Interlude: a decentralized, permissionless scavenger hunt game

September 8, 2021

Abstract

We describe a practical implementation of a decentralized, permissionless scavenger hunt game using the blockchain (Ethereum or any other smart contract platform). Developers can hide keys (the private part of a private key pair) anywhere in their games or on the web and register the public part on the blockchain. The goal for the players is to go and find these keys. The protocol defines a cryptocurrency that is mined each time a player finds a key and used to reward both players and developers. The game is decentralized because it is made independently by potentially thousands of developers with no central coordination - one could also say it is "decentrally designed". The game is permissionless in the sense that not only anyone can play, but also anyone can take part in designing it. In that sense, it can also be seen as one game with two classes of participants: the Key Hunters (players) and the Key Hiders (developers).

1 Background

The development process of video games has changed dramatically since the heroic times of Doom and Unreal. The advent of advanced game engine technologies has made it possible for reduced teams, and often for individual developers, to create 3D games and experiences that would have necessitated huge teams in former days. The flip side of this coin is that the public's expectations have increased in tandem with the quality of games. As a result, even though individual and non-professional developers are today able to create very interesting content, they have no way to market it to the public and monetize it, since they are competing for player's time and attention with commercial games with huge teams and resources.

We see this as an untapped potential: thousands of developers may be very creative and talented, but all this creativity and effort is spread between thousands of unrelated small games that individually have very little chance to attract players attention. They need to unite!

2 Decentralized game design and game protocols

To solve this problem and allow for the exploitation of this potential we are introducing the concept of “decentrally designed game”: games that are based on a number of rules/standard, that would be implemented by independent uncoordinated developers in their individual apps, and would provide them shared meaning and gameplay. The easiest way to explain this concept would be to draw an analogy with the web: just like the web protocols allow individual sites to make sense and together form a coherent mass of information, these “game protocols” would allow all these small games to share information to provide a coherent gaming experience to the player. For example like the web protocols define an $< img >$ balise to tell the browser to display an image, the game protocol could define an $< enemy >$ balise that would warn the “game browser” of a potential target, and enable them to provide the relevant actions to take. In the same analogy links to other websites could be analogous to portals towards other game, etc...

We propose a very simple example of game protocol. If a game protocol is defined as a codified way for games (nodes) to share information and modify a shared state, our example would take the smallest possible amount of information for each node: a single bit. Each node would just be able to switch a boolean in the player’s game state, e.g. to declare that a condition has been fulfilled. This condition is arbitrarily defined by the game node’s developer: it could be reaching a certain place in his game, beating a level, killing a boss, just opening the game, etc. By indexing each of these bits/conditions on the blockchain, and adding a random sequential selection mechanism (when fulfilling the condition the mechanism randomly computes a new id of condition to fulfill) we get our game: a decentralized scavenger hunt, that would let millions of players compete to find keys across potentially thousands of these uncoordinated “game nodes”.

We now describe a practical implementation of this game, using the blockchain (ethereum) to store the shared state.

3 Implementation, algorithm description

We use a smart contract called *KeyHunt* to store our state and implement the selection mechanism. Since we use public cryptographic key pairs for the implementation, we will call *key* the boolean/condition of the previous section. Likewise we will use the expression “find the key” instead of “fulfilling the condition”.

3.1 Developer side

Developers generate a key pair (or ethereum address and private key). They register (for a fee) the public part on the Key Hunt smart contract (with a small message as a hint to find the key, for example an IP address or a URL), and “hide” the private part anywhere they want in cyberspace (most often on a website or inside their game, but it could be a real world place, for example a QR code somewhere, although this would be very impractical). In practice, “hiding the key” means placing a script that is triggered when the condition decided by the developer is fulfilled. This script signs a message with the private key, that will prove that the player “found the key” (aka fulfilled the condition). In practice, for convenience and security reasons the private key will most often be stored on a private server and “finding the key” will simply call a signing API point on this server to sign the required message.

3.2 Player side

When they want to start a game, players call the “StartHunt” function of the *KeyHunt* smart contract. This function registers the player’s participation and generates a *ticket* (nonce) as randomly as Ethereum allows it.

The goal of the game is to compute an array of bytes32 called $Proofs$. Each entry of this array is the proof that a corresponding key was found. The proofs are defined recursively in the following way:

$$\begin{cases} Proof[0] = Hash(playerAddress, ticket) \\ \forall i > 0, Proof[i] = Sign(Hash(playerAddress, Proof[i-1]), PrivateKey[k_i]) \\ \forall i >= 0, k_{i+1} = Proof[i]\%NbKey \end{cases} \tag{1}$$

where $Hash$ is a hashing function (in practice the Ethereum Keccak256 function), $Sign$ is likewise a public key cryptography signing function, k_i is the index of the i-th key to find (starting at index 1) and $NbKey$ is the total number of keys.

In other words, when the player ”finds a key”, it actually gets a signature of the previous proof by the private part of the key that

- proves he found this key
- allows him to compute the id of the next key

This defines a recursive series of proofs, where the proof for key k_i contains the proof for key k_{i-1}, and where to find the proof $Proof[i]$ the player has to find the corresponding key. For playability reasons, we limit the time of a single session: the goal is to find as many keys as possible, and to return the series of proofs in the imparted time (in practice, two or three hours).

3.3 Reward function

The reward for a session is the sum of the rewards for each key found. The reward for a key is proportional to:

- a multiplier: exponential function is the number of the key in the key serie, with a basis to determine (certainly 2)

- the difficulty of the key: a ratio of the number of times the key was found and the number of times it was supposed to be found.
- a coefficient that is determined so that the number of distributed tokens is roughly equal each month.

The reward for each key is shared between the player and the developer, with a ratio to determine. (most certainly equally shared)

4 Cryptocurrency, tokenomics

The protocol defines a cryptocurrency that is minted each time a key is validated by the contract. The tokens are shared between the player and the developer that created the key. A fixed number of tokens is distributed each month, so there is no fixed amount of token for each key. Rather, the algorithm defines an intermediate quantity, called points, that are awarded when a player finds a key, and a proportional constant tokenPerPoints that is updated each week to ensure the fixed coin creation rate - similar to the way the bitcoin protocol changes the mining difficulty to ensure a constant amount of bitcoins is created.

The cryptocurrency thus defined is very different from the vast majority of tokens defined on the Ethereum network, because in our case the token creation rules are integrated in the game to incentivize participation. We are able to do this by creating what is essentially a cryptographic puzzle similar to the Proof of Work puzzle in eponymous blockchains: if we assume no collusion between the key creators, the sequence of proofs that the keys were found constitutes a "Proof of Game" that:

- is impossible to fake (enforced scarcity)
- and is formally verifiable (ie can be checked with computer code)

The protocol is thus able to reward this "Proof of Game" directly, in the same way Bitcoin can reward the Proof of Work solution.

This allows the Interlude network to benefit from a positive feedback loop that is arguably even stronger than that of blockchains: an increase in token price makes the game more rewarding to players and developers, thus attracting more developers which makes the game better for players. More players makes the game more profitable for developers (per the second key hunt mechanisms, where dev's income depends on player numbers, and also through increased traffic and potential publicity), and overall increases the "public awareness market share" of the network, which in turn will increase the token value.

Once again this is very different from most ERC20 tokens. These tokens are usually used as a means of exchange to trade resources that are not easily priceable on chain: there is no simple function that determines exactly how much each participant in the project deserves and the protocol usually relies on an external market system. As a result, in these cases the token price has no influence on the network: even if it increases a lot, the network won't become more useful to its participants. This is how we were able to see token price skyrocket during the ICO boom in 2018, while the number of users would stay extremely small: the token price had no impact on the network's usefulness.

Hence the "crypto-economic profile" of Interlude's native cryptocurrency is more similar to that of Bitcoin and Ethereum than to most ERC20 Ethereum tokens.

5 Potential problems and their solutions

5.1 Decompiling the game to get the keys

This is the most obvious potential problem: the attacker could get the key by analyzing the game or website source code. This can be easily thwarted by not storing the key on the client side, and instead relying on a server that would be called in an ad hoc way. Of course, it is also possible to reverse engineer the procedure for calling the server, but here we should remember the hunt is limited in time (about three hours). A key is supposed to be found in about ten minutes, which is too short an amount of time to decompile and analyze the game's sources (or its memory layout during gameplay), especially if they are sufficiently obfuscated.

5.2 Loss of secrecy/publication of solution

Most scavenger hunts can only be played once; once their secrets are discovered the game is a lot less fun. This is especially true in the age of the internet, where you only need one player to find a secret for it to be known by all of humanity in a matter of minutes. This problem can only partially be solved (once someone finds a key there's no way to prevent him from transmitting the info), but we think it won't be too much of an hindrance in practice for the following reasons:

- First, not all keys are "secret". Many keys could be seen more like achievements, for example "Beat a given level in a game", or "kill this boss".
- Second, from a purely economical perspective, it is not rational to publish any information on a key you found. This is a global competition and a zero sum game: by publishing information on a key, you're not only allowing other players to win more without earning anything yourself, but you're actually decreasing your potential gain (since the number of tokens is limited).
- Third and, in our opinion, decisively, because of the sheer scale of the game, stemming from its "crowd-designed" aspect. Traditional games are made once and for all by very big teams, and are played in the same way by millions of players. In this case a secret will be found very quickly, and once it's found it's impossible to change the game to account for that. This is not the case with a crowd-designed game: we are talking of thousands, maybe tens of thousands of small games made by tens of

thousands of independent developers. These games are very small, made over the course of a few week-ends by hobbyists; if a key is found and published, the developer can move it somewhere else, rebuild the game and reupload it in a matter of minutes. Worse, the developer could then put wrong information on any solution website, and even design a trap to have players believe the information is right and waste a lot of time... Let's remember the key must be found in a limited time; it seems like a big risk to go look on the internet for info that could be outdated or even intentionally misleading.

5.3 Sybil attack

The main type of attack on the system would revolve around getting the reward without really playing the game, e.g. without finding the keys. One could create a lot of keys, try to find them himself by controlling a lot of Ethereum addresses, and initiate the hunt at the right moment. Because of the sequential nature of the key selection mechanisms alluded to above, we don't think this is practical from a crypto economic standpoint (the attacker would lose a lot of money trying to compute the right address).

5.4 Key quality

Since the game is decentralized and permissionless, we have no control on key quality. In particular, several problems may arise:

- Poor key quality: developers could just register a key and put it plainly on a website. This would not be attacking the system strictly speaking, but it would defeat its purpose - which is to be a game, e.g. to be fun.
- Lost key: as happens with many games or internet based resources, the developer could just forget about the key, or the site or game where it is to be found could go offline or not be playable anymore, etc. This is one of the biggest problems and could have the most destructive effect in the player's experience: imagine you are on an incredible winning streak with potential for huge gain and you're stopped abruptly because the website where your next key is supposed to be is offline... Infuriating!
- Key spam: since a key registered on the smart contract represents a permanent income stream (because keys are chosen at random by the algorithm, so each key earns the number of tokens minted each month divided by the number of keys), in a naive setting a rational decision would be to create as many keys as possible, until the lifetime revenue of a key is equal to the gas price for registering them. This would, of course, completely spoil the experience for players and discourage honest developers from creating valuable content.

The solution to these problems is clear: we need to incentivize the developers properly, to make them care about the quality of their keys - basically the income for each key must be proportional to the “quality of the key”, in a way to be defined. This will be done in three steps:

1. Making the registration of a new key expensive. It should be seen as an “investment” from the developer; think several hundreds of dollars in price (the price would be calculated in native tokens).

2. Second, we design a scoring system that takes into account a notation from player (to evaluate the gameplay quality of each keys, which is very subjective) and other elements: for example, if a key is the last one (eg it wasn’t found by the player) several times in a row, its score would be decreased drastically to account for the possibility that it can’t be found at all anymore.

3. Third, the selection process for an key takes into account the score of the key: basically, the hash that is used to determine the next key id will also provide a “quality draw” (for example, taking the n most significant bit of the hash) that should be lower than the key score. If it’s not, the hash is hashed again until a proper key is found. This does not change the deterministic character of the key selection procedure. Then, the probability that a key is drawn, and therefore the revenues it yields to its owner/developer, will be directly proportional to the quality of the key - and the key being “unfindable” would translate into an immediate revenue hit. Combined with the high entry price for creating a key, this will ensure developers are incentivized to provide a good experience.

6 Protocol improvement

We can bring several ameliorations that will make the overall game more fun and add replayability:

6.1 Key tags and categories

We will allow developers to add a tag to the key (by adding a bitflag field to the key struct). The developer will be free to determine the tag of his key. Likewise the startHunt function will be supercharged to allow for a tag argument, which will restrict the type of key to be selected. For example keys could be tagged in function of their estimated finding time, or there could be a tag for keys hidden in a certain genre of game (eg horror or action). The meaning of each tag would be determined by convention, possibly after the launch of the game.

6.2 Competition mode

We will implement a slight modification of the KeyHunt algorithm by allowing players to create a “common hunt”, where everyone shares the same ticket and therefore competes in a kind of race to find the highest number key. In this case, there would be no token reward (it would be too easy to cheat), so the players that want to participate would have to stake a certain amount of tokens that will be shared between the winners.

This mode provides a different, and somewhat more spectacular gameplay experience. We expect it to become the main use of the smart contract, once the native rewards per player decreases too much.

6.3 Additional monetization mechanisms

As stated above the protocol will include a rating mechanism that will award a score to each game according to its overall quality and creativity. The probability of a key being selected will then be proportional to this score. A very natural way to increase value creation by the protocol (and thus demand for its token) would be to allow developers to increase the probability their key is selected, in exchange for a fee (either staking or burning the network token). This would enable value to flow not only from the players to the developers but also from some developers (for example free-to-play mobile app developers that would use Interlude to drive traffic and installs) to others (more creative, indie game developers that would contribute higher quality content to the network and would be rewarded for it).