Exploring a Privacy-focused Decentralised Pet Quest App on the Secret Network

Ryan Scofield
Computer Science and Software Engineering
University of Canterbury, Christchurch, New Zealand
rsc104@uclive.ac.nz

Abstract

Traditional online games rely on centralised servers, where user assets are controlled by a single authority. This leads to reliance on trust, and a lack of privacy and true ownership of assets. We aim to address this issues by implementing a decentralised application (dApp) on the Secret Network that enables users to own, upgrade, and battle their pet NFTs while earning loot FTs. Secret Network addresses the issues of traditional online games, as the rules of the smart contracts cannot be broken, meaning there is trustless interaction. Secret Network also encrypts user data, and users can only see private data if they supply the correct permit(s). By using two external contracts, which implement the SNIP-20 and SNIP-721 protocols respectively, we can also guarantee true ownership and authority of assets for users.

1 Problem Statement and Motivation

Nowadays, it is common for online games to have a centralised servers which store and manipulate all user assets. The fundamental issue with centralised online games is that users do not own their own digital assets, and they must trust the centralised server to act in good faith. Our decentralised pet quest app aims to address the lack of data privacy, trustless transactions, and true digital asset ownership for users within currently existing online games.

When user data is stored in a centralised database, we have to trust that data is not misused, and is secure against external threats. A dApp on the Secret Network addresses this, as data is encrypted on-chain, and queries for this data can be locked behind permits. This is especially important for our pet battling feature, where users wager their loot tokens in the hopes that their pet is stronger than the opposing pet. If all pet data was public, there simply would be no reason to battle pets as there would be a known outcome.

Our dApp works alongside non-fungible pet tokens, and fungible loot tokens. These could hypothetically have real-world value, and could be exchanged between users. In a centralised application, this would require complete trust in a middleman, the centralised server, to exchange goods. The same thing applies to when we want to upgrade a pet using our fungible loot tokens, or claim a quest for our awarded fungible loot tokens. In a centralised application, we would again have to trust the centralised server to exchange our tokens for an upgrade to our pet, or award us our loot tokens. A dApp using smart contracts ensures that rules are followed, and users do not need to blindly trust a middleman.

Users do not have true user-ownership over their digital assets in centralised online games. Users can devote hours of their time, and even real world currency into their digital assets, but at the end of the day, the owners of the centralised servers have complete control of ownership. Our solution will ensure that users have true ownership over their in-game pets and loot tokens. This offers two key benefits: users retain control over their digital assets without relying on a centralised company, and this will support future built marketplaces for pet and loot exchanging beyond our current implementation.

2 Related Work and Content

One of the first NFT pet games was CryptoKitties [**?**]. CryptKitties introduced the idea of minting NFT pets, and breeding them to create random and unique genes for different looks and styles.

Axie Infinity is a dApp built on Etherium smart contracts, which allows users to buy, sell, and trade their pet "axies" [1]. Axies each have genetic code, which means each of them look slightly different. Axie Infinity allows for battling, fighting bosses, plotting land, farming resources, and a lot more. Being built on Etherium, Axie Infinity uses ERC-721 for axies, and ERC-20 for their small love potion currency (SLP).

Another example, similar to Axie Infinity, is Aavegotchi [2]. Aavegotchis gain experience to level up, which they gain by participating in DAO votes, and playing mini-games. Aavegotchis also have intrinsic value, as they are staked by collateral, in the form of aDAI (aave tokens).

There are many more examples of this kind of dApp using Etherium and other blockchain platforms, but none of these applications have a particular interest in data privacy.

Secret Heroes is an example of a dApp game on the Secret Network [3]. This allows players to buy packs of heroes with four random skills, create a deck of heroes, and battle them against other players. With this being an information-based game, it was implemented on the Secret Network for data privacy.

Our dApp contains a mix of different features from the discussed related works. It allows for minting, questing, and upgrading pets, like Axie Infinity and Aavegotchi. It also allows for battling other pets while privatising information to ensure fairness, similar to Secret Heroes.

3 Architecture and Contract Design

The high-level architecture of our implementation consists of a front-end which interacts with our main contract, and two token contracts using the SNIP-20 and SNIP-721 protocols [4, 5]. A detailed diagram showing components, messages, and dataflow can be seen in Figure 1.

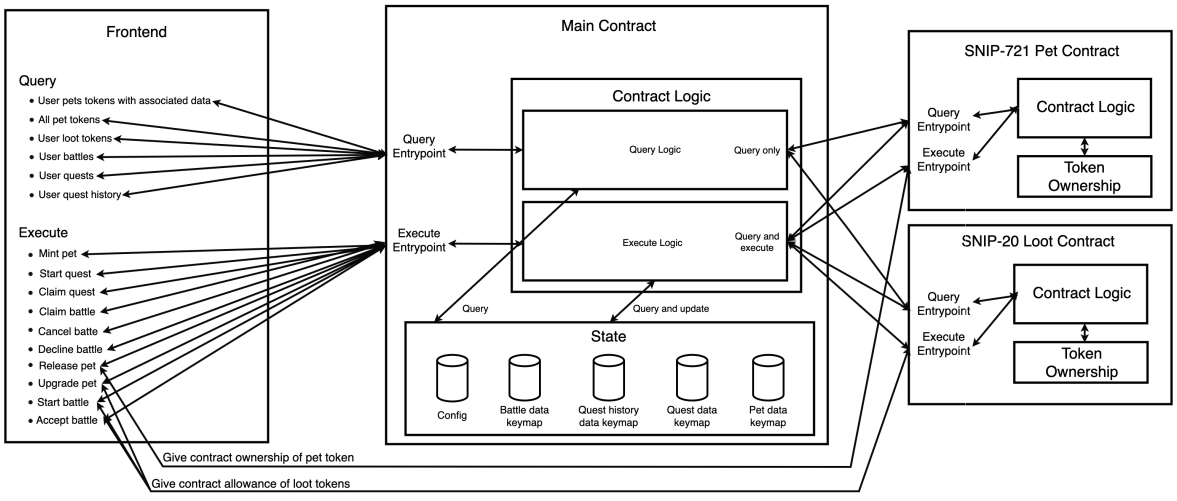

Figure 1: Diagram showing interaction between the fronend, and the main contract, SNIP-20 loot contract, and SNIP-721 pet contract.

Our queries are split into two groups, with permit and without permit. Queries for all pet and loot token ids and do not require permits, but all other queries require permits, as they access personal user data. Many query and execute messages require multiple permits, one for each contract, because we often need to forward a second or even third query with a permit to the SNIP-721 and SNIP-20 contracts for validation purposes. To simplify the front-end, the main contract handles and forwards

almost all messages to the loot and pet contracts. There are, however, some cases where our contract does not have full authority over the SNIP-721 and SNIP-20 contracts, and instead the front-end had to send messages directly to these contracts before executing a message on the main contract.

In terms of storage, the main contract stores all config data, pet data, quests, quest histories, and battles on-chain. The ownership of pets is stored by a SNIP-721 contract, and the ownership of loot is stored by a SNIP-20 contract.

There is also a strict ownership checking policy within the main smart contract. For example, to send a pet on a quest, we first have to check that the pet is owned by the user sending the execute message. Since our contract only manages the associated data of a pet, and not the ownership of a pet itself, we first have to query the SNIP-721 contract to check for ownership of the specified pet before further action is allowed. This is why, as discussed previously, many messages required SNIP-721 and SNIP-20 permits to be passes alongside the main contract permit.

The general flow of handling each message goes as follows. There is a number of validation checks, and each check ensures one of the following; pet ownership, pet state, quest state, user LTK balance, or battles state. If a check fails, we simply return early with an error message. If all checks pass, a block of logic will be run, followed by a returned transaction or answer message, depending on the type of message we are handling.

4 Frontend Workflow and UX

The front-end first requires the user to connect their wallet using keplr, and accept a permit for each of the three contracts. Permits are stored in local storage, so the user is not required to constantly permit queries. Had we not done this, the UX would be extremely poor, since the current front-end queries the contract for all user data around every five seconds.

Transactions generally requires a single user sign off using the keplr wallet. This is not the case for a few transactions however. A trade-off of giving users true ownership over their digital assets, is that the main contract has no authority unless the user explicitly tells the SNIP-721 or SNIP-20 contracts that the actions are acceptable. This means that when the user wants to release their pet, they must first make the contract the owner, and then the contract can release it on their behalf. This is also true when the user wants to use their loot tokens. Since the contract must remove loot tokens from the user's account, we must first give the contract an allowance of loot tokens. For this reason, any actions which require the removing of loot tokens, or pet tokens, will require sending a transaction directly to either the SNIP-20 or SNIP-721 contract.

All contract querying errors are shown to the user through a toast popup. Some execute errors are displayed to the user, if they are difficult to see when using the application. For example, if you fail to release a pet, or if you fail to start a battle with another pet. Making better use of the returned values within transactions would definitely be a worthwhile next step for an improved user experience.

5 Implementation Choices and Trade-Offs

Using multiple contracts was a key implementation decision that was made, and had a large trade-off. Using multiple contracts was important because we wanted true ownership of loot and pet tokens. This was also an important decision to make, because it allowed us to rely on existing and well-tested code, rather than starting from scratch. The downside is that our contract does not have any authority over these tokens, which leads to poor user experience, as discussed previously.

Our main contract has to store a large amount of state, this consists of the following:

- pet ids and their pet data
- user addresses and their quests
- user addresses and their quest history

- pet ids and their battle ids
- battle ids and their battle data

I have used key maps for each of these, as key maps allow for efficient querying and storage on block chains. In some cases, it has been necessary to store data in lists. Specifically, we have used lists to store the quests for each canonical address, the quest history for each canonical address, and the battle id for each pet id. This is acceptable for quests, as each use has at most four of them. On the other hand, quest history and battle ids have no limit on their size, and in retrospect there was likely a more efficient way to store these than a simple list.

6 Future Work and Roadmap

This prototype was built to allow for the extension of a marketplace, which would allow for the trading of our pet SNIP-721 and loot SNIP-20 tokens. A simple extension would be developing this marketplace, and allowing for the trading of our pets and loot tokens. Further extending this, IBC integration would allow for the exchanging of our loot and pet tokens pet SNIP-721 and loot SNIP-20 tokens for tokens on other chains outside of the Secret Network.

While this prototype used multiple contracts, all of the logic for the application was contained within the main contract. Further development could split this into multiple contracts itself. For example, one contract for managing pets and loot, one contract for battling, one contract for questing, and one contract for the future implemented marketplace.

Other application features may include using on-chain randomness to generate genes for which determine the look of a pet, and extending the pet battling to be more interactive and less deterministic. Random gene generation, along with breeding, is something we found in the related work section, with the likes of CryptoKitties and Axie Infinity. While our implementation requries privacy of data for battling, taking inspiration from Secret Heroes and Axie Infinity for card game style battling may be interesting. If a card game style battling system was explored, it would also be interesting to see if it is possible to improve the UX and not require explicit confirmation using keplr after each move.

References

[1] S. Mavis, "Axie infinity whitepaper," November 2021, accessed May 28, 2025. [Online]. Available: https://whitepaper.axieinfinity.com/

[2] Aavegotchi, "Aavegotchi whitepaper v1.1," n.d., accessed May 28, 2025. [Online]. Available: https://docs.google.com/document/d/186zOapKeHNNJ9y8LIByQQ64rs0eJUlEF/edit

[3] "Secret heroes: Nft-based card battle game," https://ethglobal.com/showcase/secret-heros-6ym1n, accessed May 28, 2025.

[4] S. N. D. Team, "Secret tokens (snip-20) - secret network introduction," https://docs.scrt.network/secret-network-documentation/development/development-concepts/create-your-own-snip-20-token-on-secret-network, 2024, accessed: 2025-05-29.

[5] S. Foundation, "Snip-721: Secret non-fungible token standard," https://github.com/SecretFoundation/SNIPs/blob/master/SNIP-721.md, 2021, accessed: 2025-05-29.