SNIP1155 is one of the token standards on Secret Network. A single contract can create an arbitrary number of tokens, including a combination of fungible tokens and NFTs. This eliminates the need for factory contracts, reducing cross-contract messages (which is expensive gas-wise), and provides useful features such as batch transferring a mix of NFTs and fungible tokens in a single tx. It benefits any dapp that needs to issue more than one token.

This is a proposal to upgrade SNIP1155 reference contract, comprising 4 items: 1) refactor to CosmWasm1.0, 2) add IBC features, 3) modularize reference, and 4) integrate with secret.js.

This will contribute to Secret Network's ongoing efforts to be more attractive to developers. Particularly it will give devs access to CosmWasm1.0 features, provide standardized IBC features for cross-chain dapps, simplify the process of maintaining and upgrading customized contracts, and streamline front-end development with native secret.js integration.

Table of contents

A recap on the benefits of SNIP1155

Use case: Games:

SNIP1155 is ideal for games. Games (including metaverse dapps) often need to create many tokens with various characteristics – some non-fungible and some fungible, and some a mix of features from both. SNIP1155 simplifies the design in such cases. The ability to batch-handle a mix of fungible tokens and NFTs in a single tx is useful for situations, for example, when a player wants to sell its game character (NFT) along with the items (a mix of NFT and fungible tokens) and game currency (fungible tokens) in one go.

Games are an important use case at Secret Network, as Secret Network's features make games inherently resistant against players using block-analysis to gain an advantage. Secret Network should aspire to attract more game devs for this reason, and improving SNIP1155 will be an important step.

Use case: DeFi:

Many DeFi dapps require managing a multitude of tokens, for example, LP tokens, derivatives, etc. This is often done using factory contracts that create customized tokens. If SNIP1155 is used instead, factory contracts can be eliminated.

Use case: NFT collections:

SNIP1155 can also be used as "pure" NFTs, and is particularly useful in creating collections. It offers an alternative standard to complement the SNIP721 & 722.

Improved dev experience:

Inter-contract messages and factory contracts are complex and prone to errors. Using SNIP1155 can substantially reduce design complexity and gas costs in transactions.

Improved user experience:

Users can view multiple balances from a single viewing key, and need fewer approval transactions (potentially just one approval) to cover all tokens for a dapp.

Detailed design

Scope overview

1. Upgrade contract reference implementation to CosmWasm1.0, including tests and reference/documentation
2. Add IBC functionality, allowing SNIP1155 to send and receive execute and query messages through IBC
3. Modularize reference implementation, to reduce complexity of customizing, maintaining and upgrading SNIP1155 compliant contracts
4. Merge secret.js extension, so SNIP1155 API becomes natively accessible within secret.js

Item 1: Upgrade to CosmWasm1.0

Refactor existing contract. The idea is to keep the base (non-IBC) API unchanged, but allow extensibility with CosmWasm1.0 features. Unit and integration tests need to be updated.

Item 2: IBC entry points and functionality

IBC-enabled SNIP1155 will allow cross-chain dapps with contracts on multiple blockchains to operate on Secret Network.

The added features will allow custom messages to be sent between chains. It could also allow users to transfer and query SNIP1155 tokens directly from their chains while retaining some privacy, as the tokens don't leave Secret Network. (Unlike transferring tokens through IBC the normal way, where privacy is lost. However, I don't anticipate simple transfers and balance queries to be the main use case of SNIP1155 IBC).

In order for to achieve this, we need to add new entry points and functions, modify existing functions and ensure IBC authenticated queries have some degree of privacy. We'll continue with simple illustration then elaborate on these points.

An illustration

Let's examine an example of a cross-chain game whose core game logic is encoded in juno_contract

, a smart contract sitting on a transparent blockchain. It also has a snip1155

contract deployed on Secret Network. There are two players on different chains: secret1xyz

and juno1xyz

. These two players have a mix of game currency (fungible gold

) and items (nft

s), which are all SNIP1155 tokens.

If secret1xyz

performs an execute function (eg: transfer gold to juno1xyz

), secret1xyz

sends a message to the SNIP1155 contract. The contract will update its ledger to reflect the new gold

balances of the two players. As you can see, the transaction has the full privacy of a native Secret Network transaction although juno1xyz

was involved.

Now, let's say an NPC (non-player character) mugs juno1xyz

of his rare item (an nft

). Since juno_contract

holds the core game logic, juno_contract

needs to send an IBC message to SNIP1155 so it updates its ledger such that juno1xyz

no longer owns the rare item. Once snip1155

receives this IBC message, authenticates it and performs the required actions to execute the game logic, and sends a success response which goes through the IBC channel again back to the juno_contract

. In this case, the contents of messages sent from juno_contract

will be viewable on the public blockchain (seeprivacy considerations).

Add IBC entry points

In our example above, when snip1155

was deployed, it needed to establish an open IBC channel with juno_contract

. Once done, it can communicate with snip1155

.

For this to work, SNIP1155 needs to have the required entry points to allow it to perform therequired handshakes as defined in the IBC specifications. In addition, once the IBC channels are open, SNIP1155 needs to be able to send and receive data packets over the channel.

These needs to be done according to the IBC specifications to ensure security. For example, there needs to be proof that a message sent across IBC is processed at most once by the remote chain. Also, if a message fails to reach its destination, the sender must be able to receive a timeout proof from the relayer. With CosmWasm1.0, this can be done by adding at least 6 additional entry points to the contract (note that all SNIPs currently only have 3 entry points).

Implement authenticated queries through IBC

In our example above, juno1xyz

may choose to query its gold

balance through IBC messages from its native blockchain. If authenticated queries are required, it can be done with query permits (viewing keys are insecure). Queries are not recorded on the blockchain so juno1xyz

’s balance remains private from anyone observing a block explorer. However, being on a transparent blockchain, permit queries from a remote address are not secure against snooping nodes, unless further steps are taken (see privacy considerations).

SNIP1155’s base implementation will disallow viewing key creation and queries through IBC as these are vulnerable. A transparent chain address cannot call SetViewingKey{}

or CreateViewingKey{}

without revealing their viewing keys to anyone observing the block. Note this is solvable if we encrypt messages on the transparent chain before sending, but that is out of scope in this proposal.

Modify functions

The new entry points need to integrate with the existing contract logic that handles messages and maintains a state. Functions need to be modified to handle a multi-blockchain environment. For example, additional authentication is required to ensure IBC messages are handled securely, contract functions need to be able to handle addresses from multiple blockchains, and query functions need to account for the vulnerability with viewing keys through IBC.

Privacy considerations

SNIP1155 will allow users on transparent chains to have “partial privacy” by utilizing Secret Network features. The degree of privacy depends on exactly how the dapp implements its contracts. Unsurprisingly, in most cases privacy is highest when operating fully on Secret Network.

Secret Network native level of privacy is still retained when SNIP1155 execute messages are initiated by a Secret Network address, as we saw in our illustration earlier, even if remote addresses may be involved.

If initiated by a remote address, privacy is likely to be lower. All other major IBC chains are transparent, so execute messages sent through IBC to SNIP1155 contracts will be recorded on the remote block. Information such as token balances and full tx history remain private as SNIP1155’s ledger remains on Secret Network, but if large amounts of transactions are made, an attacker can deduce information with block analysis.

Query messages aren’t saved on the block, but some vulnerabilities exist: nodes of transparent chains can see the messages that go through them, so they may be able to see query permit messages. To partially mitigate this, the dapp can design their contracts on remote chains such that they revoke their query permits after each query. By doing this, the validator may see the result of the particular IBC query, but it cannot later perform further queries on its own.

(Out of this proposal’s scope): It is also possible to strengthen the privacy in IBC messages by encrypting the messages that go through the transparent chain, such that only the intended user can decrypt it. With this, juno1xyz

can privately query its gold

balance, or send execute messages to SNIP1155 without anyone being able to view its contents despite being on a fully transparent chain. This is out of scope for now, as it is likely to require implementing a light client.

Some functions should not be called from remote chains. For example, SetViewingKey{}

and CreateViewingKey{}

are insecure if called from a transparent blockchain, as the message containing the viewing key will be recorded on the block for all to see. Unless the messages are encrypted as described in the previous paragraph, remote chain accounts should use query permits.

Item 3: Modularization

This is inspired by how cw-plus implements its reference contracts. By modularizing, the ref. implementation is separated into a “package” and a “contract”. The package will be a single source for all the core contract functionality, which is kept up to date. The contract reference is forked by dapp teams. A basic contract will be light and mostly refer to the package, and devs can then extend it with additional code without needing to fork the entire base standard. This is particularly useful for SNIP1155 target use cases, where customization is more likely. I will also aim to make it more efficient than cw-plus.

[

modularization0

1353×807 74.1 KB

](https://global.discourse-cdn.com/standard17/uploads/enigma1/original/2X/6/682bc9f825332eb8575b79fc25f9b132c8269dd7.png)

Let’s look at an example where a dapp launches today. Several months later, Secret Network upgrades, and the dapp wants to take advantage of new features from the network upgrade. The diagrams below illustrate the difference between the workflows with and without a modular reference implementation.

[

modularization1

1023×499 71 KB

](https://global.discourse-cdn.com/standard17/uploads/enigma1/original/2X/5/5a92d67087dd89a8f20837daa5e4d7aa42d06bf1.png)

[

modularization2

865×557 68 KB

](https://global.discourse-cdn.com/standard17/uploads/enigma1/original/2X/a/af130c9c58a4228f05fd7f61f7ef2442b841577e.png)

Dev teams can of course design their dapps to simplify the process of updating the contract in the first scenario, but the basic idea holds: it is likely a lot more work and error-prone without modularization.

Item 4: Secret.js integration

SNIP1155’s integration tests already contain most of the code required to create a secret.js extension. An agent-contributor has helpfully made a PR to add a SNIP1155 extension to secret.js. I can work with the contributor and Secret Labs to make any needed modifications and get the PR over the line. With this, dapps can use secret.js natively to interact with SNIP1155 tokens.

Example of how it will look like for developers calling SNIP1155 using secret.js:

const txExec = await secretjs.tx.snip1155.transfer({ sender: secretjs.address, contractAddress, msg: { transfer: { tokenId: "goldCoins", from: accounts[0].address, recipient: accounts[1].address, amount: "2_000_000" } }, }, { gasLimit: 500_000, },)

Timeline

- 2.5 months to refactor to CosmWasm1.0, add IBC and modularize contract
- secret.js integration to follow in a couple of weeks, dependent on Slab’s availability

Team

DDT (Discord: DDT#4695). I designed the SNIP1155 standard and built the reference implementation. Also created the fractionalized NFT standard reference and Scrt-Random Number Generator. A core team member of Secret University.

Funding request

USD 29,000 payable in SCRT + 10% volatility buffer