Whitepaper: Generic Smart Debt Instruments (GSDI).

Crypto Shipwright et al., 2021, v0.2.3

Background

Current defi lending platforms rely on vaults or automated money markets. Both of these face several issues. First, new assets must be whitelisted. Second, collateralization ratios and interest rates are set by governance even if via algorithmic parameters. Third, these platforms rely on price oracles which may be vulnerable to exploitation. These issues combined open up the potential for a new type of Defi lending primitive: Generic Smart Debt Instruments (GSDI).

Borrowers transfer arbitrary assets into a smart wallet and mint a new GSDI NFT with a set price, face value, and maturity date. When the bond matures the borrower either covers the GSDI, regaining control over the smart wallet, or the lender seizes the smart wallet. This novel lending system removes systemic risk from flash crashes, allows leverage on arbitrary assets, and empowers the market to fully decide what the optimal collateral and interest rates are for each individual borrower and asset.

In addition to this paper, (1) a whitepaper on GYFI covers the GSDI Yield Farming token which will incentivize adoption and provide governance for the GSDI platform and (2) a whitepaper on GSDI Auctions provides a sample implementation of a defi application on top of GSDI.

GSDI Overview

Generic Smart Debt Instruments (GSDI) are NFT debt tokens backed by arbitrary assets held in a smart contract wallet. Since the debt is backed by a wallet rather than a particular asset, any type of current or future asset supported by smart wallets may be used to back each GSDI including ERC20, ERC721, ERC1155, tokens staked in a dapp, a smart contract wallet from a different defi platform, or any other current or future blockchain asset.

As a defi primitive, many different applications may be built on top of GSDI. Possibilities include but are not limited to:
(1) Unsecured debt from borrowers with high reputation and offchain validation of good credit;
(2) Automated leverage of liquidity farms and vaults for enhanced returns;
(3) Leverage of staked collateral for platforms that do not issue ERC20/721 tokens;
(4) Interest bearing stablecoin ERC20 tokens backed by GSDI;
(5) Project funding through sale of GSDI backed by the new protocol's revenue;
(6) Coupon bonds which pay regular interest to holders; and
(7) Default insurance which automatically compensates defaulted debt.

The GSDI Auction (GAUC) whitepaper describes in detail a sample implementation of a dapp operating on GSDI. GAUC provides an overcollateralized auctionhouse for ERC20 tokens. It has no special rights or access to the GZCB primitive contracts. More details are in the GAUC whitepaper.

GSDI NFTs traded on secondary markets should be verified that they are produced by a trusted application as "approval attacks", where the smart contract wallet approves a third party who can steal held assets, are possible with poorly designed applications. Developers should take care that the GSDI Wallet is controlled by their application on behalf of the borrower before the GSDI NFT is minted.

GSDI is deployed on 3 chains: Ethereum 1.0, xDai, and Binance Smart Chain. The contracts are identical across all chains. All contracts are upgradeable with admin and owner keys controlled by the GSDI Dao. The first byte of each GSDI is the chain ID where the GSDI was minted.

GSDI Primitives

A new GSDI is created by a borrower through the following steps:

1. The borrower creates or acquires a GSDI Wallet .
2. The borrower transfers assets to the GSDI Wallet.
4. The borrower proposes a GSDI NFT which:
    - 4a. Locks the GSDI wallet;
    - 4b. Sets the token currency in which the GSDI is denominated (such as Dai);
    - 4c. Sets the price to mint the NFT, in the selected currency;
    - 4d. Sets the face value of the GSDI, in the selected currency;
    - 4e. Sets the maturity date timestamp, in seconds; and
    - 4f. Sets the registered borrower, as an EVM address.
5. The lender mints the GSDI NFT and pays the borrower the price plus an optional 0.3% fee.
6. As the GSDI matures, there are two options:
    - 6a. The borrower covers the GSDI with currency equal to the face value. This must be done before the maturity date, or if after the maturity date before the GSDI is seized.
    - 6b. The lender seizes the GSDI Wallet and receives ownership of its assets. This can only be done if the GSDI is not covered and the current time is after the maturity date.

Note that GSDI's are defi primitives, so the above steps will always be taken through an application built on top of GSDI. Not doing so may present security risks; for instance, an IGSDIWallet that has approved tokens for transfer could have collateral removed after the sale.

GSDI is deployed on BSC, xDai, and Ethereum 1.0. GSDI token ids include a chain id prefix to allow the identification of GSDI NFTs across different chains.

GSDI Primitive Interfaces: IGSDIWallet

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol";
import "@openzeppelin/contracts/token/ERC1155/IERC1155Receiver.sol";

/// @title Smart wallet to hold generic assets.
/// @author Crypto Shipwright
interface IGSDIWallet is IERC721Receiver, IERC1155Receiver {
    /// @notice Returns the gsdiNft contract.
    /// @return gsdiNft_ Address of the IGSDINFT contract managing the wallet.
    function gsdiNft() external view returns (address gsdiNft_);

    /// @notice Returns current executor of the IGSDIWallet.
    /// @dev For GSDI dapps, guarantee that the wallet is not approved to transfer
contained locked assets.
    /// @return executor_ Current executor. May be a borrower, lender, dapp, or
IGSDINFT when locked.
    function executor() external view returns (address executor_);

    /// @notice New GSDI wallets are deployed as Open Zeppelin proxies. Initialize
is called on creation.
    /// @param _gsdiNft of the IGSDINFT contract managing the wallet.
    /// @param _executor Current executor. May be a borrower, lender, dapp, or
IGSDINFT when locked.
    function initialize(address _gsdiNft, address _executor) external;
```

```
    /// @notice Sets the current executor. May only be called by executor or
IGSDINFT.
    /// @param _executor New executor. IGSDINFT sets to itself to lock the wallet.
    function setExecutor(address _executor) external;

    /// @notice Executes an arbitrary transaction. May only be called by executor.
    /// @dev `execute` will not revert if the call reverts. For automatic revert,
use `safeExecute`.
    /// @param _to Address to call.
    /// @param _value Ether to send for call.
    /// @param _data ABI encoded with signature transaction data to send in the
call
    /// @return data_ Data returned by the call.
    /// @return success_ Whether call completed successfully.
    function execute(
        address _to,
        uint256 _value,
        bytes memory _data
    ) external returns (bytes memory data_, bool success_);

    /// @notice Executes an arbitrary transaction. May only be called by executor.
Reverts on failure.
    /// @param _to Address to call.
    /// @param _value Ether to send for call.
    /// @param _data ABI encoded with signature transaction data to send in the
call
    /// @return data_ Data returned by the call.
    function safeExecute(
        address _to,
        uint256 _value,
        bytes memory _data
    ) external returns (bytes memory data_);

    /// @notice Tranfers arbitrary IERC20 tokens from the wallet and reverts on
failure. May only be called by executor.
    /// @param _token Address of IERC20 token supporting transfer.
    /// @param _to Address to receive the tokens.
    /// @param _value Quantity of tokens to send.
    function safeTransferIERC20(
        address _token,
        address _to,
        uint256 _value
    ) external;

    /// @notice Transfers arbitrary IERC721 tokens from the wallet and reverts on
failure. May only be called by executor.
    /// @param _token Address of IERC721 token supporting transfer.
    /// @param _to Address to receive the tokens.
    /// @param _ids IDs of tokens to send.
    function safeTransferIERC721(
        address _token,
        address _to,
        uint256[] calldata _ids
    ) external;

    /// @notice Transfers arbitrary IERC1155 tokens from the wallet and reverts on
failure. May only be called by executor.
    /// @param _token Address of IERC1155 token supporting transfer.
```

```
    /// @param _to Address to receive the tokens.
    /// @param _ids IDs of tokens to send.
    /// @param _values Number of each token to send.
    function safeTransferIERC1155(
        address _token,
        address _to,
        uint256[] calldata _ids,
        uint256[] calldata _values
    ) external;
}
```

GSDI Primitive Interfaces: IGSDINFT

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/extensions/IERC721Enumerable.sol";
import "../interfaces/IGSDIWallet.sol";

/// @title Generic Smart Debt Instrument NFTs for lending against generic assets
including vaults.
/// @author Crypto Shipwright
interface IGSDINFT is IERC721Enumerable {
    /// @return chainId_ ChainID on which the contract is deployed.
    function chainId() external view returns (uint96 chainId_);

    /// @return governance_ Address which sets governance parameters.
    function governance() external view returns (address governance_);

    /// @return isFeeEnabled_ Whether the 0.3% fee is enabled.
    function isFeeEnabled() external view returns (address isFeeEnabled_);

    /// @notice Sets the treasury address to which the fee must be sent.
    /// @return treasury_ Address to receive the 0.3% fee.
    function treasury() external view returns (address treasury_);

    /// @param _id GSDI ID to view the chain ID for.
    /// @return gsdiChainId_ ChainID for the GSDI. Leftmost 12 bytes of the GSDI
id.
    function gsdiChainId(uint256 _id)
        external
        view
        returns (uint8 gsdiChainId_);

    /// @notice Returns the full onchain metadata for a GSDI or GSDI Proposal.
Reverts if ID is from a different chain.
    /// @param _id ID of the GSDI. First byte is the chainID of the GSDI.
    function metadata(uint256 _id)
        external
        view
        returns (
            uint256 maturity_,
            uint256 faceValue_,
            uint256 price_,
            IGSDIWallet wallet_,
            address currency_,
            address borrower_,
```

```
            bool isInProposal
        );

    /// @notice Changes the current borrower which will receive the GSDI after it is covered. Reverts if sender is not borrower.
    /// @param _borrower New address to set the borrower to.
    function transferBorrower(address _borrower) external;

    /// @notice Sets whether the fee is enabled. Only callable by governance.
    /// @param _isFeeEnabled Whether to enable the 0.3% fee.
    function setIsFeeEnabled(bool _isFeeEnabled) external;

    /// @notice Sets the governance treasury. Only callable by governance.
    /// @param _treasury New address for the treasury.
    function setTreasury(address _treasury) external;

    /// @notice Sets the governance address. Only callable by governance.
    /// @param _governance New address for governance.
    function setGovernance(address _governance) external;

    /// @notice Mints a new GSDI in proposal to IGSDINFT. Locks the IGSDIWallet by setting IGSDINFT as the wallet's executor. Only callable by the current IGSDIWallet executor.
    /// @param _maturity Timestamp when the GSDI matures and the lender may seize the wallet.
    /// @param _faceValue Amount of currency borrower must pay to cover the GSDI.
    /// @param _price Amount of currency the lender must pay the borrower to mint the GSDI.
    /// @param _wallet Wallet containing collateral backing the GSDI.
    /// @param _currency Token currency for the price and face value.
    /// @param _borrower Address which will become the wallet executor after the GSDI is covered.
    function propose(
        uint256 _maturity,
        uint256 _faceValue,
        uint256 _price,
        IGSDIWallet _wallet,
        address _currency,
        address _borrower
    ) external;

    ///@notice Cancels the GSDI proposal. Sender must be borrower. GSDI must currently be in proposal. GSDI must be on the current chain. Burns the GSDI.
    /// @param _id GSDI to cancel.
    function cancel(uint256 _id) external;

    /// @notice Sends the GSDI to sender. The IGSDIWallet must be in proposal. GSDI must be on the current chain. Transfers price in currency from sender to borrower. Removes GSDI from proposal.
    /// @notice If governance has enabled the 0.3% fee, the sender must also transfer the fee to the governance's treasury.
    /// @dev Sender must approve the contract to transfer price of currency before call.
    /// @param _id GSDI to purchase.
    function purchase(uint256 _id) external;

    /// @notice Sets the borrower for IGSDIWallet to executor. The IGSDIWallet must not be in proposal. GSDI must be on the current chain. Sender transfers face value in currency to current GSDI holder. Burns the GSDI.
```

```
    /// @dev Sender must approve the contract to transfer face value of currency
before call.
    /// @param _id GSDI to cover.
    function cover(uint256 _id) external;

    /// @notice Sets the lender for IGSDIWallet to executor. The GSDI must be held
by sender. GSDI must be on the current chain. Must be after maturity. Burns the
GSDI.
    /// @param _id GSDI to seize.
    function seize(uint256 _id) external;
}
```