Abstract

This proposal describes Non-Transferable Contracts with updatable ownership and solves single person contract ownership problem. NTCs are contracts which owned by a single-person with protection from sale or transfer to another person. While it's possible to implement now with solidity there is one issue which isn't solved yet it's a possibility to update owner address with the guarantee it's not a buyer, but the same person.

Motivation

There are a lot of cases when we need to make sure that owner of some asset is a single person which wouldn't be change in the Future. It's easy to solve with a regular address via checking signature, but it's not available for contracts yet. Contracts have many benefits to be used in a single-person cases. Due to address could be lost or compromised, what means total loss of control, also it restricts multiple devices to be used without sharing a single private key.

Who requires transfer protection:

- token time locks,
- reputation systems,
- access control lists.

All this systems require the only one person to be able to receive assets/points/scores etc. All of them could be enhanced with non-transferable contract which allows to:

1. Use multiple keys to manage assets.
2. Restore access in the case of missing master key via different mechanics.

Specification

To make this work we should make sure that contract couldn't be transferred to third party. This could be achieved with preventing contract from switching owner, what makes the idea useless in the most cases. There should be possibility to update master key and to make sure that the same person own new key. The solution here is the knowledge of the fact that the current private key and the new one is known to one person in one point in time, and before the contract ownership changed.

To achieve this we should sign current private key with the new one. And then disclose old private key to the contract. This makes impossible the sale of the contract to third party, because a buyer should receive a private key of a seller and thus could own the contract without paying, this will stop a seller from making a deal.

Transfer algorithm

1. Sign old private key body (not the hash of the key) with the new key.
2. Call startUpdate()

to start master key update and pass new master key address and a signature as params. After this call any operations with the contract are restricted. This action couldn't be rolled back or stopped.

1. Call finishUpdate()

and send the old private key to the contract.

This algorithm could be implemented for multiple master keys too, it would require to update all master keys simultaneously.

Contract

contract NonTransferableContract { // Method locks the contract and stores call params. function startUpdate(address newAddress, bytes32 r, bytes32 s, uint8 v); // This method compares signatures, adds new master key and unlock the contract function finishUpdate(bytes32 prevPrivateKey); }

Requirements

Precompiled contract (optional)

NTC requires new precompiled contract to calculate address from a private key using secp256k1 algorithm. It's needed to verify that private key in finishUpdate()

is the one that that the previous owner address matches with. While it could be implemented in solidity, it seems rational to use it in more effective way.

Opcode (optional)

New opcode to burn address and transfer remained funds to another address to restrict future key usage.

Limitations

1. Master key must be used only for NTC contract management.
2. Master key must be used only on a single network.
3. Before calling finishUpdate()

there should be a confirmation delay to prevent a situation when private key is revealed, but the ownership update is rolled back due to a fork.

1. All funds from master key should be

Rationale

Revealing or burning private key is decided to be a solution here for the cases when the master key is compromised and should be removed from the contract. The ability to remove master key make it switchable, and thus allows to sell contract's ownership. Cross signing of private keys protects from this.