Non-Fungible Programs: Private Full-Stack Applications for Web3

Blake Regalia[*] Benjamin Adams[†]

April 5, 2024

Abstract

The greatest advantage that Web3 applications offer over Web 2.0 is the evolution of the data access layer. Opaque, centralized services that compelled trust from users are replaced by trustless, decentralized systems of smart contracts. However, the public nature of blockchain-based databases, on which smart contracts transact, has typically presented a challenge for applications that depend on data privacy or that rely on participants having incomplete information. This has changed with the introduction of confidential smart contract networks that encrypt the memory state of active contracts as well as their databases stored on-chain. With confidentiality, contracts can more readily implement novel interaction mechanisms that were previously infeasible. Meanwhile, in both Web 2.0 and Web3 applications the user interface continues to play a crucial role in translating user intent into actionable requests. In many cases, developers have shifted intelligence and autonomy into the client-side, leveraging Web technologies for compute, graphics, and networking. Web3's reliance on such frontends has revealed a pain point though, namely that decentralized applications are not accessible to end users without a persistent host serving the application. Here we introduce the Non-Fungible Program (NFP) model for developing self-contained frontend applications that are distributed via blockchain, powered by Web technology, and backed by private databases persisted in encrypted smart contracts. Access to frontend code, as well as backend services, is controlled and guaranteed by smart contracts according to the NFT ownership model, eliminating the need for a separate host. By extension, NFP applications bring interactivity to token owners and enable new functionalities, such as authorization mechanisms for oracles, supplementary Web services, and overlay networks in a secure manner. In addition to releasing an open-source software development kit for building NFPs, we demonstrate the utility of NFPs with an interactive Bayesian game implemented on Secret Network.

1 Introduction & Motivation

In the decade since the development of the Ethereum network, the promise of decentralized applications powered by smart contracts has been touted as an integral part of the vision of Web3.[1] The vision idealizes myriad decentralized applications (dApps) spanning finance, gaming, healthcare, real estate, energy, censorship-resistant social media, and the metaverse. In contrast to this aspirational and wide-ranging vision, the most notable real-world successes for smart contract-enabled applications have focused on decentralized finance (DeFi), where self-custody and trading of fungible cryptocurrency tokens can happen without the role of an intermediary bank.[2] The recent boom (and subsequent bust) in markets for non-fungible tokens (NFTs) used as signifiers of ownership for digital art, mark a more mixed result for the technology.[3,4] On the one hand it demonstrated that smart contracts can have utility for applications with broad appeal beyond finance. However, the phenomenon of "right click save" clearly showed that value comes not only from a digital certificate of ownership but also possession of the content of the owned thing as well.

[*]Solar Republic LLC, Washington, USA - blake.regalia@gmail.com

[†]University of Canterbury, Christchurch, New Zealand - benjamin.adams@canterbury.ac.nz

Beyond these applications, the promise of Web3 has largely been unrealized. The user experience, adoption, and functionality of decentralized versions of applications often compare poorly to more centralized Web 2.0 applications.[5] Arguably the greatest successes of Web3 have been in new types of applications outright (such as DeFi), which did not exist previously, more so than refactoring existing Web applications to run on the blockchain. This is because the added value of self-custody and decentralized automation that blockchains and smart contracts provide rarely overcomes deficits in usability, especially when compared to web applications that build upon a suite of mature technologies to achieve heavy computation, high-end graphics processing, privacy, secure transmission of data, and responsive interactivity.

Confidential smart contract blockchain networks encrypt contract memory and data stored on chain allowing us to build new types of dApps that are greater than the sum of their Web 2.0 or Web3 parts.[6,7] In Web 2.0, transport layer security (TLS) is an integral technology for applications because it encrypts transmitted data, adding trust for users and facilitating regulatory compliance. Without this level of security and privacy much of the modern Web would not exist. Still, many Web applications have a shared global state stored on a centralized server, which relies on a trusted administrator. Confidential smart contracts align smart contracts with technologies like TLS because the computation on-chain is privacy-preserving providing similar guarantees for on-chain computation that TLS does for off-chain communication. This adds a layer of security and privacy to decentralized applications that require trustless automation. However, although applications that combine a Web frontend with confidential smart contracts exist, the frontends are invariably hosted on a centralized server.

The solution presented in this paper is to introduce the concept of the Non-Fungible Program (NFP), an extension of the non-fungible token (NFT). An NFP represents a privately-held token (NFT) which grants its owner exclusive access to a hidden-state, decentralized application that uses smart contracts for its backend and self-contained Web documents for its frontend. Key contributions presented in this paper include:

- A method for encapsulating self-contained applications using SVG documents that transform into HTML5 web applications with exclusive access to private modules and assets stored in a confidential backend smart contract.
- An on-chain package manager system for executable code that hosts immutable and always-accessible versions.
- A software development kit for NFPs and demonstration running on Secret Network.

2 Related Work & State of the art

2.1 Progressive Web Apps (PWAs)

Progressive Web Apps (PWAs) were coined as a marketing concept but have coalesced around a set of common Web technologies that allow applications to implement features for consistent user experience.[8] The growth of PWAs came out of a desire to make Web applications on mobile platforms match the features of native applications by being installable and runnable offline.[9] However, in contrast to native applications, PWAs are also an approach for cross-platform development, because browsers that can run Web applications are ubiquitous on network-enabled devices. Other key features of PWAs include push notifications and background synchronization. PWAs are made possible by two key Web technologies: Service Worker support in browsers and HTTPS.[10] Service Workers operate as proxies that can choose when to serve cached data instead of fetching from a remote host when completing a web request. Browsers require that Service

Workers run in a Secure Context,[11] which HTTPS satisfies. An application's source code and data are therefore able to function even if the client doesn't have an active internet connection, i.e., once the PWA has been installed.

2.2 Privacy for blockchains

Addressing privacy in blockchain networks is an active research area. Non-interactive zero-knowledge proof (ZKP) systems, such as Succinct Non-Interactive Argument of Knowledge (ZK-SNARK), Scalable Transparent Argument of Knowledge (ZK-STARK), and Bulletproof have become popular tools for building privacy mechanisms into networks.[12–14] In blockchains, zero-knowledge proof systems can probabilistically check that statements are true about transactions and balances, and have become integral features of privacy cryptocurrencies.[15] In smart contract networks, they can also be used for authentication and identity management, and to verify that a computation has occurred, opening up the opportunity to move expensive but verified computation off-chain.[16,17] Since many dApps also depend on oracles to provide off-chain data to smart contracts, zero-knowledge proofs can also prove the authenticity of a data source while preserving privacy.[18] Although ZKPs are a useful mechanism to provide many privacy features for users of blockchains, they are only suitable for a specific form of problems between a prover and verifier. Other classes of problems involving private data from multiple sources—e.g., a shared program memory state contributed to by multiple users each with partial knowledge—require different solutions.

Confidential smart contract networks approach the problem of privacy in blockchains in a complementary but fundamentally different way. Confidential smart contracts are general purpose blockchain programs that enable some form of default privacy for contracts as they run (in contrast to network-level transactional privacy).[19–21] Active programs do not reveal internal state to an outside observer, including an administrator of the node executing the code, and the data stored on chain is encrypted in a format only readable by the contract. Currently, the most practical implementations of confidential smart contracts rely on the use of hardware secure-enclave encryption to achieve confidentiality. Two networks—Secret Network[6,22] and Oasis Network[7]—have working mainnets using this approach. Secret Network is a Tendermint-based network using a heavily-modified version of the Cosmos Internet of Blockchains software development kit.[23] Smart contracts on Secret Network are written in Rust and stored as Web Assembly (WASM) binaries. Oasis Network uses a parachain model with public and private networks built on an extension of the Ethereum Virtual Machine (EVM). Although the networks differ in many ways, both networks use Intel Software Guard Extensions (SGX) to execute contract code inside a secure trusted execution environment (TEE) for private computation.[24] Reliance on hardware for confidentiality is not without risks, as exploits can arise due to the setup of secure enclave keys, side-channel attacks, patching regimes for nodes on the network, and access pattern leakage.[25]

Confidential dApps vary in terms of long-term and short-term privacy obligations.[26] For example, in a game of incomplete information between two or more players, the imperative of confidentiality only lasts as long as the game is being played, so as not to benefit any player. In contrast, the medical history of an individual would need long-term privacy guarantees. Beyond hardware-based encryption, there is current research to implement general-purpose confidential smart contracts using fully-homomorphic encryption (FHE), which would eliminate such risks.[27] The NFP model we present here is not dependent on any one type of confidential smart contract implementation. However, hardware confidentiality is likely to remain the most viable option for developing NFPs in the near term, because even as these new methods are developed, they will be far less computationally-efficient than TEEs.

2.3 Non-fungible tokens

A non-fungible token (NFT) is a digital representation of ownership over a unique asset recorded on a blockchain.[3] The rules for minting, transferring ownership, and authenticating NFTs are governed by smart contracts. Non-fungible token standards, such as ERC-721 on the Ethereum network, are used to define common behavior for NFTs enabling them to be bought and sold on marketplaces.[28] Digital content that an NFT refers to can be stored entirely on chain, but commonly only metadata is stored on chain and will reference content stored off chain. In the case of digital art NFTs this external content is often stored on peer-to-peer data networks, such as the InterPlanetary File System (IPFS), in an effort to decentralize distribution. All aspects of NFTs stored on public blockchains are publicly readable, including full ownership history as well as any associated content stored off chain.

Secret NFTs are NFTs that are created using the SNIP-721 standard on Secret Network.[29] Other confidential smart contract networks such as Oasis Network also have the capacity to implement Secret NFTs, though no equivalent standard exists, yet. Because token data and metadata are encrypted, owners of Secret NFTs are able to irrevocably maintain exclusive access to digital content and prove ownership in zero-knowledge exchanges.

2.4 Web application categories

Analogous to the enabling role that Service Workers and HTTPS play for PWAs, confidential smart contracts are foundational for implementing NFPs. We compare Web applications across four qualifiers as follows:

- Decentralized – Is the backend service neither operated nor controlled by single authority?
- Private – Is the backend storage of user data provably private?
- Hostless – Does access to the content/application critically rely upon a connection to an active Web provider? Or is the content accessible without an active Web provider?
- Computable – Is the frontend capable of client-side computation?

Table 1 compares the various Web application types using these qualifiers. Single-page applications (SPAs) and PWAs are indicative of Web 2.0's capacity to provide computable functionality on the Web. DeFi, Secret DeFi (DeFi with confidential smart contracts), NFTs, and Secret NFTs are Web3 artifacts that vary in terms of hostless-ness and frontend computability. The NFP model we propose in the following section satisfies all four qualifiers.

	Decentralized	Private	Hostless	Computable
SPA	×	×	×	✓
PWA	×	×	✓	✓
DeFi	✓	×	×	✓
Secret DeFi	✓	✓	×	✓
NFT	✓	×	✓	×
Secret NFT	✓	✓	✓	×
NFP	✓	✓	✓	✓

Table 1: Comparison of application types on the Web.

3 Non-Fungible Program Model

In this section we detail the NFP model, covering the methods involved in creating a decentralized, private data service backend, the computable Web application frontend that runs in a browser, and the interface to couple them together. As a reference for the following section we provide a concept diagram illustrating some of the key elements of the model and their relationships for a sample NFP deployment in Figure 1.

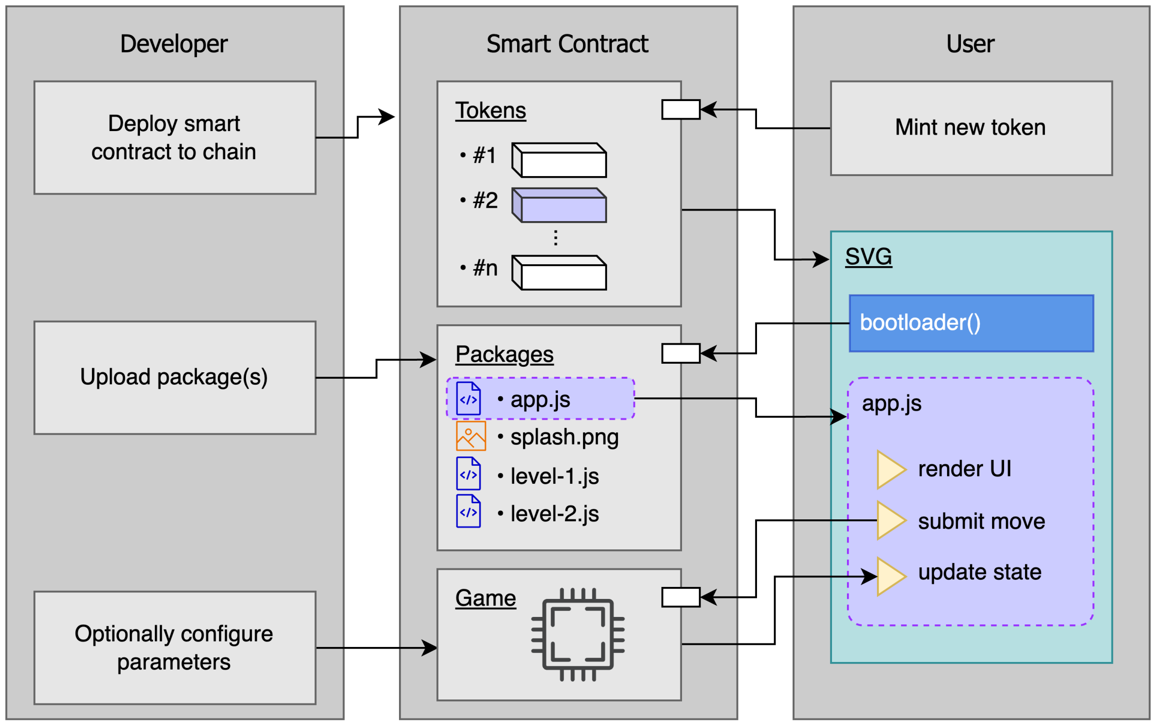


Figure 1: Schematic diagram of a sample NFP game deployment and its execution, which shows the interactions between the developer, the on-chain smart contract, and the user.

3.1 Ownership

Unlike other digital assets which can be replicated without limitation, one of the fundamental tenets of NFTs is that they offer verified proof of authenticity and proof of ownership over a particular digital item or piece of content. Moreover, an NFT collection can be designed to allow for ownership to be transferred, or forbid it entirely. Since transferring an NFT executes its contract, collections will often enforce custom rules, restrictions or behaviors on transfer events such as minimum qualifications, trading windows, or royalties, respectively. These extensible regulations on ownership are what allow NFTs to serve such a wide variety of use-cases. As described in Section 2.3, Secret NFTs built using confidential smart contracts enable exclusive access to content. NFPs adopt the Secret NFT ownership model.

Ownership is enforced by the NFP contract. New NFPs can be 'minted' by authorized minter accounts executing a mint transaction, the specifics of which depend on the application. As NFPs are minted, an encrypted index of owners is maintained. The contract provides logic to transfer token ownership and to enforce access permissions to view private, exclusive data. For Secret NFTs, data can be stored entirely on-chain or in the case of large files, a secret key can be stored on-chain to provide access to off-chain encrypted data. With NFPs, storage of the main Web document and associated packages are stored fully on-chain (details below).

3.2 Backend & Infrastructure

NFPs critically rely on the concept of confidential smart contracts, which by design are immutable and keep their active memory state hidden. This enables the development of trustless backend services that can execute logic based on a global encrypted state. Smart contract code is executed and new blocks are proposed by nodes within the blockchain network. A full exploration into blockchain network architectures is beyond the scope of this article. However, both Secret Network and Oasis Network have a set of validator nodes that use a Proof-of-Stake consensus mechanism to verify new blocks written to the chain. Public API endpoints exist to receive encrypted, signed transactions from external applications and return responses, and anyone can upload and interact with contracts on the chain. NFPs rely on such an infrastructure to provide decentralization. All of our experiments using the NFP model have been tested on the Secret Network testnet (Pulsar-3) infrastructure, but future work will expand to other networks.

3.3 SVG

Scalable Vector Graphics (SVG) is a broadly-supported, versatile XML-based file format developed by W3C that encapsulates a broad spectrum of visual representations.[30,31] In its simplest form, an SVG document can be rendered as a static image scalable to any resolution. The static image representation is most commonly seen in the thumbnail preview of file browsers or image galleries, or when used as an icon in application launchers. We refer to this most basic viewing mode as the "First Frame Preview" of an NFP, acting as a splash page to the interactive content within. Ideally, the preview also conveys what unique characteristics of the token make it non-fungible, in the same manner that JPEGs do for classic NFT collections.

The Synchronized Multimedia Integration Language (SMIL) is a W3C markup language capable of adding transitions, animations, and a limited range of interactivity to SVG documents.[32] Some operating systems' default image previewer will play SMIL animations for SVG documents, but disable CSS animations and prevent scripts from executing. Similarly, SMIL animations are enabled when an SVG is linked or embedded in an HTML document no matter how it is referenced, although SMIL interactivity will differ depending on the embedding method[1]. Given these capabilities and constraints, developers are free to embellish their SVG with dynamic content in what we refer to as the "Active Preview" viewing mode. For example, a collection of mythical creature NFPs might give each character a breathing animation in order to evoke a sense of liveliness. In some contexts, attribute changes and animations can be triggered in response to input events, such as mouse clicks, which allows for some basic interactivity. For example, clicking on a character's head might trigger an emote and reveal a speech bubble, adding to the interactive depth of the preview.

SVGs opened directly in a modern web browser, or embedded in an HTML document without sandboxing, typically have the ability to execute scripts. With scripting enabled, the SVG is able access the DOM and available Web APIs including Fetch, Crypto, WebRTC, and WebGL, to name a few. By appending an SVG `foreignObject` element to the document once opened, the web page can transform into a modern HTML5 web application.

When a downloaded NFP SVG document is opened directly in the browser, it is served from the `file:` scheme, imposing certain security restrictions. As the Secure Contexts specification advises, "the user agent SHOULD treat file URLs as potentially trustworthy",[11] meaning that

[1] `https://www.w3.org/TR/SVG2/conform.html#examples`

implementations may vary. This could affect an application's ability to leverage some advanced or experimental Web APIs including but not limited to Service Workers, Push, Geolocation, Web Authentication, Web USB, and Web Bluetooth. However, the breadth of features available to an application served from the `file:` scheme is powerful enough for most use cases we envision, including online gaming, social media, and marketplaces for digital content.

If an application depends on a Web API that requires a Secure Context, or if a user wants to audit or control the requests being made by an SVG, then a special sandboxing web application served over HTTPS can be used to host the SVG as it runs. In this paper, we do not evaluate sandboxing but plan to build such an application in future work.

3.4 Reliability

Running a Secret Network node requires SGX hardware to create Trusted Execution Environments. A collection of public API endpoints are hosted and funded by the community. Embedded into each token's SVG are a set of URLs pointing to API endpoints so that the frontend can communicate with the blockchain. Since the SVG stored on-chain is immutable, there is a risk of the embedded URLs becoming obsolete over the long-term. However, clients are free to locally modify an SVG's contents before launching it without compromising the application. In order to ensure these URLs are accessible for change, we introduce an XML namespace for NFPs and embed a set of configurable metadata properties in the SVG document in the form `<metadata><nfp:web nfp:lcds="..."/>`. These XML elements are designed to be both human-readable and machine-readable, the latter being useful for NFP sandboxes.

3.5 Packages

An NFP's SVG is a self-contained document; it does not use any native import methods to include assets from the web. All graphics, styling, and scripts are embedded directly into the SVG file and its contents get permanently written to the blockchain upon token mint. The on-chain contents are immutable by design. However, immutability poses a challenge to maintainability. NFP publishers may discover bugs in their application or may wish to add new features after launching a production mint. To address this shortcoming, we introduce the concept of a package manager as part of the NFP contract interface specification. With packages, admins of an NFP are able to publish revisions to parts of their application.

Analogous to traditional package manager systems, a *package* refers to a collection of *package versions*, where each version has its own data payload and metadata. Each package must be given an identifier that is unique within the contract. The identifiers do not need to be opaque nor human-readable since it is the responsibility of the developer to manage a naming scheme for their NFP. For example, a simple convention would be to use the basename of the file with its extension for readability, e.g., "main.js", "sprites.webp", "models.gltf". A more advanced system on the other hand might benefit from namespacing each package to avoid conflicts between manually uploaded and contract-generated packages.

In order to help guarantee token owners irrevocable access to application features, the contract code for the package manager ensures that all versions of a package are immutable and remain forever accessible to the owners of tokens for which they are designated. In our draft specification, we define four distinct *access specifiers* that determines how access-control is applied to a package. Once created, a package's access specifier cannot be changed in future versions. Keep in mind that *package* refers to a series of package versions immutably stored on-chain. We describe each of the four access specifiers below:

`public` – No authentication is required. Any client can anonymously query the contract to access the package.

`owners` – Only accounts owning at least one token are authorized to view the package.

`cleared` – Token owners must be individually approved to access the specific package. For example, a game may require the user to clear level 1 and prove it to the contract before they are allowed to fetch the module for level 2. Implementors are free to choose whether transferring an NFP resets a token's cleared status on a per-package basis.

`token` – The package was specifically created for an individual token, and that token's owner has exclusive access to the package. Whereas admins are able to publish and view packages having the other specifiers, they are explicitly forbidden from publishing or viewing packages with the `token` specifier. Consequently, the only means to publish a package having the `token` specifier is through contract logic. For example, a contract might generate a package for a specific token based on some rules and an internal secret. Another use case might be to provide token owners a private, decentralized, version-controlled file system.

When a client needs to retrieve a package's contents, it might not be desirable to always return the most recent version. There are a variety of use-cases that depend on multiple versions of a package being accessible simultaneously. We found that using a tagging scheme provides the most flexibility for these use cases. When uploaded, each package version can be assigned an optional set of tags, e.g., "latest", "1.x", "beta", and so on. When clients query for a package version, they can either specify the exact version by its serial number, or select the most recent version that includes a given tag, such as "latest".

3.6 Storage

Storage is a limited resource for decentralized applications. Most NFTs use third-party overlay networks such as the InterPlanetary File System (IPFS) to store the contents of digital assets.[33] Some Web3 applications have experimented with using IPFS to host their client-side code. However, IPFS also brings its own set of challenges when it comes to data longevity and accessibility.[34,35] For example, each file requires persistent seeds in order to be available on the network. Other factors such as peering and network topology can also affect clients differently, playing an important role in download bandwidth. As for hosting application frontends, current solutions require special software or augmented browsers in order to handle routing requests for lookups such as the Ethereum Name Service.

With our NFP approach, the application's frontend code is considered part of the token's digital asset, and it is privately stored directly on the same blockchain as the token. This approach eliminates the dependency on a third-party content hosting solution while preserving all the benefits of an immutable, decentralized storage provider. Additionally, it secures the contents automatically, i.e., without requiring a separate encryption layer. Secret Network was built using the Cosmos SDK[36] and extends the CosmWasm[37] smart contract platform, which uses a floating key-value store to persist contract state. For CosmWasm-enabled chains, the 'at-the-pump' gas cost per byte written to storage from a contract depends on several factors including current market value of the gas token and certain network parameters. Similarly, the maximum size of a bytestream that can be uploaded to a contract's storage in a single transaction depends on network parameters and can be affected by validator configurations. As we will discuss in Section 4, the effective costs and limitations observed on Secret Network currently make our storage solution feasible, but may impose greater challenges if network consensus were to change certain parameters. It may be the case that storing large assets in contract memory is prohibitive on other privacy chains. A possible workaround to upload size limitation we have explored involves chunking an asset into multiple transactions and then reassembling upon retrieval.

Given that storage in contract memory is a valuable resource for decentralized applications, each package version contains an attribute for content encoding, e.g., `gzip`. Consequently, the host environment running the package code should support the ability to decode compressed formats. Fortunately, the Compression Streams Web API provides this functionality natively in the browser.

3.7 Bootloading

Once an SVG is opened in a browser with the ability to run scripts, it transforms into an HTML5 web application which we refer to as the *Client*. The transformation has several discrete steps that we will describe in this subsection.

In all cases, an NFP Client will need to retrieve private data from the contract. To do that, it must encode RPC requests and submit them over the HTTPS transport. Querying and executing a confidential contract on the Secret Network requires encrypting the contents of the message that will get sent into the node's secure enclave where it will then be decrypted and forwarded to the target contract. Similarly, the Client must also decrypt the response data in order to read the results returned by the contract. This process requires several cryptographic functions that are not available in the Web Crypto API.

While the entire Client's source could be contained in the SVG file, a more flexible design pattern, as we mentioned in Section 3.5, is to load the application in modules by leveraging the contract's package manager. With this approach, each token's SVG can query the chain for the "latest" stable version of the application's entrypoint JavaScript module and evaluate it by appending a `script` element to the DOM. We refer to this process as *bootloading*, and the requisite script embedded in the SVG as the *bootloader*.

Using a bootloader and modules to defer loading the application this way also helps reduce the size of each token's SVG since the same module can apply to multiple tokens. A reduction in the size of the SVG ultimately has cost savings in terms of gas paid by users during mint operations. One trade-off to this approach is that it may take a few seconds longer to start up when the SVG is opened or reloaded as the bootloader has to wait for the query response from the network. A possible mitigation however is to cache the module locally upon initial download and then check if a newer package version is available in the background on subsequent loads.

3.8 Querying and Executing

With the NFP contract acting as a backend service to the Client, there are two types of requests that can be made, queries and transactions. Queries are the means to call functions on the contract that do not write data to the blockchain, but are still able to read from the contract's private database and perform confidential computing. On Secret Network, the query request and response are always encrypted, but clients must also authenticate with the contract to prove they have permission to view the requested information. Contracts may expose both public (unauthenticated) and private (authenticated) query methods to clients. One popular authentication technique involves the account owner digitally signing a document known as a Query Permit. A signed Query Permit grants any client with it in their possession the ability to perform some specified set of read-only queries on behalf of the signer. For example, Alice signs a Query Permit designating contract X and the ability to perform all private queries. She then transfers the signed Query Permit to Bob who submits it with his query to contract X asking for the token balance of Alice's account. Contract X verifies the signature on the document using Alice's public key, checks the permissions granted, and returns Alice's token balance. As long as the authentication passes, whether Alice or someone else submitted the query is inconsequential to the query process. Query Permits can later be revoked by executing a transaction.

Executions are used to mutate the contract's internal state. An execution is carried out by writing a transaction to the blockchain. Like queries, executions are also able to read from the contract's private database and perform confidential computing. Unlike queries, executions are automatically authenticated since each transaction message requires the account owner's digital signature. Whereas queries can be served by the secure enclave within any compliant, participating node on the network, executions can only be performed by validator nodes securing the network through proof-of-stake. The amount of confidential computing time an individual query or execution may consume is regulated in terms of gas consumption, but the gas used for an execution must be paid for while the gas used for queries do not.

For supporting the combination of queries and executions, the Client must perform cryptographic operations that are not natively supported by the browser or any Web API. At minimum this includes Bech32 encoding/decoding,[38] elliptic curve scalar multiplication on Curve25519,[39] RIPEMD-160 hashing,[40] AES-GCM-SIV,[41] Secp256k1[42] key generation, ECDSA[43] (elliptic curve digital signature algorithm) for signing/verification and ECDH (elliptic curve Diffie-Hellman) for asymmetric encryption. As we discuss in Section 3.10, we publish a runtime library for NFP Client developers covering all the aforementioned operations, including common abstractions for interacting with Secret Contracts and other modules on the Secret Network.

3.9 Fee Grant and Delegation

When executing contracts from the Client, a potential point of friction for the user experience in NFPs is the frequency of transactions that must be signed by a compatible wallet representing the token owner's account. For example, every move in a game of Chess might require the player to sign a transaction on their air-gapped hardware wallet, making the experience inconvenient enough to avoid entirely. Here we present a solution that combines a native network feature with smart contract authorization by creating a 'hot' hot wallet in the browser to execute transactions on behalf of the token owner without prompts.

Cosmos SDK provides a mechanism called Fee Grants that allows users to pay their transaction fees from the balance of another account. The granter signs and broadcasts a transaction approving some grantee with an optional spending limit and expiration. When the Client loads, it generates a securely random private key for a new hot hot wallet account and saves it to local storage. It then requests a Fee Grant approval to grant the new account some limited spending allowance, ensuring that the actual balance of the new account remains nil in case it is compromised. The spending limit ensures that the entire balance of the granter isn't at risk of being wasted on transaction fees.

The hot hot wallet account also needs authorization to execute contract methods on behalf of the token owner. Rather than granting the new account unlimited discretionary power, our approach with the NFP contract interface specification defines a set of operations to approve or revoke delegate accounts on a per-owner or per-token basis. For example, the Client for an NFP chess game might request token delegate approval to execute only a limited subset of methods, such as creating new games, joining existing games, and submitting moves.

In summary, Fee Grant and execution delegation provides reasonable security such that if the hot hot wallet account of the Client were to be compromised, an attacker would not be able to transfer ownership, steal funds, or drain the owner's account balance.

3.10 Supporting Development

An important consideration for NFPs as a platform is the developer experience. Streamlining the build process, making reusable components, and defining interfaces are crucial elements to fostering resources for developers. As part of our work, we have published open-source tooling and draft specifications to make these resources available to developers from the start. Namely, we define a contract interface specification for NFP contracts that includes query and execution methods for ownership, transfer-ability, delegation, package management, private notifications, and key-value storage. On the Client-side, we define an XML namespace and element schema to provide SVGs with human-readable and machine-readable metadata, including configuration for API endpoints.

For the development of scripts for the Client, a custom NFP SDK provides a suite of tools designed to automate best practices and parts of the build process. For example, the SDK exports a Vite plugin allowing developers to use ES import and export syntax to create pseudo-modules that work across package boundaries. By default, the plugin also strongly favors producing small distributables to be uploaded to the chain by bundling and minifying scripts and optimizing the SVG. Finally, as mentioned in 3.8, an NFP runtime library provides all the functionality needed to transact with Secret Contracts, including bootloading and private notifications.

4 Evaluation

We evaluate our approach by implementing a two-player, turn-based Bayesian game as an NFP.[44]

4.1 The Game

Inspired by the classic game Salvo and derivatives such as Battleship, two players compete in a zero-sum game by taking turns submitting a coordinate to attack on their opponent's 10x10 grid. At the beginning of the game, each player configures five stationary 'vehicles' of varying lengths to occupy the available cells of their own grid which is hidden from their opponent. The objective is to be the first to destroy all five of their opponent's vehicles. Each vehicle can be placed on the grid either horizontally or vertically, must be entirely contained within the grid, and cannot overlap with other vehicles. After submitting each attack, the player is informed whether or not it struck a cell occupied by a vehicle, i.e., either hit or miss. Once a player has completely destroyed a vehicle by striking all of its cells, the vehicle's type and the coordinates it occupied are revealed to the player.

4.2 Requirements

In order to satisfy the four dimensions—decentralized, private, hostless, and computable—of an NFP, the implementation has the following requirements. First, minting of a game NFP can be done via CLI or a public website that requires payment in SCRT to the NFP smart contract. The newly minted NFP is then governed by the NFT ownership model: the buyer can be verified as the exclusive owner of the NFP and transferring is done via contract execution. All asset contents (including SVG data) are stored on-chain. Upon minting, the owner of the NFP can then download the SVG to their local device. For each NFP that is minted, its SVG image is unique. When opened in a browser it will render an animated splash page for the game (Active Preview) and a button will appear to connect to the chain and load the game package. When the package is received, the game loads with a list of open matches that have been started by other NFP owners which the player can join, as well as an option to initiate a new match and wait for another player to join. When initiating a new match, an optional wager in SCRT can also be sent to the contract, which must also be met by any joining player. Once joining (or another player joining the owner's game), a new screen appears for both players to set up the game board. Invalid board configurations are rejected by the backend contract. After both players have set up their boards, the match turns begin (with one player randomly chosen to start). Players take turns attacking a cell in the 10x10 grid and the result (a hit or miss) is returned. Vehicle types are only revealed once they are fully destroyed. Turns are enforced by the backend contract and out-of-turn submissions are rejected. Whenever an attack is made, in addition to the hit or miss for the attacker, the opponent is also notified of the event, updating their game state. Once all vehicles are destroyed for one player the match ends and any wager yield is sent to the winner. At any point during play, the state of an open match persists across client restarts and can be recovered by simply reopening the SVG in the browser.

The game application is 1) decentralized—game state is handled by smart contracts running on Secret Network; 2) private—global game state is only known to the confidential contract, allowing for the implementation of a Bayesian game; 3) hostless—the game can be run by the owner (and only by the owner) simply opening an SVG file in the browser; and 4) computable—the game has an interactive interface with multiple views using HTML5 Web graphics.

4.3 Implementation

Here we briefly describe how our game implements all of the features mentioned in Section 3. The NFP is first minted by the user on a public website or by CLI, granting them exclusive ownership over a new, unique, private digital asset. The SVG file is then automatically downloaded to their device where they can open it directly in their browser. The SVG contains XML that configures a set of failover API endpoints. The SVG's bootloader fetches the *latest* `app.js` package from the contract and injects it into the document, transforming into a web application. The now functioning Client then subscribes to a private notifications message channel which is how it will receive updates to the player's game state. A custom designed hot wallet UI component guides the user through requesting a Fee Grant and authorizing the hot wallet account as a token delegate. Once approved, the user can start a new match or browse open matches to join in the lobby. Once a game has started, players configure and submit their grid setup to the contract and begin exchanging attacks one at a time, each execution being signed automatically by the hot wallet. A monorepo containing code to build the SVG template, the source for the game packages, and smart contract backend is available here: `https://github.com/nfps-dev/nfp-examples`. Figures 2–5 in the Appendix show screenshots of the implemented game.

4.4 Discussion

The evaluation game shows that NFPs are a workable model for building decentralized, private, hostless Web applications that can easily match the functionality of a traditional HTML5 Web 2.0 game. The development of design patterns and tooling for NFPs will be important to not only facilitate development (beyond what we've discussed in Section 3.10) but also to better identify best practices for managing privacy across the front and backend of the application. This is not a trivial endeavor as each application will have unique privacy guarantee requirements. Furthermore, it should be noted that the degree of decentralization of an NFP depends on the network it is running on. For an application like the one detailed here, a validator set of 75 (Secret Network's active set as of March 2024) is sufficiently decentralized for all intents and purposes, but there are other cases where this might be considered inadequate.

The size of the applications that can be stored on chain remains a limiting factor. At time of writing, the block gas limit on Secret Network is set at 6 million gas units, which enforces an upper bound on the amount of data that can be stored on chain in a single transaction. In our experiments we found that gas used to upload packages scales linearly with the size of package, reaching a current limit of 320KB after compression. While queries do not cost gas, they are still metered, but since reading is much less expensive than writing, any package written to chain will be accessible to download. The size limit places a restriction on code that can fit in a single package, but with optimization and compression enough leeway exists to build interactive web applications, such as the game described above. If required, the package manager could be extended to allow chunked packages to be downloaded over multiple query requests.

The minted SVG file was 23 KB. Our primary application bundle which gets loaded from the package manager was 338KB before compression – 60% of that going to ambitiously high-resolution graphics – and 227KB after gzip compression. Keep in mind, the bundle includes all game assets, client and library logic, styling, DOM structure, and UI text for the application.

This proof of concept demonstrates the basic utility of the NFP model. Some potential applications we envision for NFPs include:

- Interactive digital art and graphics which access high-end browser-runnable code such as shaders and web assembly.
- Games that reveal encrypted content based on program state and input events.
- Scalable, trustless, and decentralized eSports and competitive multiplayer gaming with cheat resistance.
- Censorship-resistant digital content sharing/distribution and Web3 clients.
- Applications, such as task workers, with embedded wallets to automate interaction with smart contracts on (any) blockchain.
- Autonomous private overlay networks, such as private peer-to-peer communication networks authenticated via NFP or authenticated access to private oracle services.

5 Conclusion

In this paper we presented the NFP model for building full-stack Web3 decentralized applications. NFPs leverage existing Web standards and APIs with recent innovations in NFTs and confidential smart contracts to build hostless web applications that encapsulate the usability and interaction of modern Web 2.0 applications joined to a trustless, privacy-preserving backend database service. We demonstrated the utility of the model with an interactive, cheat-resistant and openly verifiable Bayesian game implemented as an NFP. Our next step will be to explore sandbox environments that can safely run NFPs that require access to full Web APIs.

References

[1] Buterin, V., *et al.* “A next-generation smart contract and decentralized application platform.” *white paper* **3.37** 2–1 (2014).

[2] Zetzsche, D. A., Arner, D. W., Buckley, R. P. “Decentralized finance (DeFi).” *Journal of Financial Regulation* **6** 172–203 (2020).

[3] Wang, Q., Li, R., Wang, Q., Chen, S. “Non-fungible token (NFT): Overview, evaluation, opportunities and challenges.” *arXiv preprint arXiv:2105.07447* .

[4] Hofstetter, R., *et al.* "Crypto-marketing: How non-fungible tokens (NFTs) challenge traditional marketing." *Marketing Letters* **33.4** 705–711 (2022).

[5] Murray, A., Kim, D., Combs, J. "The promise of a decentralized internet: What is Web3 and how can firms prepare?" *Business Horizons* **66.2** 191–202 (2023).

[6] "Secret network." URL `https://scrt.network/`.

[7] "Oasis Network." URL `https://oasisprotocol.org/`.

[8] Hume, D. *Progressive web apps*. Simon and Schuster (2017).

[9] Biørn-Hansen, A., Majchrzak, T. A., Grønli, T.-M. "Progressive web apps: The possible web-native unifier for mobile development." In *International Conference on Web Information Systems and Technologies*. **2** SciTePress 344–351 (2017) .

[10] Steiner, T. "What is in a web view: An analysis of progressive web app features when the means of web access is not a web browser." In *Companion Proceedings of the The Web Conference 2018* 789–796 (2018) .

[11] West, M. *Secure Contexts*. *Candidate recommendation* W3C (2021) https://www.w3.org/TR/2021/CRD-secure-contexts-20210918/.

[12] Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M. "Succinct {Non-Interactive} zero knowledge for a von Neumann architecture." In *23rd USENIX Security Symposium (USENIX Security 14)* 781–796 (2014) .

[13] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M. "Scalable, transparent, and post-quantum secure computational integrity." *Cryptology ePrint Archive* .

[14] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G. "Bulletproofs: Short proofs for confidential transactions and more." In *2018 IEEE symposium on security and privacy (SP)* IEEE 315–334 (2018) .

[15] Sun, X., Yu, F. R., Zhang, P., Sun, Z., Xie, W., Peng, X. "A survey on zero-knowledge proof in blockchain." *IEEE network* **35.4** 198–205 (2021).

[16] Yang, X., Li, W. "A zero-knowledge-proof-based digital identity management scheme in blockchain." *Computers & Security* **99** 102050 (2020).

[17] Buterin, V. "An incomplete guide to rollups." (2021) URL `https://vitalik.ca/general/2021/01/05/rollup.html`.

[18] Zhang, F., Maram, D., Malvai, H., Goldfeder, S., Juels, A. "Deco: Liberating web data using decentralized oracles for TLS." In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* 1919–1938 (2020) .

[19] Zyskind, G., Nathan, O., *et al.* "Decentralizing privacy: Using blockchain to protect personal data." In *2015 IEEE security and privacy workshops* IEEE 180–184 (2015) .

[20] Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C. "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts." In *2016 IEEE symposium on security and privacy (SP)* IEEE 839–858 (2016) .

[21] Cheng, R., *et al.* "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts." In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)* IEEE 185–200 (2019) .

[22] Woetzel, C. “Secret network: A privacy-preserving secret contract & decentralized application platform.” .

[23] Buchman, E. *Tendermint: Byzantine fault tolerance in the age of blockchains*. Ph.D. thesis University of Guelph (2016).

[24] Costan, V., Devadas, S. “Intel SGX explained.” *Cryptology ePrint Archive* .

[25] Jean-Louis, N., *et al.* “SGXonerated: Finding (and partially fixing) privacy flaws in TEE-based smart contract platforms without breaking the TEE.” *Cryptology ePrint Archive* .

[26] Casassa Mont, M. “Dealing with privacy obligations: Important aspects and technical approaches.” In *International Conference on Trust, Privacy and Security in Digital Business* Springer 120–131 (2004) .

[27] Gentry, C. *A fully homomorphic encryption scheme*. Stanford university (2009).

[28] “ERC-721: Non-Fungible Token Standard.” (2018) URL `https://eips.ethereum.org/EIPS/eip-721`.

[29] “SNIP-721: Private, Non-Fungible Tokens.” (2021) URL `https://github.com/SecretFoundation/SNIPs/blob/30a70a6bc71cc0d1711b5bc4d7b3b1a7a547625a/SNIP-721.md`.

[30] Lilley, C., *et al. Scalable Vector Graphics (SVG) 1.1 (Second Edition). W3C recommendation* W3C (2011) https://www.w3.org/TR/2011/REC-SVG11-20110816/.

[31] Jackson, D., Ferraiolo, J., Fujisawa, J. *Scalable Vector Graphics (SVG) 1.1 Specification. W3C recommendation* W3C (2003) https://www.w3.org/TR/2003/REC-SVG11-20030114/.

[32] Hoschka, P. *Synchronized Multimedia Integration Language (SMIL) 1.0 Specification. W3C recommendation* W3C (1998) https://www.w3.org/TR/1998/REC-smil-19980615/.

[33] Benet, J. “IPFS – Content addressed, versioned, P2P file system.” *arXiv preprint arXiv:1407.3561* .

[34] Daniel, E., Tschorsch, F. “IPFS and friends: A qualitative comparison of next generation peer-to-peer data networks.” *IEEE Communications Surveys & Tutorials* **24.1** 31–52 (2022).

[35] Doan, T. V., Psaras, Y., Ott, J., Bajpai, V. “Towards decentralised cloud storage with IPFS: opportunities, challenges, and future considerations.” *IEEE Internet Computing* .

[36] Tendermint “Cosmos SDK.” (2023) URL `https://tendermint.com/sdk/`.

[37] CosmWasm “CosmWasm.” (2023) URL `https://cosmwasm.com/`.

[38] Wuille, P., Maxwell, G. “Base32 address format for native v0-16 witness outputs.” (2017) URL `https://github.com/bitcoin/bips/blob/master/bip-0173.mediawiki`.

[39] Bernstein, D. J. “Curve25519: new Diffie-Hellman speed records.” In *Public Key Cryptography-PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings 9* Springer 207–228 (2006) .

[40] Dobbertin, H., Bosselaers, A., Preneel, B. “RIPEMD-160: A strengthened version of RIPEMD.” In *International Workshop on Fast Software Encryption* Springer 71–82 (1996) .

[41] Gueron, S., Langley, A., Lindell, Y. "AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption." (2019) RFC 8452 doi:10.17487/RFC8452 URL `https://www.rfc-editor.org/info/rfc8452`.

[42] Brown, D. R. L. "SEC 2: Recommended Elliptic Curve Domain Parameters." (2010) URL `https://www.secg.org/sec2-v2.pdf`.

[43] Johnson, D., Menezes, A., Vanstone, S. "The elliptic curve digital signature algorithm (ECDSA)." *International journal of information security* **1** 36–63 (2001).

[44] Mertens, J. F., Zamir, S. "Formulation of Bayesian analysis for games with incomplete information." *International journal of game theory* **14** 1–29 (1985).

Evaluation game screenshots

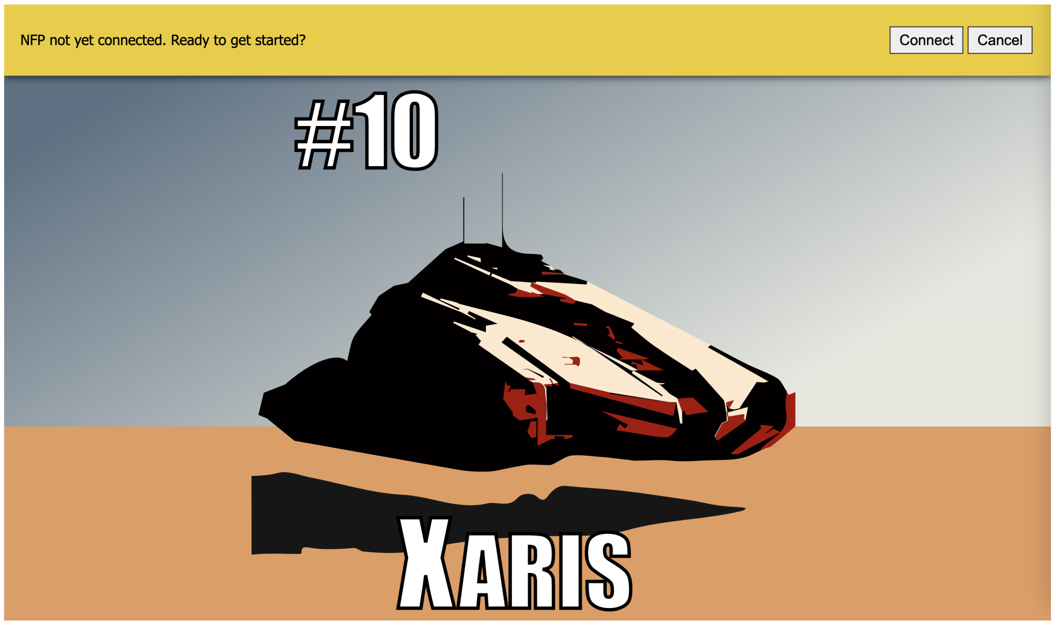
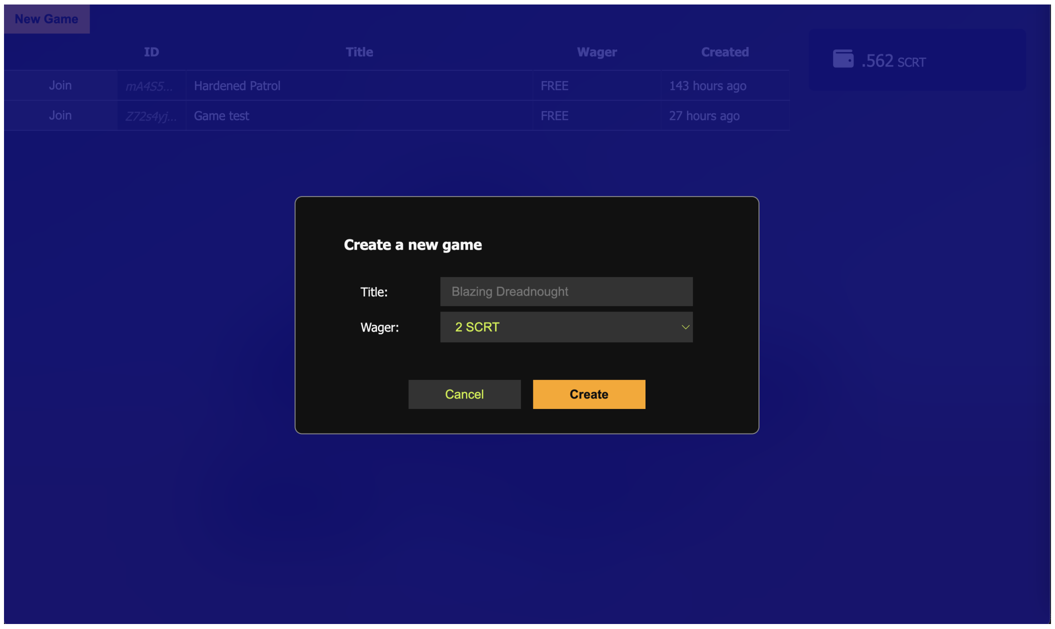


Figure 2: Screenshot of the active preview of the evaluation game NFP. This view is shown when the SVG file is loaded directly from the file system. The ship is animated to float over the desert floor, and the action bar at the top provides the user the ability to connect to the chain and run the game package stored on-chain in the NFP smart contract.



	ID	Title	Wager	Created
Join	mA4S5...	Hardened Patrol	FREE	143 hours ago
Join	Z72s4yj...	Game test	FREE	27 hours ago

Create a new game

Title: Blazing Dreadnought

Wager: 2 SCRT

Figure 3: Screenshot of creating a new game after connecting and running the on-chain package code. Behind the dialog overlay, the menu for selecting active games started by other NFP owners is visible.

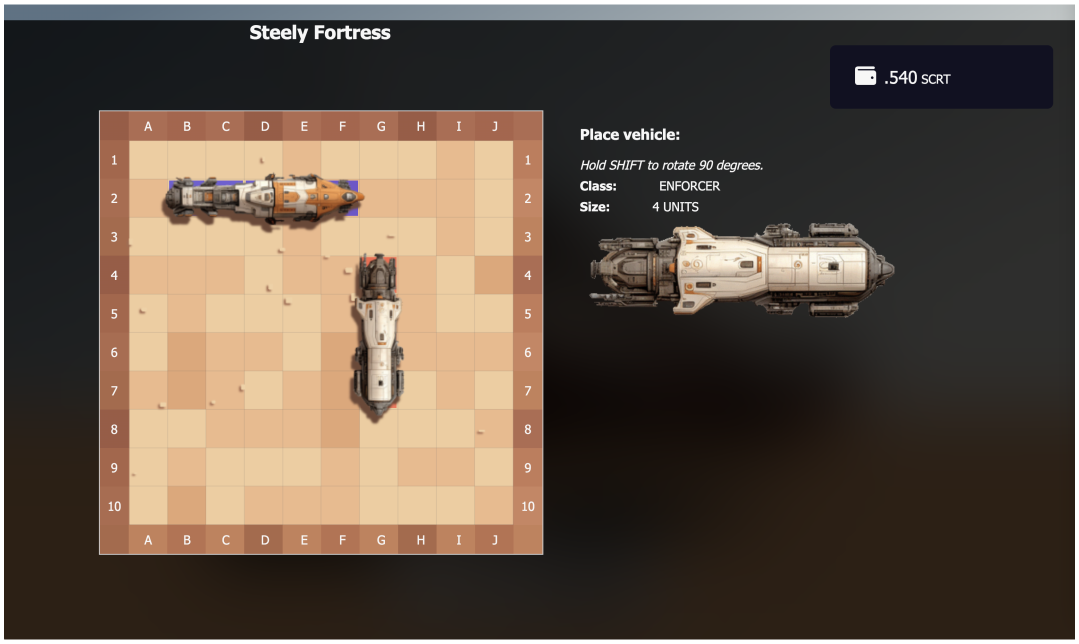


Figure 4: Screenshot of the board setup page for a new game where the player configures the placements of vehicles.

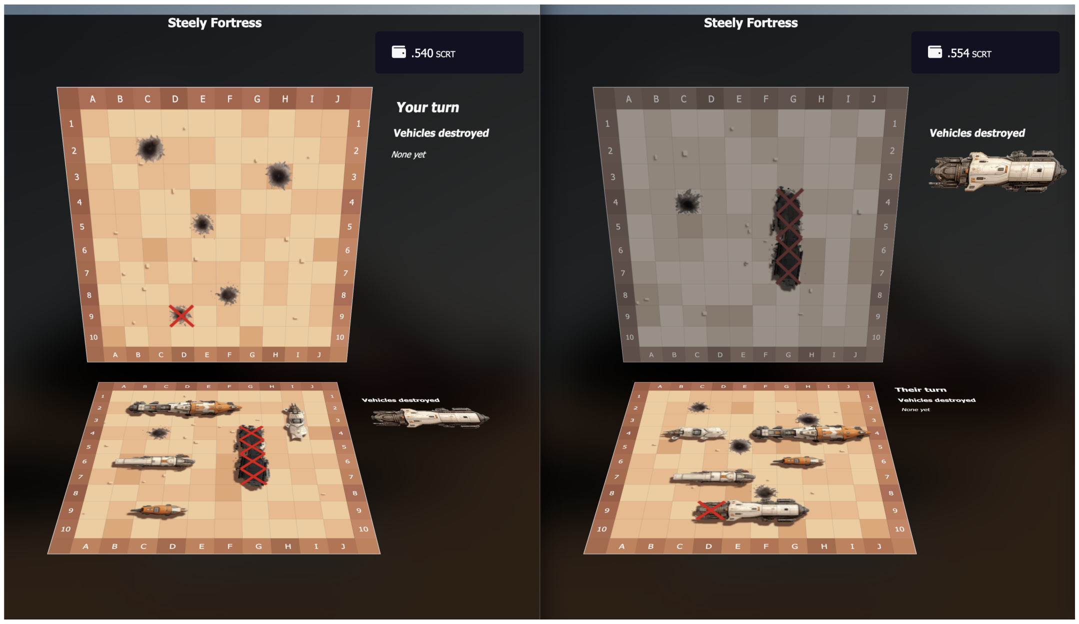


Figure 5: Screenshot of two competing players' screens shown side-by-side as one selects their next attack. The board is interactive with cell selection based on mouse over and click. The state of the game from the player's perspective is shown on separate 'home' and 'away' boards.