ICHI Network
oneToken Factory
oneSUSHI Implementation

Statement of Intended Behavior

Purpose

oneSUSHI is a collateralized ERC20 token designed to maintain a peg to one USD. oneSUSHI is issued in return for a blend of other stablecoins and xSUSHI (ERC20 representing a share of staked SUSHI) at a specific ratio algorithmically adjusted over time.

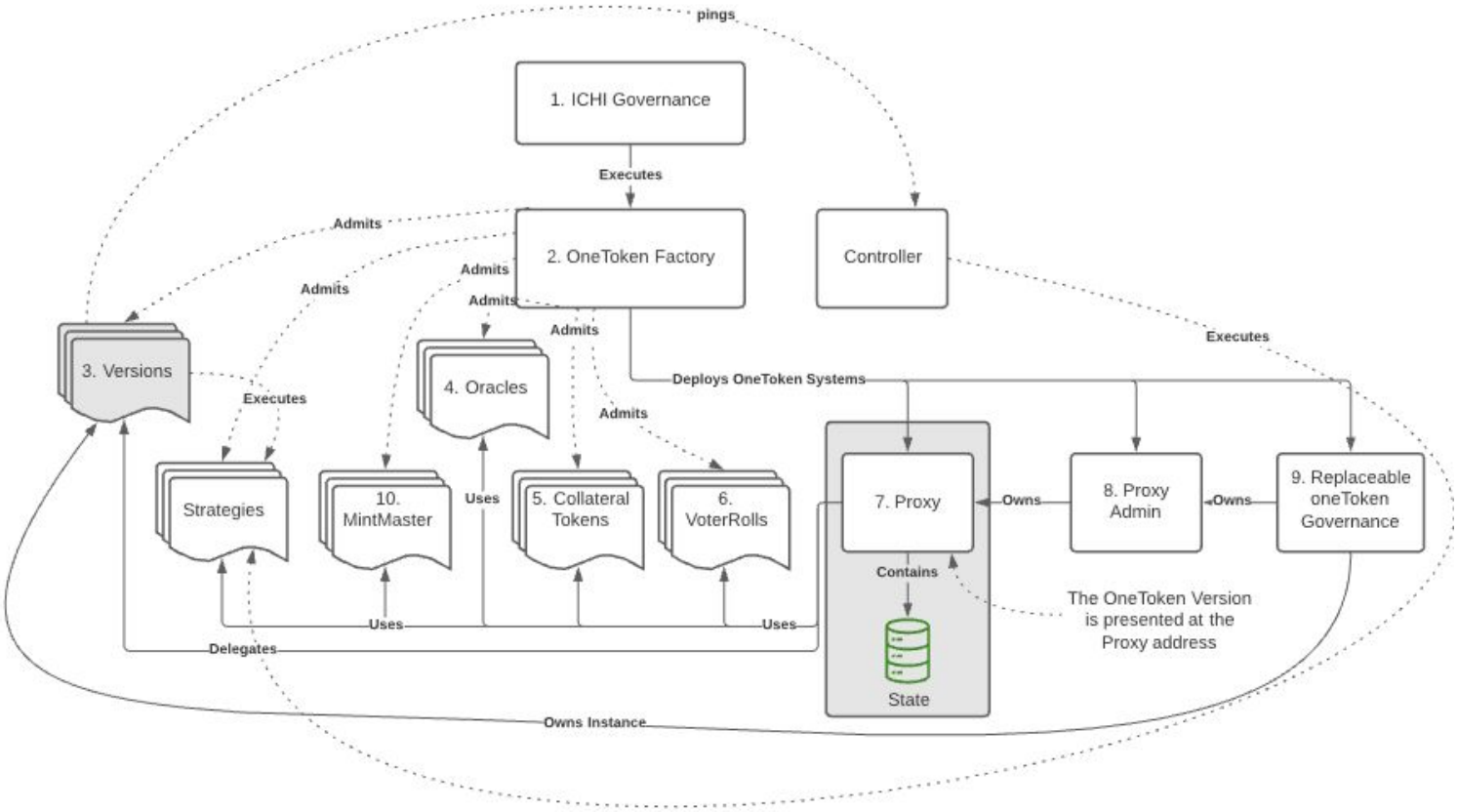
oneSUSHI is the first instance of a more generalized design that produces flexible and upgradeable oneTokens with distinct governance and internal logic within boundaries defined at the global, ICHI governance level.

Architecture

oneToken system:	oneSUSHI is an instance of modular contracts where oneTokenGovernance can replace modules or upgrade the core contract using the Proxy pattern.
oneTokenFactory:	oneSUSHI will be the first of many contract systems deployed by a contract factory. Global (ICHI) state information is stored and managed in this system-wide hub.
ICHIGoverance:	Global governance controls the oneTokenFactory and manages global parameters. Governance itself is external to the system.
oneTokenGovernance:	oneTokens, including oneSUSHI, are managed by distinct governance. Governance itself is external to the system.
oneTokenVault:	The core upgradeable (Proxy) contract holds assets, mints and burns ERC20 tokens, and controls the process flow during the minting and redemption process, including coordinating modularized components.
Versions:	oneTokenVault implementations (Proxy pattern) are admitted to the network by ICHIGovernance. oneTokenGovernance can optionally

	adopt (upgrade, downgrade) an approved version of the oneTokenVault.
Oracles:	Price feed modules report the value of assets in the oneTokenVault through a normalized interface. The oneTokenVault passes Oracle price conclusions and volatility signals to MintMaster modules.
CollateralTokens:	oneTokenVaults (oneSUSHI) accept 0-n stablecoins in tandem with Member Tokens (SUSHI) at a specific ratio, the Minting ratio. Upon redemption, oneTokenVaults burn oneTokens and in return for stablecoins of equal value. Since more than one kind of stablecoin exists in the oneTokenVault at any given time, species options are always offered on a “while supplies last” basis. Sufficient value always exists.
MemberTokens:	oneTokenVaults accept another token, usually the namesake, e.g., oneSUSHI receives xSUSHI and CollateralTokens in a specific MintingRatio and emits a oneSUSHI token for every $1 of value.
Minting Ratio:	The required ratio of CollateralTokens strongly pegged to 1 USD and MemberTokens that may be illiquid and volatile that is exchanged for oneTokens of equal value. For example, a MintingRatio of 90% means that stablecoin value of 90% and MemberToken value of 10% can be traded for an equal value of oneTokens (100%). Oracles report USD exchange rates. Equal value means equal value in USD.
VoterRoll:	ICHIGovernance and oneTokenGovernance rely on voting power based on tokens staked in external contracts. VoterRoll contracts report the computed voting power of any given user. Any formulation is technically permissible. ICHIGovernance and each oneTokenGovernance may appoint successor governance schemes from VoterRoll implementations admitted into the system by ICHIGovernance.
MintMaster:	The MintingRatio adjusts over time to maintain the exchange rate peg of oneToken (oneSUSHI) to USD 1. A MintMaster uses any available information to compute the current minting ratio. The interface includes a maximum transaction volume figure (can be infinite) to accommodate anticipated future logic requirements. Since the rate can be up to 100% and the maximum volume can be as low as zero, the MintMaster can be a circuit-breaker if so desired. Oracles report volatility. MintMasters decide what to do.

Treasury Management: oneTokenGovernance may invest CollateralTokens and MemberTokens in the oneTokenVault by sending arbitrary instructions, similar to a multi-signature wallet.



Interface Extensions, Configurability

The design includes interfaces and hooks to establish a framework to accommodate the expected evolution of functionality. The aim is to reduce the frequency of Version upgrades at the oneTokenVault level by defining flow-control and generalized interfaces that anticipate future requirements while remaining agnostic about implementation details.

Modules use the oneTokenFactory as a storage repository for parameters that persist when modules are replaced. The design is agnostic about the internal logic and parameters employed by modules. The structure defines flow control, generalized storage of configuration parameters, and interfaces modules must implement, including anticipated modules in the future.

Automated Strategies

The current implementation of oneToken for oneSUSHI entrusts the oneTokenGovernance to manage, invest, rebalance, stake, and withdraw Collateral, MemberTokens, and other tokens (e.g., Liquidity Pool token) that are held by the oneTokenVault. This falls short of an ideal arrangement in that there are no constraints on that authority, no protection from human error, and no automation.

The design anticipates the introduction of automated strategies that attend to treasury management concerns and automate ongoing processes.

oneToken Strategies: These contracts would implement processes such as rebalancing and investing. Each would be admitted by ICHIGoverance and chosen by oneTokenGovernance.

ICHIController: This global contract would implement system-wide invariants such as enforcing (double-checking) conservation of value or allowable fees/slippage rules. The Controller would also be responsible for scheduling/periodicity of automated processes.

In this arrangement, the ICHIController executes the appointed strategy for a given oneTokenVault at the specified periodicity.

The oneTokenVault pings the ICHIController from the deposit and withdrawal functions. In this implementation, the ICHIController stub simply returns because no Strategy is defined, and there is nothing to do.

The expected evolution includes deploying Strategies that automate investment activity and well-solved processes such as rebalancing collateral, staking it, and working with acquired liquidity tokens. A whitelist of allowable activities will replace the initial “governance knows best” approach.

The initial oneSUSHI implementation employs a oneTokenVault that pings the Controller for flow-control purposes, a replaceable Controller that does nothing, and a Strategy that does anything the oneTokenGovernance asks it to do when instructed to do so directly by oneTokenGovernance.

Oracles

The oneTokenVault passes the Oracles’ quotes and volatility metrics to the MintMaster, whose job is to return a MintingRatio and MaximumOrderVolume. These modules’ initial implementations return “no volatility” and “infinite order volume” to passively permit operations within the oneTokenVault’s flow-control and validation logic.

A pegged stablecoin Oracle returns a hard-coded value of USD 1 without consulting external sources. This Oracle is suitable for liquid stablecoins with strong pegs and saves gas by avoiding unnecessarily external calls.

oneTokenVaults share Oracle modules. oneTokenGovernance chooses Oracles based on preference for price feed source, interpretation, and volatility-detection logic. oneTokenGovernance supplies configuration and control parameters determined by the Oracle designer.

An Oracle would not be structured this way because this approach would result in many near-duplicate implementations. :

> “The Ethereum Price Oracle used by oneSUSHI.”

Oracles are structured in reusable fashion:

> “A time-weighted average price Oracle using a specified Sushiswap Liquidity Pool, with Volatility-detection based on configurable fast and slow-moving averages and configurable standard deviations.”

Within this structure, the Oracle for very stable stablecoins and optimized for gas efficiency would be described as::

> “A trivial Oracle that unconditionally returns 1 USD and zero volatility..”

oneTokenGovernance determines which Oracle to use for each token in the oneTokenVault and sets the values of parameters the Oracle needs. As used by a given oneToken implementation, Oracle configuration parameters are stored in the oneTokenVault and are available to successor Oracle implementation.