**A Solidity-based Smart Contract**

**Solution for Life Assurance**

David Wade & Richard Kelly

December 2018

Contents

[**SECTION 1** 3](#_Toc533849753)

[**Background** 3](#_Toc533849754)

[**Rationale** 3](#_Toc533849755)

[**Business Requirements** 4](#_Toc533849756)

[**SECTION 2** 5](#_Toc533849757)

[**Design and Requirements Analysis** 5](#_Toc533849758)

[**Final Design** 6](#_Toc533849759)

[**SECTION 3** 8](#_Toc533849760)

[**Smart Contract Interaction Pattern** 8](#_Toc533849761)

[**SECTION 4** 15](#_Toc533849762)

[**Evaluation** 15](#_Toc533849763)

# **SECTION 1**

## **Background**

The aim of this project was to harness the functionality of Ethereum-based smart contracts and ERC20 cryptocurrencies to aid in the execution, tracking and settlement of transactions within an Insurance contract. These transactions are based on a savings contract between an individual and a Life Assurer.

In this scenario, the client is seeking to invest in a diverse basket of assets, each offering a different amount of investment growth, with the aim of accumulating funds for his/her child’s College education. Once a sufficient amount has been saved, the client will have the option to withdraw all or part of their funds upon request.

It is assumed, for benefit of our exercise, that the client is happy to both fund and accept withdrawals into their personal wallet on the Ethereum Network. It is further assumed that all transactions occur in either ETH (Ethereum) or the tokens issued by the Insurer itself, albeit some of the activity therein will refer to ongoing ‘real-world’ transactions, as highlighted below.

## **Rationale**

The rationale for seeking to utilise the functionality of the Ethereum blockchain for an Insurance Savings contract is for the obvious benefits that could be derived in terms of automation and therefore cost savings, both for the Insurer and the client. More specifically, for an Insurance savings contract these could be:

* Cost and time savings in the administration of the contract, from client onboarding to payments processing and reconciliation
* Heightened visibility both for the client as everything is stored on the blockchain and is irreversible. There are also benefits for the insurer in terms of audit traceability
* Disintermediation between the client and insurer and pay
* Payments can be collected and settled directly with the clients Ethereum wallet, without the need for traditional banks and the associated fees
* The ‘if and else’ features within the solidity smart contracts inherently lend themselves to Insurance contracts, which follow the same cause and effect cycle
* Enhanced fraud protection as the rules of insurance contract are hard-coded into the smart contract itself, meaning the terms of the contract can only be executed when the conditions have been fulfilled. Were fraud to take place this could easily be traced back to the individual transaction, time and date, via the blockchain.
* Improving the quality of data available to underwriting and actuarial divisions leading to an improvement in risk assessment and pricing

Of course, with any new technology there will always be a bridge to gap in terms of the clients understanding of a new technology and their willingness to adopt it. However, the benefits above, if realisable, would put an onus on the Insurer to aid their clients in making this transition.

## **Business Requirements**

A client has established a Regular Savings contract with a Life Assurer. It will save a fixed amount on an ongoing basis. The savings will be split equally between three funds, each applying a different rate of growth/uplift depending on market returns. The savings come from a separate personal account, which is unrelated, receiving no yield and has no influence on the contract.

The client can save a fixed amount of per month using this Regular Savings Contract with the hope of reaching a given amount of savings that will pay for College Fee’s at some point in the future. The monthly contributions are split equally into 3 different asset types, each managed by the Insurer:

**Fund Account:** Equity Fund; Property Fund; Bonds Fund.

**Annualised Growth Rate:** A company employee will monitor the ‘real world’ performance of the three assets once a month and apply an adjustment to the savings in each fund account to reflect this.

At some point in the future the client will decide to crystallise his/her assets. The assets are transferred back to the Client Savings whereby the process ceases.

The idea here being to lock-in/secure some of the value of the savings.

# **SECTION 2**

## **Design and Requirements Analysis**

Based on the business scenario outlined above, we identified 3 main requirements or processes which would have to be supported:

1. The receipt of incoming funds from the client and the allocation of these funds to the appropriate fund accounts
2. The ongoing monitoring of fund performance and subsequent adjustments to the amounts held in each fund, based on this activity
3. The final settlement to the client once a request is made to withdraw the balance saved

As we analysed each of these requirements in turn, the following design elements

**Wallets/Addresses:**

Each client will require a dedicated address/wallet for each of their funds – in this case three addresses for each of the Equity, Property and Bonds funds. This is required to ensure all client funds are kept distinct from each other.

An additional unique address will be allocated to the client based on the deployment of their ‘Fund Splitter’ smart contract, which has been described in more detail below.

**Wallets/Addresses:**

Upon review it was agreed that three smart contracts types are required:

1. Fund Splitter Contract:
   * **Purpose:** Identify when the Client Savings have received monies.   
     Take these monies and divide the amount received in 3 equal amounts.   
     Transfer to each Wallet (Equity, Property, and Bonds).
2. ICO Contract:
   * **Purpose:** Convert ETH to new tokens – called ‘CSV Tokens’. These can be used to buy back ETH at a later date when a withdrawal is requested.  
     Looks at how much ETH it receives and sends 100 tokens back to the fund account for every 1 ETH received
3. Token Profit/Loss Contract:
   * **Purpose:** To add or remove tokens based on profit/loss of the three asset types and evaluate the growth/loss applicable to each asset type

The process then repeats itself each time the client adds to his Client Savings.

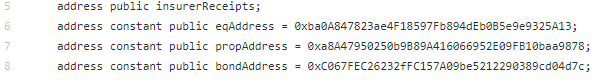
When a withdrawal request is received an entity called ‘Employee’ manually instructs the contract to subtract the equivalent number of CSV Tokens. These are used to buy ETH which is sent back to the Client Savings.

## **Final Design**

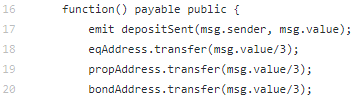
**Fund Splitter Contract:**

The fund splitting contract was designed with a simple payable function to process the incoming payments to the destination fund accounts.

Initially, the contract identifies the forwarding addresses of the client fund accounts. In this way, the contract could be re-used for additional clients with the only change required being an update to the addresses specified at the outset:



The payable function then specifies a transfer to each of the above identified addresses, splitting the value of the incoming payment (‘msg.value’) into three, given the number of addresses involved. This could be adjusted, were more or less addresses involved.:



**Token Issuance / Token P&L Contracts:**

The contracts for both the token issuance and P&L contracts were based on the sample code provided below:

1. CSV Token ICO Contract Base Code:

<https://github.com/bitfwdcommunity/ICO-tutorial/blob/master/ico-contract.sol>

1. CSV Token P&L Contract Base Code:

<https://github.com/bitfwdcommunity/Issue-your-own-ERC20-token/blob/master/contracts/erc20_tutorial.sol>

Both code bases were then repurposed for the needs of the CSV token, with some unnecessary functionality removed (e.g. ‘transferAnyERC20Token’ and ‘approveAndCall’ functions).

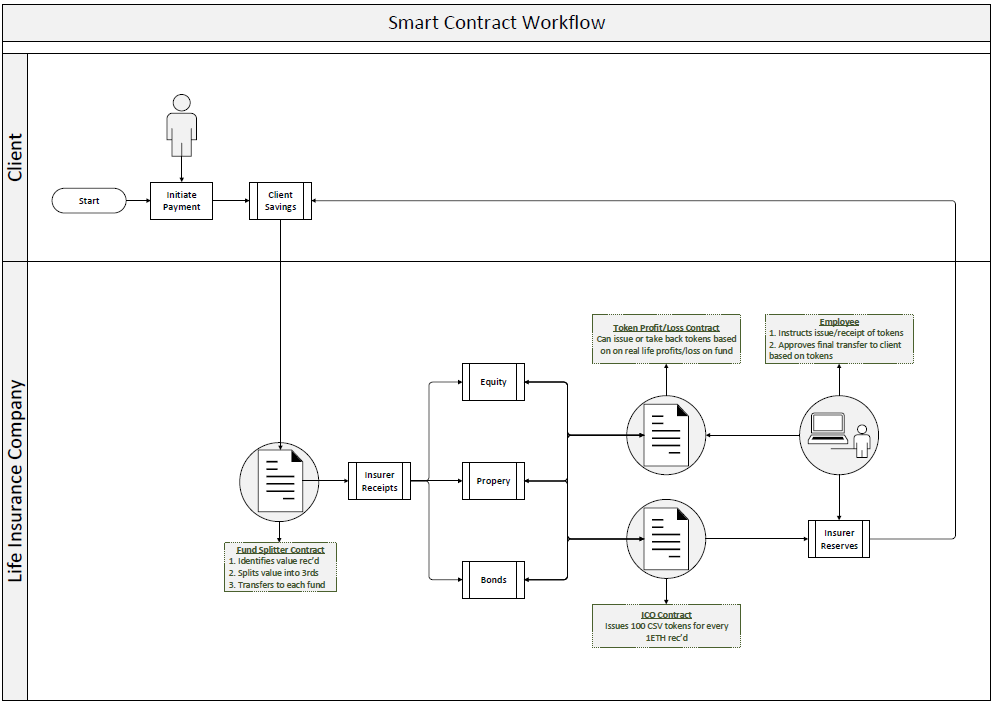
The token issuance contract gave us the ability to automatically assign tokens to any address from which it receives ETH. It has an unlimited supply which makes it scalable. The same contract address could also be used for all client fund accounts making it easy to maintain, with client balances available through the ‘balanceOf’ function at any time.

The Token P&L contract, while similar to the Token Issuance contract, is designed so that it cannot receive ETH (protection against error) and is simply used to issue or withdraw tokens based on real life activity. Again, balances can be checked either through the contract functionality or via a quick check on Etherscan.

# **SECTION 3**

## **Smart Contract Interaction Pattern**

Below we have provided a workflow diagram and live walkthrough of the smart contracts as implemented on the Ropsten Test Network, using the Remix compiler and utilising test addresses setup in advance on the MetaMask application.

**Workflow:**

*Fig 1. Lifecyle of client savings*

**Live Walkthrough:**

1. A client initiates a transfer of 3 ETH from their personal wallet (‘Client Savings’) to a ‘Fund Splitter’ Contract, provided to them at inception. This smart contract automatically splits the incoming funds into three equal parts and transfers these amounts onwards to three fund accounts, pre-agreed with the client. Each client would need to be setup with their individual ‘Fund Splitter’ contract and Client Fund accounts at inception:

**Addresses**

**Client Savings:** 0x89be8d51c9325264f0793970D8bd5885eDc3f037

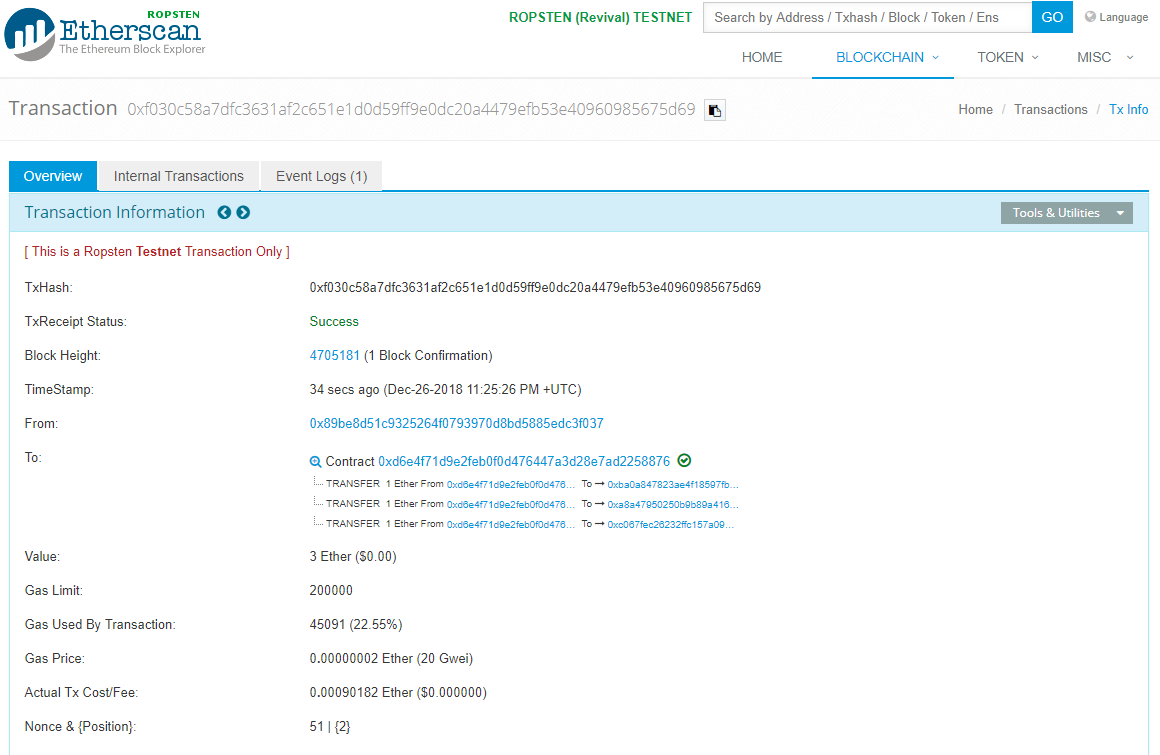
**Fund Splitter Contract:** 0xD6e4F71d9E2FEB0f0D476447a3D28e7ad2258876

**Insurer Receipts:**  0x20ef5163841644962ED3e2424d659E21Cb96Fd54

**Equity Account:** 0xba0A847823ae4F18597Fb894dEb0B5e9e9325A13

**Property Account:** 0xa8A47950250b9B89A416066952E09FB10baa9878

**Bonds Account:** 0xC067FEC26232fFC157A09be5212290389cd04d7c



*Fig 2. Fund Splitter contract sending ETH received*

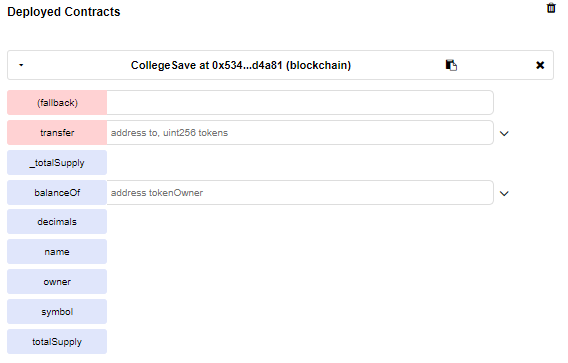
*to three client fund accounts*

1. An employee of the insurer is responsible for manually transferring any ETH received to the client fund account to the ‘Token Issuance’ contract. The contract will automatically issue 100 ‘**College Save**’ tokens or ‘**CSV**’ in return for any ETH received. This operates like an Initial Coin Offering (ICO) and has an unlimited supply. The contract can issue CSV to any account it receives ETH and it can track the individual token balance of any address on the network using the ‘balanceOf’ function:

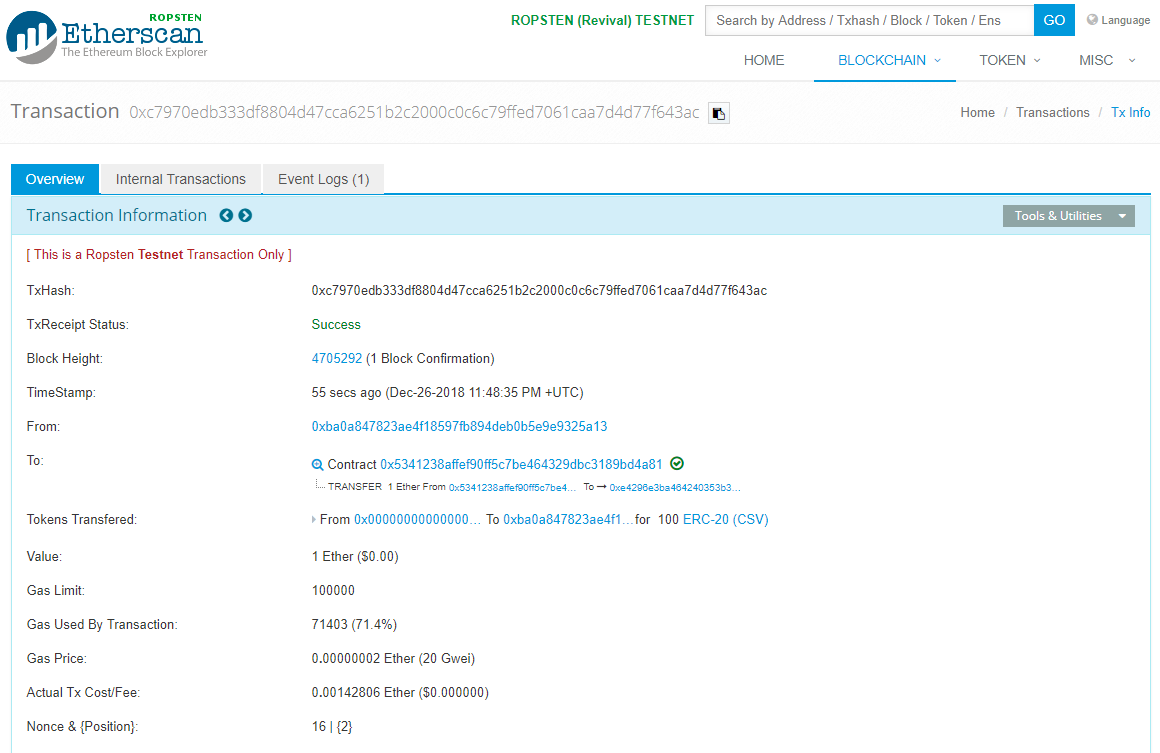
**Addresses**

**CSV Token ICO Contract:** 0x5341238affef90ff5c7be464329dbc3189bd4a81

**Insurer Reserves:** 0xe4296e3BA464240353b39ff520C5584Ed5925b9D



*Fig 3. College Save ICO contract deployed in Remix*



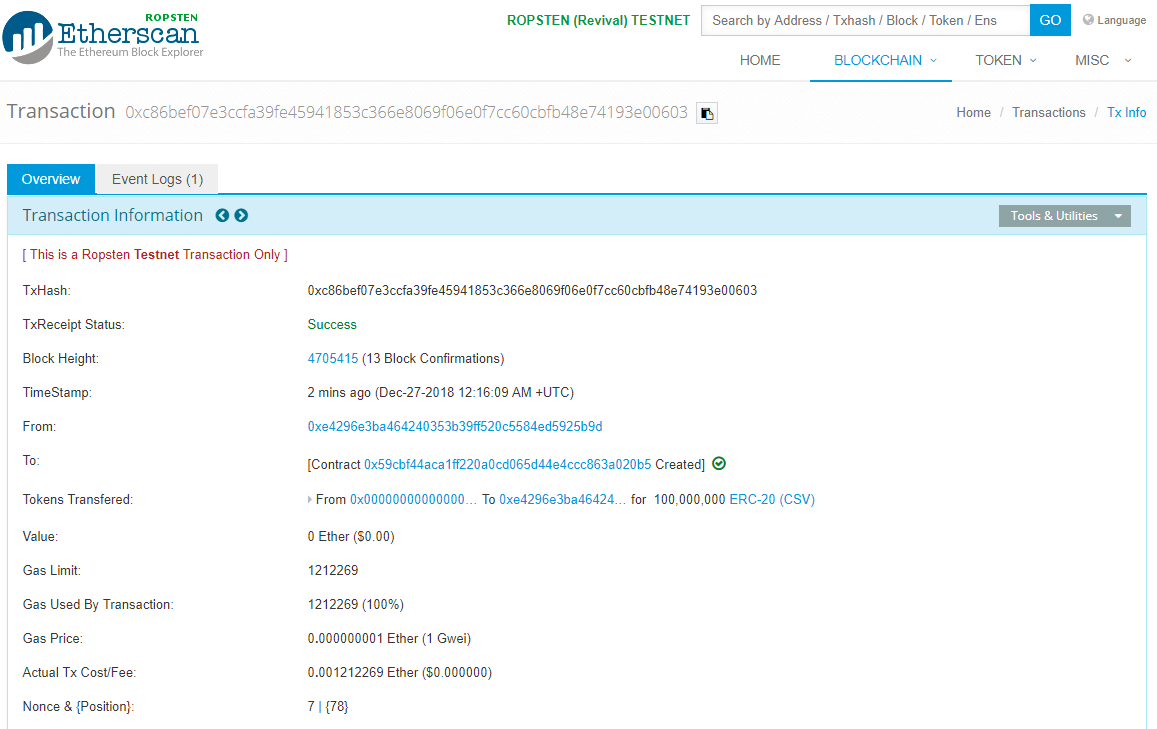
*Fig 4. College Save Contract issuing 100 tokens*

*in return for 1 ETH received from Equity Fund Account*

1. An employee tracks the ‘real life’ performance of each fund in the market. Each month, the employee sends each fund further CSV or calls for funds to be returned depending on whether the fund made a profit or loss in each month. This contract is much like the ICO contract, except this contract does not accept ETH and has been granted a limited supply:

**Addresses**

**CSV Token P&L Contract:** 0x59cbf44aca1ff220a0cd065d44e4ccc863a020b5

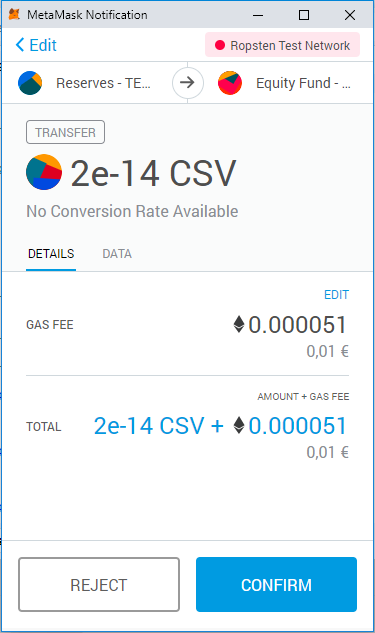


*Fig 5. CSV Token P&L Contract deployed to*

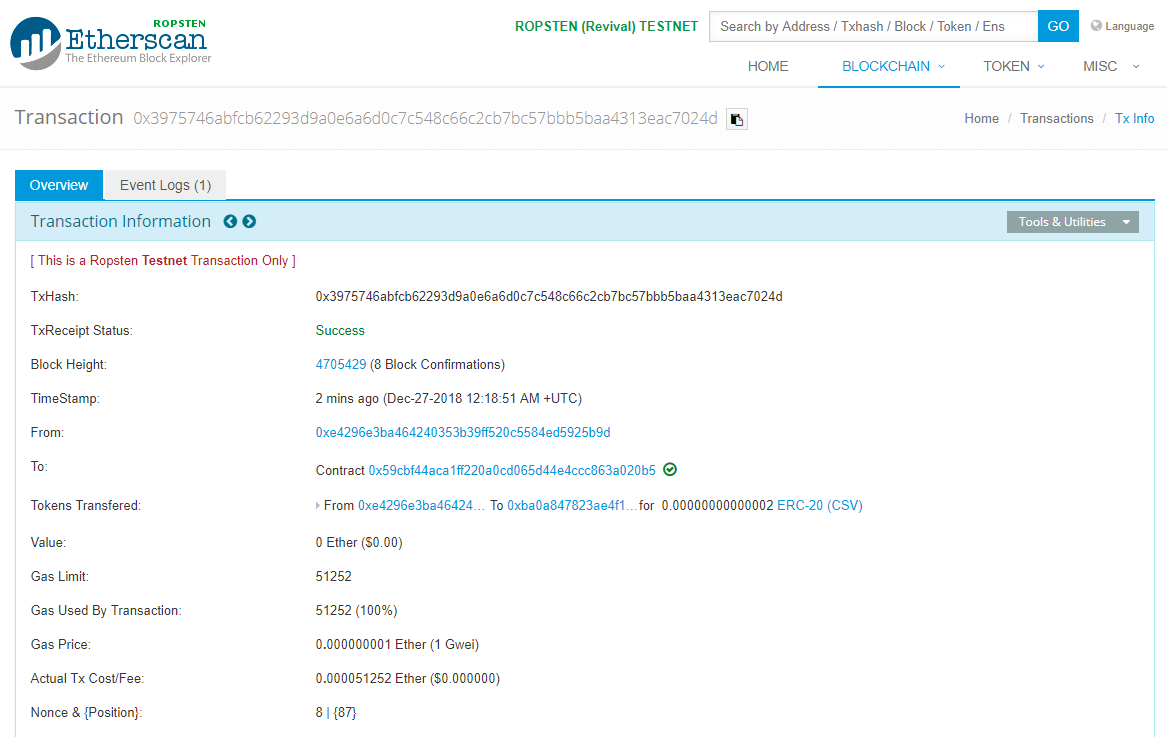
*Insurer Reserves Account*



*Fig 6. Transfer of 20,000 initiated via the contract in Remix*



*Fig 7. Confirmation request for token transfer from MetaMask*



*Fig 8. Etherscan confirmation of transfer of 20,000 CSV tokens to the*

*Equity Fund account via the Token P&L Contract*



*Fig 9. Approving the transfer of 10,000 CSV from the Equity Fund*

*back to the Insurer Reserve account via the contract on Remix*



*Fig 10. Confirming the allowance approved by the token owner (in this case the*

*Insurer Reserve Account) which can now be transferred from the Equity Account*

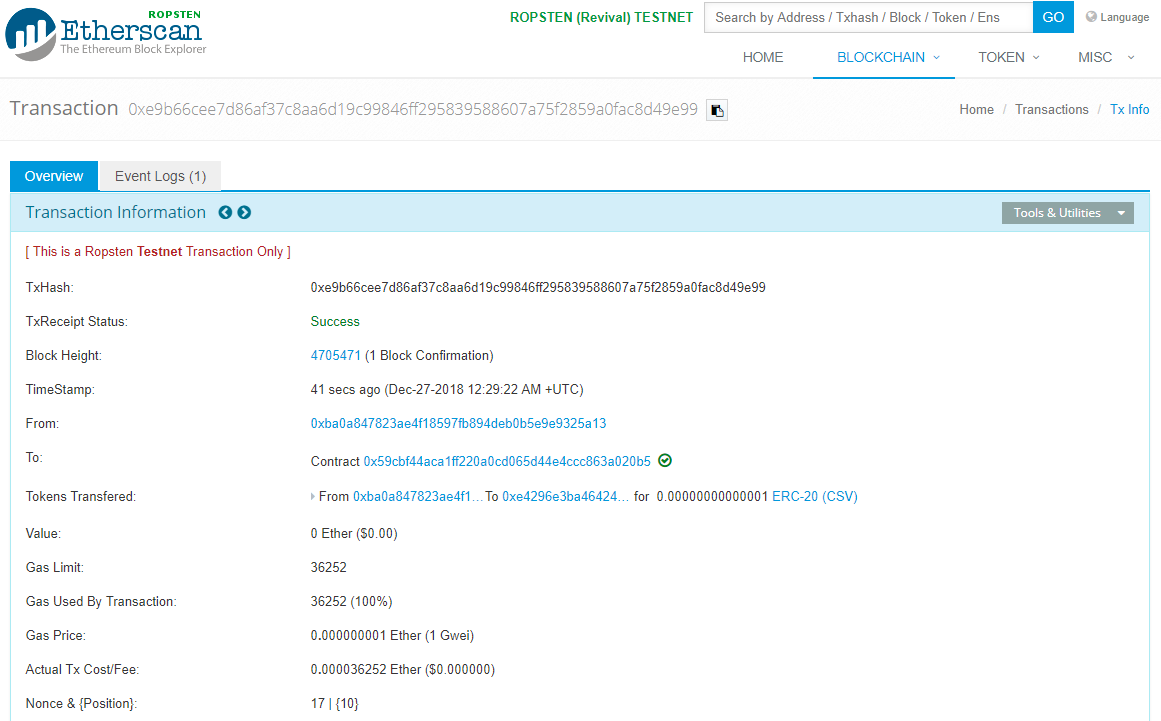


*Fig 11. Initiating return of 10,000 CSV to the*

*Token Owner via the contract in Remix*



*Fig 12. Confirmation request for token transfer from MetaMask*



*Fig 13. Etherscan confirmation of transfer*



*Fig 14. Checking balance of CSV Tokens in Equity*

*Account via smart contract in Remix*

# **SECTION 4**

## **Evaluation**

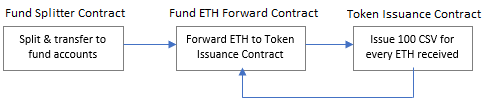
Reflecting on this project, there were a number of clear advantages and challenges to using a solidity smart contract and token solution to support the business scenario in question:

Advantages:

1. **Speed & Cost:** The execution of fund splitter contract was complete in a matter of seconds. While there is some effort initially in setting up the relevant addresses and contract, once in place this functionality could provide real time and cost savings to a company, who traditionally would have to monitor a bank account for incoming client funds and manually allocate & reconcile thereafter. Even outside of the contract-driven transactions, there are further time and cost savings on the initial transfer from the client and ultimate settlement at the end of the contract. Again, these transactions are executed in an extremely short timescale and for a minimal gas fee. Some consideration does need to be given to the fact this was the Ropsten test network and transactions on the live network may take longer and require more gas. However, these savings in time and cost would remain.
2. **Transparency:** The system, as implemented, provides a high degree of visibility, both to the client and the Insurer. On the client side, view access for the three fund accounts could be provided to the client so they could monitor each transaction and balance in real time. This removes the dependency on Insurer reporting for the client and enhances trust in the process. This would likely give the client more control as they could take immediate action on an underperforming account rather than retrospectively, once a monthly client report was received, for example. For the Insurer, the token issuance allows the them to track client funds on a grand scale and through the deployment of a single contract on the network. While all client funds are deposited into a single ‘Insurer Reserves’ account, the ongoing daily activity on each client fund account would be individually tracked via the contract, removing any reconciliation concerns.

Challenges

1. **Incoming Client Fund Transfer:** Our initial intention had been to deploy additional smart contracts on each of the client fund accounts, with the requirement to transfer onwards any ETH received to the ‘Token Issuance’ contract, thereby making the entire cash receipt/token issuance process one seamless transaction:



Unfortunately, when we implemented the contracts on the client fund accounts, the transactions from the initial ‘Insurer Receipts’ account failed to execute due to an ‘Out of Gas’ error. Following some research, it became clear that using the *address.transfer* or *address.send* functions in solidity was problematic as these functions have a [2300 gas limit](https://solidity.readthedocs.io/en/v0.4.24/security-considerations.html#sending-and-receiving-ether) on what they will forward for subsequent transactions. This means there is insufficient gas for the fund account contracts to execute their tasks. We attempted to use the *address.call.value* function, as this comes without the same limits (and gas used can be dictated) but the contract failed to run when used.

Of course, there is a strong possibility that a workaround would present itself with deeper knowledge of solidity. However, the gas limits on the payable functions in general seem like a real limitation, particularly when attempting to execute multi-dependant smart contracts.

1. **Ongoing Fund Performance Monitoring:** We had considered the use of an Oracle to track offline fund performance and trigger smart contract-driven adjustment to the funds. However, we understand that each ‘ping’ to an external database like this comes with a gas cost. This could lead to substantial costs were the oracle triggered with a high frequency such as hourly/daily. There is perhaps more potential, were this used on a monthly basis to limit gas costs. However, at this level of infrequency the effort of manually executing the same transaction would likely be the same or less. Again, through further research there may be options available which do not demand the same gas commitment. If not, this would feel like a challenge worth surmounting for the Ethereum Network as all contracts would refer in some way to ‘real life’ activity – so to make this connection prohibitively expensive would surely limit the potential of the functionality.
2. **Use of ETH:** In our example we assumed the client had transferred Ethereum to the Insurer and accepted settlement in same. This would be problematic in today’s market conditions where volatility is at an all time high. Were the client to attempt to convert the ETH back to a fiat currency, there is a high risk they will have suffered a loss. Once way to surmount this would be for the insurer to accept fiat into an offline bank account and issue tokens on the basis of this. That would however introduce additional manual steps and would reduce some of the benefits mentioned above. Alternatively, asking the client to utilise a ‘stable coin’ such as TrueUSD (TUSD), Circle Coin (USDC) may offset some of the volatility risk, albeit these coins can sometimes come with issues of trust regarding the levels of fiat held in reserve. A final solution may be time itself, that is waiting until Ethereum is widely adopted (not guaranteed) and the volatility mirrors that of the more traditional fiats.