Refundable Token: A no intermediaries time-ensurance eletronic refund system

Author: *Labelle Moon (hugo.card@usp.br)*

1. Introduction

Everytime a transaction is sent in a cryptocurrency, it has to be approved, validated and mined. The time taken to realize this operation is very variable, as observed in many blockchains. The lack of precision in time to a transaction suceeds characterize a flaw on the model. The reason that this is a flaw it's that real world needs precision.

As an example, consider the following case:

Rose needs to pay the rent of her's house. The rent must be paid every month on the 24th day, and in case of lack of payment, fines will be applied. Rose, by using crypto, has two alternatives to pay on the right day.

The first one is Rose actively stay on the computer and perform the transaction on the right day, and the second is delegate it to someone else. The first alternative is impracticable, since Rose is not always available. The second needs trust.

And both the first and the second method have the flaw presented earlier: in case a big demand arise, the Rose transaction will be delayed and can overpass the 24th day limit. That proves the fragility of the actual system.

The key element for solving this problem and many others, such as tokens sent to a wrong address, addresses that had their private keys stolen, without having an intermediary is to have reversible transactions by using smart contract token.

The term "time-ensurance", defined by "something happen in the exact time it's supposed to" is obsolete in cryptocurrency. However, by using the following system, the flaw is fixed, since transactions just became unrevertible precisely after a specific block number is reached.

Transactions should be reversible for a period of time. The 'time' is defined by the actual block number. A minimal refund block number should be assigned to every address and every transaction must have at least the value of it. The minimal refund block must be able to be changed. A debt check should occur, ensuring security for refundable amounts. All of this must be stored in a central storage array.

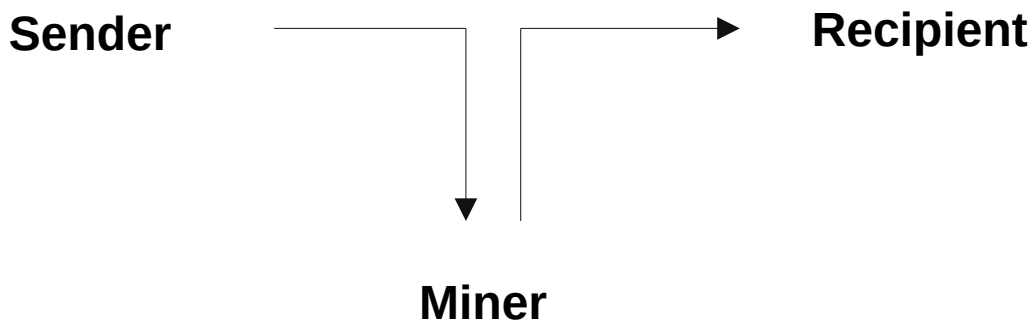
This is Refundable Token, a no intermediaries time-ensured eletronic refund system.

2. Time measurement

Simply do all the transactions in the token be reversible at every moment would be a mistake, since it would entirely kill the purpose of the token and no transactions would happen.

The ace in the hole is use time as an condition for a transaction being still refundable or not. However, a major flaw in this model it's that there is no exact timestamp in the blockchain. Miners can even cheat on the actual value altering it to their wishes.

But there's one number that miners cannot cheat: the block number, and that number will be the one that can produce reversible transactions for a period of time.



Miners can just assign the block number as being the last one plus one.

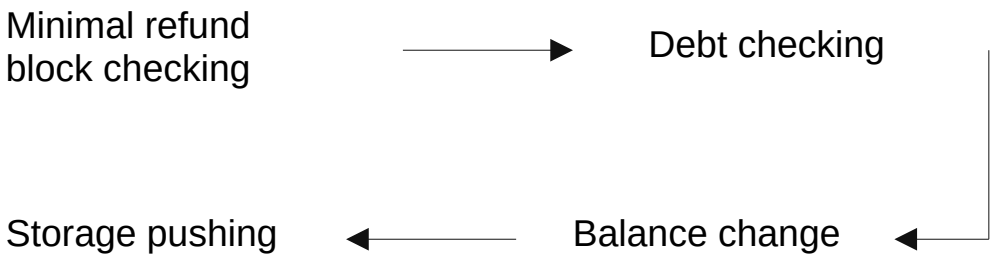
3. Transactions

Everytime a transaction is sent, four variables have to be assigned to the transfering method. These are the recipient (the address which will receive the funds), the amount, the block limit and the debt indices. The block limit is the number of blocks in which the transaction will continue to be refundable. In case this value is set as zero, the transaction is unrefundable*. The debt indices is part of the debt process checking, and will be explained later.

When a transaction happens, four diferent processes will be done: the minimal refund block checking, the debt checking, the balance change, and finally the storage pushing.

** Not necessarily true. Check 4.*

The transaction process



4. The minimal refund block checking

Another special aspect of the token is a special process called minimal refund block checking. As mentioned before, everytime a transaction is sent a block limit variable has to be specified, and if this variable is set to zero, the transaction is unrefundable. However this is a special case, where the minimal refund block of the address is set to zero as well. When the minimal refund block of an address is different than zero, the block limit is at least the

value of it.

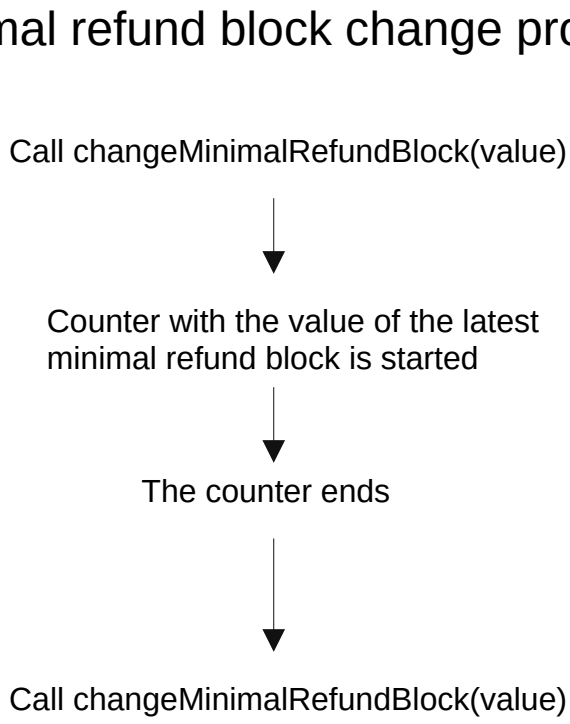
By default, when an address receives a transaction for the first time, a minimal refund block is set to it as a value of 300 blocks.

Summary,

- If the **block limit < minimal refund block, block limit = minimal refund block**.
- If minimal refund block = 0, block limit can be any value, even zero.
- By default, **minimal refund block = 300.**

The default block limit is not the most suitable for every address. For this reason, there's a special process that can be called at every moment by an address that changes the minimal refund block. This process is separated in two different stages and is performed by the method *changeMinimalRefundBlock().*

4.1. The minimal refund block change

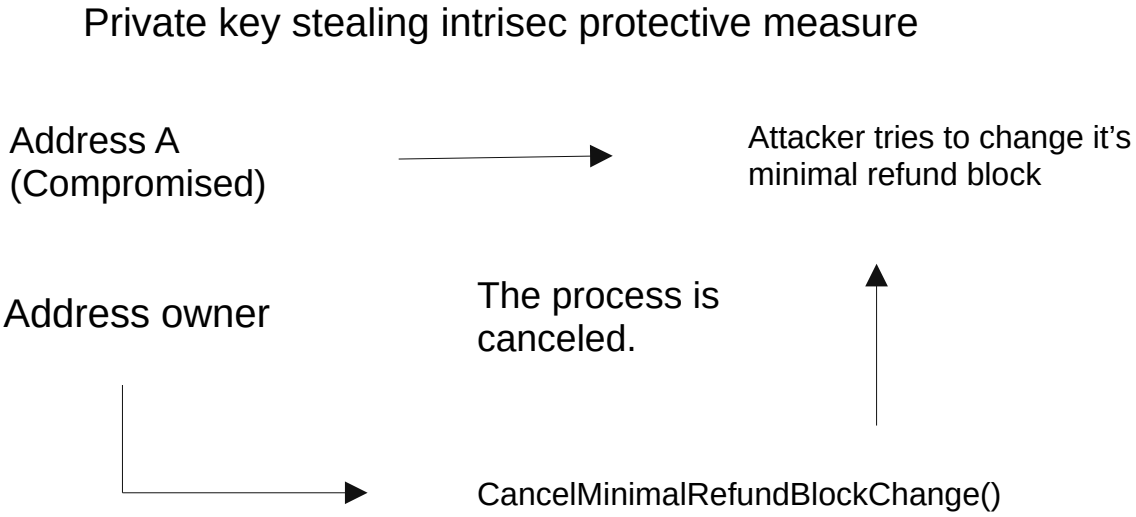


The minimal refund block change process is an extensive process, called by *changeMinimalRefundBlock()*, that has just one variable input that is the value in which the minimal refund block should be changed. However, this process is not as simple as just calling it. When called by the first time, the method will start a counter of blocks starting by the block number when it was called until it reaches the block number when called plus the latest minimal refund block.

When this counter ends, the method can be called again and now the minimal refund block of the address will be changed to the value variable *when the method was called by the first time.* Changing the value to zero enables unrefundable transactions.

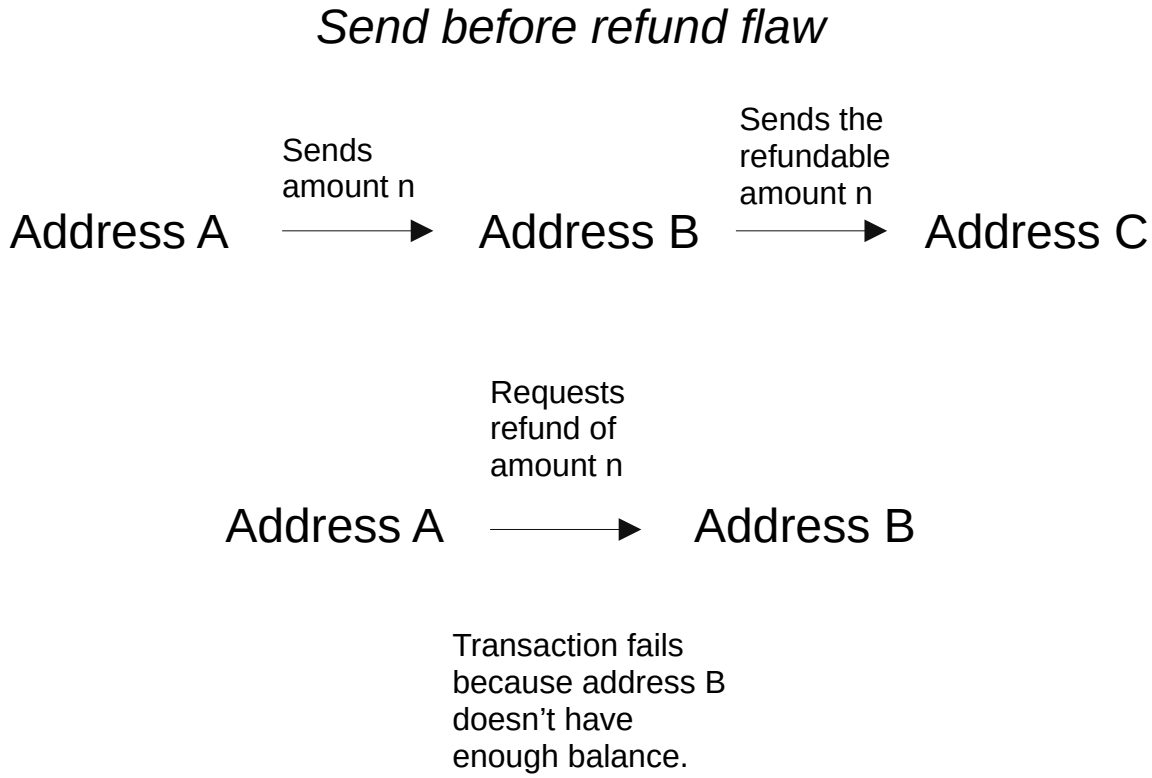
The process to change the minimal refund block can be cancelled at every moment after started by calling *cancelMinimalRefundBlockChange().* The reason this method exists is to stop unwanted minimal refund block change process. This method should be called just

one time.



5. Debt checking

An easily verifiable flaw in the token model is called *send before refund* where the recipient of a refundable transaction sends the tokens to another address while the transaction is still refundable. This makes possible token stealing and kills the purpose of refundable transactions.



However, this flaw is totally mitigated by a process called debt checking, which entirely solves the issue by a very little cost. Fundamentally, this process checks the sender of a transaction for debts and calculate the free balance. Debts are defined as amounts that can still be refundable by another addresses of an address. The free balance is the total balance amount of an address minus the debts.

A special variable that has to be assigned to the transfering method is the debt indices

variable, which is an array that has to be set by the address that made the transaction which contains the indices of the refunds that aren't valid anymore, so the total debt is discounted by the value of the expired debts. The reason this variable is not automatically set is to save gas.

In case the free balance is less than the transaction amount and the debt indices transaction variable is not specified, the transaction reverts, preventing the *send before refund* flaw. In case that the discount is not enough, the transaction also reverts.

6. The storage management

Refunds should be stored in the blockchain, so it's possible to contabilize debts and actually refund amounts. For understanding how these are stored, it's necessary to explain what a refund is. In a simplistic way, a refund is an object containing an issuer, the address that made the transaction and has the possibility to refund it, an amount and a block end variable, which is the block number when the transaction is unrefundable.

```
# This is a refund.

struct Refund {
        address issuer;
        integer amount;
        integer blockEnd;
}
```

Refunds are stored at an array that is inside a central storage variable called transactions_rf, which acts as an associate array, and are stored in the recipient of a transaction key. That means that a recipient can have multiple refunds in it's transactions_rf array.

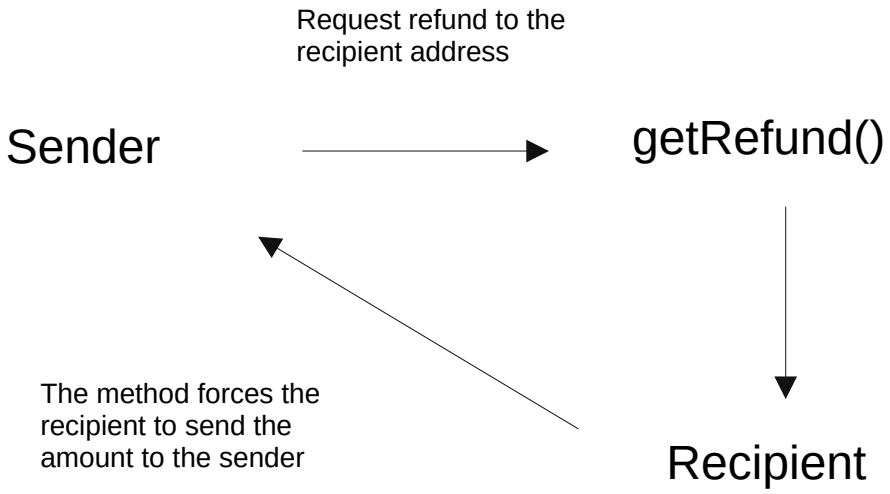
```
# Represents the refunds of a specific address.
# Note: refund_obj are Refund objects.

address_recipient: [refund_obj_1, refund_obj_2, refund_obj_3]
```

Transactions that contain the block limit variable set as zero will not push a refund object to the recipient transaction_rf array.

7. The refund method

The method used to obtain refunds from a transaction is *getRefund(recipient, id)*, where recipient is the address at which the refund should be obtained and the id is the identifier of the transaction in the recipient transactions_rf array.

The address that should call this method is the address that made the refundable transaction. After the method is successfuly executed, the amount is sent back to the address that requested the refund.



8. The auxiliary methods

The auxiliary methods are special methods used to get information about the token variables. They doesn't cost any gas cost to execute, and can be called everytime by every address. Actually, these method are eight and have special functions, and can be seen below:

Function seeRefund(address target, uint256 id):

Method used to return specific refund variables. The return will be an array as [x, y, z] where x is the refund issuer address at integer format, y is the amount and z is the block when transaction is unrefundable. The target address should be the recipient of the specific refund.

Function seeRefundSize(address target):

Method used to return the length of refunds array of a specific target address. The target address should be the recipient of the specific refunds.

Function seeMinimalRefundBlockRunning(address target):

Method used for checking a change to the minimal refund block by checking the _isMinimalRefundBlockChangeRunning. Returns true if a change is running, and false if not.

Function seeMinimalRefundBlock(address target):

Method used for returning the minimal refund block of a specific address. Returns zero if the address can create unrefundable transactions and the minimal refund block is zero.

Function seeLastMinimalRefundBlockChange(address target):

Method used for checking the last minimal refund block change, and returns a specific block number. Also, can be used to estimate the minimal refund block change block.

Function seeDesiredMinimalRefundBlock(address target):

Method used to check the number that the minimal refund block will be changed if a change is taking place. Returns an integer with the value.

Function seeAddrDebtAmount(address target):

Method used to check the actual determined debt amount of an address. Note that the amount does consider expired debts that haven't be removed as actual debts.

Function fetchRefunds(address target):

Method used to fetch all the still valid refunds. The return type is an array as [x, y, z, n, ...] where x, y, z, n, are the indices that are refundable in transactions_rf array.