Two CryptoKitties take a honeymoon on a Plasma chain

. The blazing fast transactions heat their blood

, one thing leads to another

and a rare baby-kitten is born

. On the way home to mainnet the family gets stopped at the Plasma bridge, “

tokenIDs please!”. Oh shoot, the baby-kitten doesn’t have one on the root chain

.

How can we prevent the brutal separation of kittie families, but avoid illegal immigration to the root chain?

The following approach seems obvious:

1. A new permission on the NFT contract for bridges to register tokens minted off-chain.
2. An ID-generation scheme that doesn’t collide accross different chains.

Does a scheme exist that integrates into the Plasma exit game and enforces globally unique IDs?

Lessons from Ethereum

When an Ethereum account deploys a contract, the address is generated by hashing the account address and a strictly increasing nonce. By this account addresses are kept collision free.

This concept can be applied to the minting of NFT tokens. Each address in the minter role receives a counter and new token IDs are created by hashing the minter address with the counter.

Taking a Leap onto Plasma

Now if we had to increase the counter on the root chain every time a token got minted somewhere on a child chain we would lose the scalability advantages of Plasma.

Let’s see if we can move the counter into the token itself:

1. Each token is extended with a 32-byte data field.
2. Tokens are split into two groups, those that have a data field of 0 are workers, they can not breed. Those that carry data > 0 are queens. Queens can breed, by that creating a worker and increasing the data field by 1.

Now when a queen moves onto Plasma, the Plasma Bridge contract becomes the owner of the queen. The breeding is delegated to the bridge, while the counter can be controlled by the child chain. While the queen is on the Plasma chain, freshly bread kittens can exit from the chain and be registered with the token contract.

When the queen leaves the Plasma chain it will take the breeding rights along with it, and the counter will be updated once to the latest state.

Enforcing Rules on Child Chains

The delegation of minting rights together with a stateful attribute allows to put the queen token under the control of a Plasma contract once deposited into a chain. An example contract would be as follows:

pragma solidity ^0.5.2;

import "./IERC721.sol"; import "./IPlasmaBridge.sol";

contract BreedingCondition { address constant nftAddr = 0x1111111111111111111111111111111111111111; address constant minterAddr = 0x2222222222222222222222222222222222222222;

// spending conditions TXOs are spent if // 1. hash of script matches condHash // 2. msgData evaluates the the script to true (transfer events match outputs) function breed(uint256 _tokenId, uint256 _counter, address _receiver, uint8 _v, bytes32 _r, bytes32 _s) public { // check that buyer is authorized to hold token address signer = ecrecover(keccak256(abi.encodePacked(_tokenId, _counter)), _v, _r, _s); require(signer == minterAddr, "invalid minter signature");

```
// setup
IERC721 nft = IERC721(nftAddr);
require(nft.ownerOf(_tokenId) == address(this));
require(nft.getData(_tokenId) == _counter);
uint256 newId = uint256(keccak256(abi.encodePacked(_tokenId, _counter)));
nft.setData(_tokenId, _counter + 1);
nft.mint(_receiver, newId);
```

}

// startExit // triggers the exit of all funds to a contract on mainnet // only used on mainnet address constant bridgeAddr = 0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB;

function startExit(bytes32[] memory _proof, uint _oindex) public { if (msg.sender == minterAddr) { IPlasmaBridge bridge = IPlasmaBridge(bridgeAddr); bridge.startExit(_proof, _oindex); } } }

The contract has two functions, the breed()

function that expects a signature by the owner and mints a new token and the startExit()

function, which can only be called once the contract has been deployed on the root chain to exit the queen.

When the breed()

function is called on Plasma the UTXO set is transformed as in the figure below:

breeding as new output:

```
prevOut        input       UTXOs
                         +--------+
                         |count+1 |
                      +--+condHash|
```

+---------+ +--------+ | |tokenId | |counter | <--+prevOut | | +--------+ |condHash | |msgData +--+ |tokenId | |script | | +---------+ +--------+ +--------+ | |0x00 | +--+receiver| |newId | +--------+

The above implementation is specific to the leap network. The strict increase of the breeding counter and potential other rules could probably be enforced in a similar fashion with other Plasma designs like plasma group's predicates or Matic's contracts.

Gains

While fungible tokes have a natural performance gain when using a Plasma design (single big deposit, many small transfers on the child chain), NFTs need to be minted on the root chain and deposited one by one. With the distributed breeding function described here, NFTs could be minted on child chains and only cause root chain costs on exit.