of a few cents and ask the customer to confirm the amount. Using this RWT, another entity could send the full amount, and the customer will then be able to see the amount as unsettled in their balance. If there is a mistake and the customer does not receive the amount, the entity can simply retract the transaction.

The implementation of this is left as an exercise to the reader, though it can be implemented as a simpler version of the arbitrated wrapper (#1) with similar components.

06 Related Work

Architectures for recoverable assets have been explored in a number of prior works.

Eigenmann drafted a contract for a reversible token that extends the ERC20 standard. It uses an escrow method, where the escrow period is 30 days, during which the sender could recall the money at any time. After 30 days, the recipient must call the contract to claim the funds from the escrow if the sender did not reclaim the funds. This is certainly a possible flavor of recoverable tokens but does not fill the needs of most use cases today.

The elegant Bitcoin Covenants proposal uses two keys (or more) to enable a vault owner to finalize or revert transactions from the vault 24 hours after they were posted, showing how recoverability can work not just on Ethereum.

More recently, Lossless.io provides an example ecosystem of recoverable wrapper tokens with a specific set of configurations, through their wrapper standard LERC20. All LERC20 tokens have a shared governance system through Lossless, where anyone with staked LSS tokens can initiate a freeze request for any LERC20 during a fixed 24-hour window. Once a freeze is executed, the company decides whether the reversal is warranted, and if so reverses the transfer. Unwrapping requires two transactions: a withdrawal request and the withdrawal itself. A settlement period (separate from the 24-hour period) only applies to transfers to DEXes manually whitelisted by Lossless.

Finally, Firewall is an optimistic L2 built from a modified OP-stack. Within a certain time window, decentralized governance has the ability to retroactively prepend a contract exploit with a pause on the same contract. This enables the rollup to undo losses caused by hacks.

Conclusion

We presented the recoverable wrapper token (RWT) as a configurable protective mechanism allowing for recovery of stolen ERC20 tokens.

Developers can configure it to their needs through the implementation, customizing the recovery process, arbitration system, length of recoverability, transferability of unsettled tokens, and the base token. We described two example types of RWTs: an arbitrated wrapper for designated ecosystems, and a generic wrapper token that allows senders to cancel payments within a short amount of time.

Since much of the DeFi ecosystem was not made to handle recovery events, naturally the question of composability arises when RWTs are used in DeFi. We discuss how to solve this problem through "R-Pools" in our upcoming paper, a joint collaboration with Stanford researchers (Dan Boneh, Qinchen Wang).

Open questions remain: what other RWT configurations would be useful, and in what scenarios? Can we implement the RWT in a more gas-efficient manner? How does this compare to recoverability at other layers of the stack (account, consensus)? To what extent would recoverability have a deterrent effect on exploit and theft attempts? Recoverability of some form is needed for mass adoption of blockchains, so we look forward to seeing more work in the future on this domain.

Acknowledgments

We'd like to thank Qinchen Wang and Dan Boneh as two of the co-authors for the original ERC-20R paper RWT builds upon, as well as others who have contributed thoughts to this research: Li Fan, Rachel Mayer, Calvin Cai, Joao Reginatto, Adrian Soghoian, and Marcus Boorstin.

CIRCLE

circle.com/circle-research