Provident One: decentralized perfect information insurance vehicle on an Ethereum blockchain

https://provident.one

Jorge Izquerdo
jorge@unpatent.co

Luis Cuende
luis@unpatent.co

Palo Alto, CA

November 2016

Version 0.1

Abstract

This whitepaper describes an insurance vehicle that can handle insurance subscribers, examination of claims and investments on top of the insurance float in a decentralized manner using an Ethereum-compatible blockchain ledger.

By challenging the information asymmetry in tradicional insurance models, an insurance fund can be turned into an investment vehicle in which investors will profit from the problem the insurance protects by investing on it of getting better over time, even if also by the existance of the insurance itself.

Disclaimer: This document is not (nor it attempts to be) an academic paper. We would be naïve to think this is a final version of the protocol, with the improvement rate of Ethereum and Solidity being so fast. Feedback and suggestions are always more than welcome. This is an ongoing and community driven effort.

Disclaimer 2: This whitepaper will not describe the legal forms or structures that may or may not be required to legally operate in one or more countries and we do not make responsible of any possible violation of the law that any independent party may commit by using the ideas or computer software described here. Please, check with your (hopefully open minded) counsel before doing anything you may regret.

0. Motivation

The story of this whitepaper starts with our own company Unpatent, which is on a mission to make IP more efficient in order to empower innovators. The US patent system (circa 1790) is outdated and overwhelmed with the current rate of progress, and IP extorsionists, popularily known as patent trolls, are taking advantage of this broken system to extract $29b from the economy every year.

Unpatent provides a crowdfunding platform to combat patent trolls [1] by democratizing patent reexaminations. Threattened or interested companies can create a campaign against a bad patent, find other interested parties to contribute, and evidence is crowdsourced to demonstrate the non-novelty or obviousness of the patent. After that, we create a reexamination petition with the new evidence before the USPTO.

When going to market with this product, we had a hard time finding companies to contribute to fight on a patent by patent basis, even if they were highly concerned of the existing risk. We decided to build an insurance product where companies just payed a little bit every month and could rest assured they wouldn't have to worry if they had patent problems.

The problem creating insurance products, is that you need huge sums of money just in case you experience a Black Swan (more on 4) or need to buy reinsurance from another company. In either case, because a company is moving very big amounts of money, it is subject to hard regulation and bureocracy.

When sketching out the first version of the protocol, we realised that by having the insurance on the blockchain and by it being very transparent, it could have a set of very interesting properties that would make it much more efficient than traditional insurance.

1. Introduction

The insurance industry's size is close to incommensurable — in 2012, the life and health and property and casualty insurance sectors reported $7.3 trillion in total assets [2].

Yet being such an important industry, the insurance model has not evolved very much during the last decades. In fact, the current insurance model is one in which all parties are incentivized to lie to each other, since it's (almost) a zero-sum game, where what an insured company gets in form claims, is money taken away from the insurance company balance sheet. Therefore, you see companies lying by making claims and insurance companies not paying the insured entities

what they deserve.

From the entrepreneurial point of view, traditional insurance regulations usually suppose barriers for newcomers, such as having a minimum amount in coverage, licenses and time consuming bureaucracy.

From the customers point of view, traditional insurance usually is opaque, with extremely complex contracts and drawn-out claim processes.

And finally, from the investors point of view, traditional insurance doesn't provide the necessary transparency in order to make investments on the funds performance.

1.1. Decentralization

Public decentralized ledgers provide:

- Transparency: Everything that is happening on the ledger can be reviewed by everyone.

- Immutability: Everything that happened on the ledger can be reviewed by anyone anytime, and the underlying truth is too expensive to change.

The aim of decentralizing insurance is to provide the lowest possible entry barriers so innovation can happen, and to provide extreme transparency to all the processes so both the customers and investors can benefit from it.

There are multiple decentralized ledgers that could be used to build an insurance fund on top of.

We will be looking at public ledgers, since them being public is a requisite to solve The Byzantine Generals Problem [3].

The public ledgers that a solid insurance vehicle could be built upon are Bitcoin [4] and Ethereum [5].

Ethereum is a distributed computing platform, featuring smart contracts and with its own currency, ether.

The smart contracts are executed on the Ethereum Virtual Machine (EVM), a decentralized virtual machine.

Ethereum has both the contract capabilities and the network effects to be a great choice.

Another alternative, Rootstock [6], an Ethereum-compatible blockchain that uses 2-way peg to Bitcoin and rewards Bitcoin miners via merge-mining, could be an option in the future when it surpasses the testing stage.

2. Participants

The insurance vehicle consists of:

- **Holder**: The entity that bootstraps the vehicle, parametrizes it for optimum performance and sets the conditions under which a claim can be made.
- **Subscribers**: Entities that buy protection to receive a service free of charge if a set of verifiable set by the holder conditions are met.
- **Examiners**: Those who verify the claims reported by the subscribers and its validity in order to claim the required service.
- **Investors**: Individuals or companies who bet on the funds' performance and receive dividends when it profits.

- **Service providers**: Entities that can provide a service when a claim is approved. (In some kinds of insurance this might not be needed and the claims may be payed directly to the damaged subscriber)

With the current implementation of the protocol, these different participants must be an Ethereum account, wallet or contract that is able to perform transactions on the network by spending gas.

In order to allow more traditional existing entities to use this new kind of insurance, the holder may provide a way for subscribers or investors to participate in the vehicle without directly operating from an Ethereum wallet, developing a proxy service that will allow them to just have a normal service subscription and pay with fiat.

2.1. Holder

This is not the typical definition of holder — there is not a company or individual that can arbitrarily modify the behavior of the insurance vehicle.

The holder's **duties** are:

- Bootstraping the insurance vehicle:
- Set insurance plan prices and conditions to join the insurance based on an actuarial model.
- Set conditions for claiming.
- Pick claim examiners.
- Pick service providers (if needed).
- Triggering period closing in the fund in the stablished timeframes (every quarter or similar). This is the moment when profits are split in dividends and new tokens can be minted (See 4, real world implementation).
-

- Upgrade the software of the *services* of the system, when a new update of the protocol is released.

The holder's **rights** are:

- To receive a portion of the minted tokens for operating and maintaining the vehicle.
- To freely admit or deny subscribers the right to join the vehicle depending if they meet its criteria.
- Possibility to be a claim examiner.
- Possibility to be a service provider.

The holder **cannot** (prohibited by the Smart Contract):

- Arbitrarily transfer or withdraw the money of the fund.
- Arbitrarily mint new tokens to decrease the value of the currently issued one.
- Transfer recently minted tokens until the next period.

The holder may be a company, individual, any organization, institution or in general anyone that can comply with the points listed above.

2.2. Subscribers

The subscribers' **duties** are:

- To pay their subscriptions in a temporary manner. Failing to meet this criteria, will make the subscriber unable to file any claims.
- To provide all the necessary information to the examiners for them to analyze whether the subscriber's situation complies with the vehicle's claiming conditions.

The subscribers' **rights** are:

- Get a service or payout when an events that meets the conditions meet the insurance policy.

2.3. Examiners

The examiners will be a group of people, elected by the holder arbitrarily.

The examiners' **duties** are:

- To review the claims made by subscribers and reach consensus with the rest of examiners in a temporary manner in order to determine a final decision on the claim complying with the vehicle's claim conditions or not.

The examiners' **rights** are:

- To be rewarded for their service with tokens of the fund.

2.4. Investors

An insurance fund model is such that if the premiums payed by the subscribers are greater than the amount spent on claims, profits will be split equally depending on token stake.

Since everything is transparently handled, the investors will be able to accurately know the state of the vehicle and the funs, helping them in their investment process.

The investors' **rights** are:

- To withdraw their proportional part of the float's dividends on a timely manner.
- Audit every transaction and claim in the insurance.

2.5. Service providers

In case the insurance needs it, service providers will be the ones in charge of providing the service in the case of a claim, pe: in a car insurance, the mechanic you will take your car to if you have a car accident.

The service provider **duties** are:

- To provide the mentioned service at the price set with the holder.

The service provider **rights** are:

- Receive a payment for the service once it has been provided.

3. Implementation

This implementation architecture attepts to follow best practices in smart contract design as of Solidity v0.4.4, this will probably be obsolete soon. The current contract implementation has been made open source under the GNU aGPL v3 license and it is published on Github:

github.com/providentone/contracts

Given the non-triviality of the system and the multiple interactions between different parties, the system must be composed using multiple smart contracts orquested together.

The main architecture is composed of different kinds of contracts with distict purposes: a *manager* (public interface, funds holder and dependency injection),

services (business logic) and *persistance* (storage for the system).

Aside from these main components, every time a claim is submitted a new *Claim* contract is deployed, with all its functionality being self-contained in the contract.

Thanks to this separation of functionality and purpose, it allows for different component upgrades and easily testable components.

3.1. Manager

The manager is the entry point of the insurance fund, all the funds are kept in this contract and handles dependency injection to make components accessible from other parts of the system.

It provides the infraestructure to update the *services* of the system and in the case of *persistance*, it also allows the migration of the data when a new *persistance* is deployed.

The manager contains as little logic as possible, just the minimum to ensure money flow of the contract doesn't go wrong.

It exposes the following interface to the outside world to interactuate with the insurance fund:

Javascript

```
pragma solidity ^0.4.4;

contract Provident {
// Insurer related
function getNumberOfInsurancePlans() constant public
  returns (uint16);
function getInsurancePlanPrice(uint16 plan) constant public
  returns (uint256);
function getInsuredProfile(address insured) constant
  returns (int16 plan, uint256 startDate, uint256 finalDate);
function buyInsurancePlan(uint16 plan) payable public;
function createClaim(uint16 claimType, string evidence,
  address beneficiary) returns (int);

// Investor related
function getTokenAddress() constant returns (address); //
  ERC20 Compliant Token
function getCurrentTokenOffer() constant returns (uint256
  price, uint256 availableTokens);
function buyTokens(address tokenHolder);
function withdrawDividends();
}
```

Because of the public nature of data in the blockchain, a permission system is implemented to ensure that only the right component can execute some function. A permission system that can be used with a modifier has been implemented for this purpose.

Javascript

```
contract InsuranceService {
  function superImportant()
  requiresPermission(PermissionLevel.Manager)
}
```

3.2. Services

Services are stateless contracts that encapsulate all business logic of the system.

3.2.1. Insurance service

This service handles all the logic for the selling of insurance plans, checking insured profiles, the mechanism for claim submission and the assignment of

examiners to claims.

3.2.2 Investment service

The investment service implements the logic needed for issueing an Ethereum based token: checking balances, transfers and allowances of tokens (Implements ERC20).

On top of that, the service takes care of spliting the profits of the fund and calculating what amount corresponds to each token holder as dividend, as follows:

```haskell
dividend[holder] = balance[holder] * totalDividend / totalTokenSupply
```

3.3. Persistance

Store data relevant to the system. Persistance implements a migration function to transfer its information to another instance when it needs to be updated.

3.3.1. Insurance persistance

Data store for the *insurance service* with the following data structures:

```javascript
```

```
mapping (uint16 => uint256) public planPrices;
uint16 public planTypes;

struct InsuredProfile {
uint16 plan;
uint256 startDate;
uint256 finalDate;
uint256 totalSubscribedClaims;
}

mapping (address => InsuredProfile) public insuredProfile;
mapping (address => mapping (uint256 => address)) public
subscribedClaims;

mapping (uint16 => address) public examiners;
uint16 private examinerIndex;

mapping (uint256 => address) public claims;
uint256 public claimIndex;
```

3.3.2 Investment persistance

Data store for the *investment service* with the following data structures:

Javascript

```
  uint256 public totalSupply;
  uint256 public tokenPrice;

  mapping (address => uint256) public dividends;
  mapping (address => uint256) public balances;
  mapping (address => mapping (address => uint256)) public
allowed;

  mapping (uint256 => address) public tokenHolders;
  uint256 public holderIndex;
  mapping (address => bool) private isHolder;
```

3.3.3 Accouting persistance

Because of the importance of having a trail of all the money in and out or the fund, the accounting persistance stores a ledger of all the transactions of the fund and keeps track of the accounting for every period. The data structures are:

Plain Text

```
enum TransactionDirection {
Incoming,
Outgoing
}

struct Transaction {
TransactionDirection direction;
uint256 amount;
address from;
address to;
string concept;
uint256 timestamp;
}

mapping (uint256 => Transaction) public transactions;
uint256 public lastTransaction;

struct AccountingPeriod {
uint256 premiums;
uint256 claims;
uint256 dividends;
uint256 pastLosses;
bool closed;
}

mapping (uint256 => AccountingPeriod) public
accountingPeriods;
uint256 public currentPeriod;
```

3.4 Claims

Claim management is the most complex part of the Insurance implementation, every single claim features:

- A state machine with an ACL for different state transitions and function calling.
- A voting mechanism by which *examiners* study the submitted evidence and vote on whether the claim should be accepted or rejected.
- A withdrawal mechanism by which one or multiple service providers or benificiaries can receive funds once the claim has been accepted.

Given its the complexity and inherent risk, it was decided that whenever a subscriber created a new claim, the fund would deploy a *Claim contract* to the blockchain and transfer its ownership to the claim submitter.

This decision has a set of advantages:

- A claim is not upgradable. A subscriber is certain that at the moment she submits a claim, its inherent logic won't change for that particular claim, meaning that even if the insurance logic (terms of service) changes, that claim will follow the rules of the moment it was created.
- Claim specific activity occurs outside the main insurance contract, such as voting, state handling and submission of new evidence.
- Claim funds withdrawal happens inside the claim itself and not from the main contract, having a clear history of how funds moved out of the claim.

4. Real world implementation

The problem with insurance, and the reason why it is an industry with such high barriers of entry is for the big amounts of capital needed to back up a fund in case

of Black Swan[7] scenarios.
In order to bootstrap insurance vehicles in the blockchain, an ICO event (Crowdsale) is needed to provide the fund with the liquidity it needs in order to operate without the risk of going bankrupt.

The optimal amount of tokens in circulation needs to be calculated in accordance with the predictions of the actuarial model for that given period of time, trying to meet a targetted interest for investors.

For simplification purposes lets asume there is only one insurance plan and all companies have the same risk of making a claim.

Given a targetted annualized interest rate of the fund is *r%* and it gives dividends *q* times a year (every quarter), *y* number of companies in the fund, *z* dollars per plan per period, *p* as the probability of an entity having an accepted claim every year, *c* as the cost of that claim and *t* the price of the token. We can calculate the optimal number of tokens in circulation *T* as follows:

$$T = y * \frac{z - p * c}{t * r}$$

When the amount of premiums increases, the fund will mint the number of tokens needed for this proporcion to be maintained (also because a bigger fund is needed, since the risk is greater by onboarding more entities in the fund.)

As tokens will be traded in secondary markets, every time there is a token minting event, the fund will sell the newly minted tokens at a value just below the market rate. As minting only occurs after the fund has given away dividends, it can be assumed that the price drop will match a rise because of the optimism in the fund performance. Though that is the expected scenario, until it is live we won't know what happens. The new part is that investors can check the fund's performance in real time at any time, it will be interesting to see how that affects the value of the token in the market.

In every token minting event, the holder will be granted x% (10 to 20 percent seems to be the standard these days) amount of the tokens for maintaining the vehicle in good shape. The holder won't be allowed to transfer her tokens until the next minting event, to try to avoid token dumping.

5. Improvements

These are some things we are researching that would make Provident One much better:

- Building a decentralized app (using Ethereum as its only backend) for insurance subscribers and inverstors to have a UI to interact with the Insurance Fund.
- Using prediction marketplaces instead of actuarial models to asses the risk of a new entity joining the insurance, letting the investors decide premium amount.
- Using Machine Learning for claim examination. By building a model that learns from examiners decisions given certain claim evidence, before submitting a claim, it would be possible to know the probability of the claim being accepted. It would be possible then to automatically reject or approve cases below or above a certain threshold.
- Creating Insurance index funds that can aggregate stake in various insurance funds to diversify investment.

6. Timeline

- Nov 2016: Early whitepaper and contract code is published.
- Dec 2016: Community efforts and decentralized app.

7. Conclusion

The aim of this whitepaper is to provide an overall architecture and a set of Ethereum contracts that enable anyone to freely provide, consume and invest in transparent and fair insurance vehicles.

The insurance industry is stagnant: the regulation makes it difficult to innovate, the incentives are aligned in an inappropiate way and processes are extremely opaque. The solutions to these problems can be achieved by using a decentralized ledger and a better incentives model.

---

1. Patent trolls — Wikipedia https://en.wikipedia.org/wiki/Patent_troll
2. Annual Report On The Insurance Industry — Federal Insurance Office, U.S. Department of the Treasury https://www.treasury.gov/initiatives/fio/reports-and-notices/Documents/FIO%20Annual%20Report%202013.pdf

3. The Byzantine Generals Problem — Leslie Lamport, Robert Shostak and Marshall Pease http://research.microsoft.com/en-us/um/people/lamport/pubs/byz.pdf
4. Bitcoin: A Peer-to-Peer Electronic Cash System — Satoshi Nakamoto https://bitcoin.org/bitcoin.pdf
5. Ethereum: A secure decentralised generalised transaction ledger — Dr. Gavin Wood http://gavwood.com/paper.pdf
6. RootStock White Paper: Bitcoin-powered Smart Contracts — Sergio Demian Lerner https://uploads.strikinglycdn.com/files/90847694-70f0-4668-ba7f-dd0c6b0b00a1/RootstockWhitePaperv9-Overview.pdf
7. TALEB, N. N. (2007). The black swan: the impact of the highly improbable. New York, Random House.

Technical feedback

Join the community discussion in Slack

Discussion about the paper in Github Issues