EXHIBIT A

US010025797B1

(12) **United States Patent**
Fonss

(10) **Patent No.: US 10,025,797 B1**
(45) **Date of Patent: Jul. 17, 2018**

(54) **METHOD AND SYSTEM FOR SEPARATING STORAGE AND PROCESS OF A COMPUTERIZED LEDGER FOR IMPROVED FUNCTION**

(71) Applicant: **True Return Systems LLC**, New Canaan, CT (US)

(72) Inventor: **Jack Fonss**, New Canaan, CT (US)

(73) Assignee: **True Return Systems LLC**, New Canaan, CT (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **15/923,317**

(22) Filed: **Mar. 16, 2018**

Related U.S. Application Data

(60) Provisional application No. 62/634,321, filed on Feb. 23, 2018.

(51) **Int. Cl.**
G06F 17/30 (2006.01)

(52) **U.S. Cl.**
CPC .. ***G06F 17/30194*** (2013.01); ***G06F 17/30227*** (2013.01)

(58) **Field of Classification Search**
CPC G06F 17/30227; G06F 17/30194
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2012/0158558	A1*	6/2012	Hahn-Carlson	G06Q 10/10 705/30
2012/0203645	A1*	8/2012	Sutter	G06Q 30/04 705/19
2015/0244804	A1*	8/2015	Warfield	H04L 47/6295 709/219
2017/0046792	A1*	2/2017	Haldenby	G06Q 20/0655
2018/0095662	A1*	4/2018	Brennan	G06F 3/061
2018/0115428	A1*	4/2018	Lysenko	H04L 9/3247

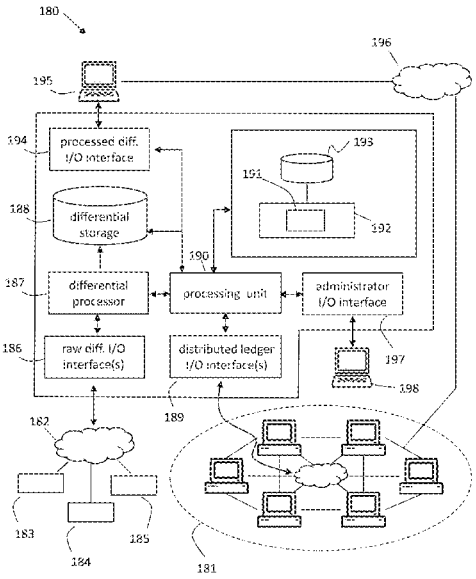
* cited by examiner

Primary Examiner — Leslie Wong
(74) *Attorney, Agent, or Firm* — Lerner, David, Littenberg, Krumholz & Mentlik, LLP

(57) **ABSTRACT**

A non-conventional method and system used with computerized ledgers provides advantages of computing efficiencies, data security, and universal use. The system, method, and computer readable storage medium for storing, creating, monitoring, managing, and modifying measurement, descriptive differences, and parameters of the records of distributed computerized ledgers works through a separation and linkage of stacked modular data storage and processing. Electronic transaction records reside on distributed ledgers and modifying measurement and descriptive differences reside in decentralized or centralized storage, where computers and related networks are improved with increased functionality through increased transaction speeds, decreased data transmissions, increased security, and improved modifiable functionality. The separation of parallel layered storage and modularity of design enable the system to perform a wide range of functionality while maintaining homogeneity with the distributed computerized ledger.

20 Claims, 9 Drawing Sheets



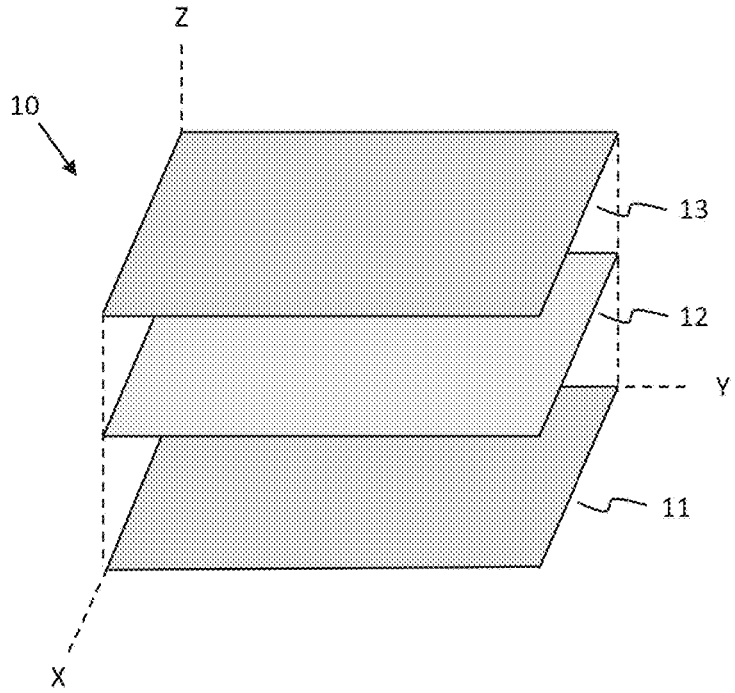

Z
10
13
12
Y
11
X

FIG. 1

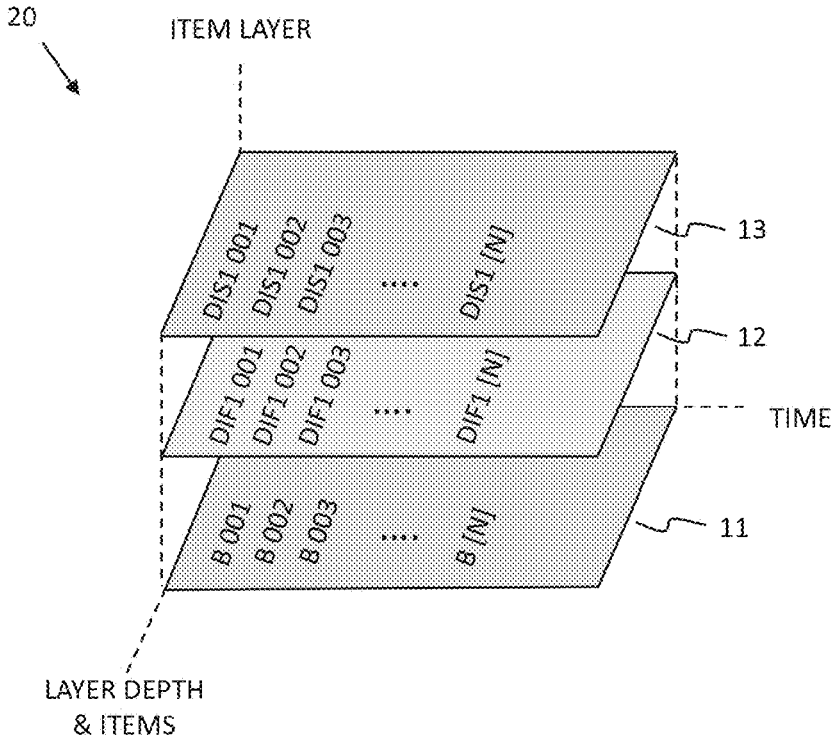

20
ITEM LAYER
DIS1 001
DIS1 002
DIS1 003
DIS1 [N]
13
DIF1 001
DIF1 002
DIF1 003
DIF1 [N]
12
TIME
B 001
B 002
B 003
B [N]
11
LAYER DEPTH
& ITEMS

FIG. 2

FIG. 3

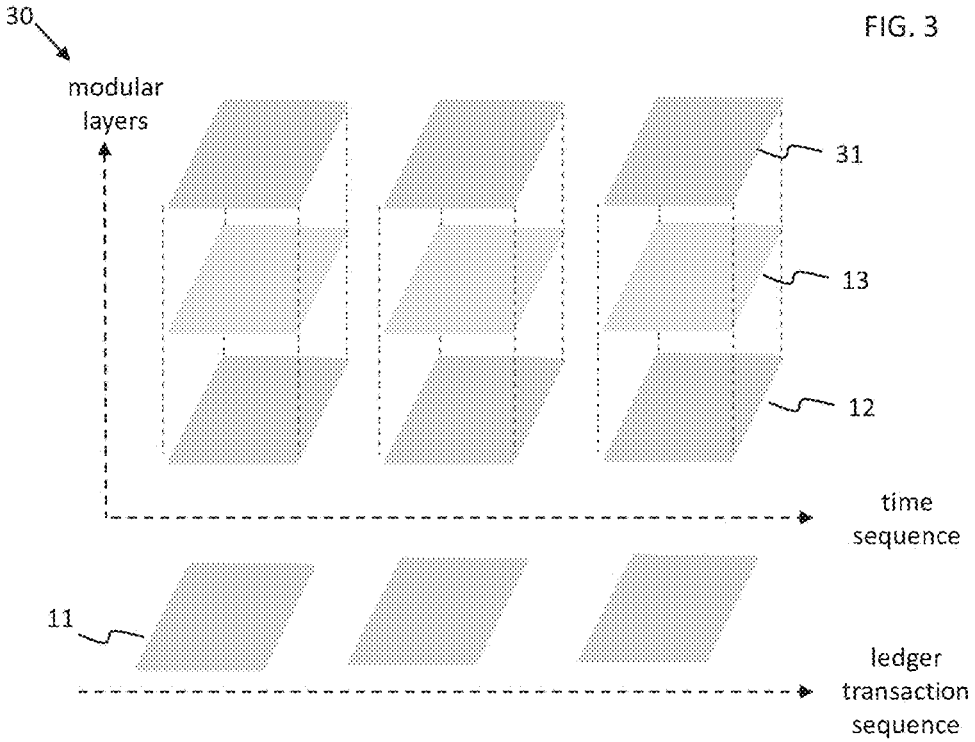
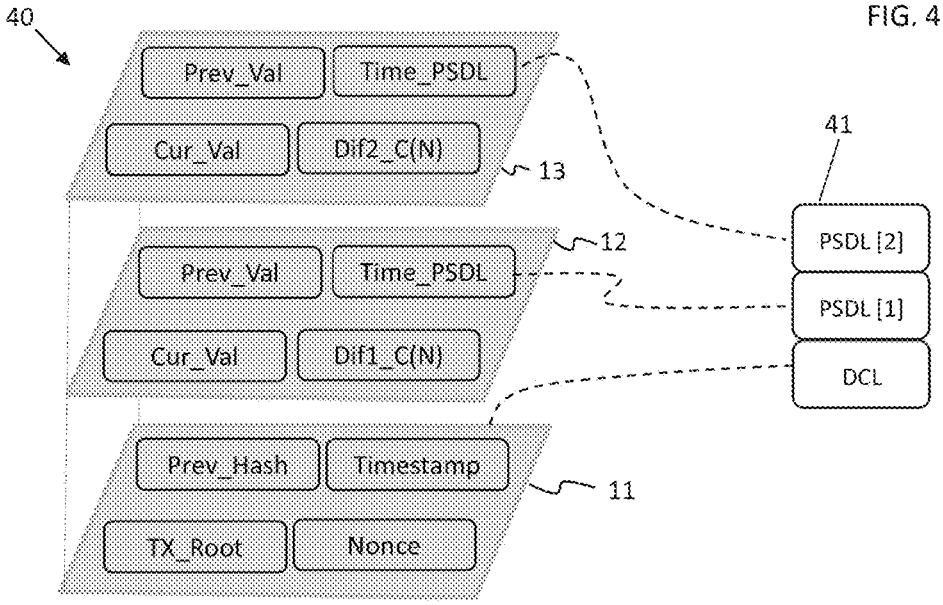

30
modular layers
31
13
12
time sequence
11
ledger transaction sequence

FIG. 4


40
Prev_Val
Time_PSDL
Cur_Val
Dif2_C(N)
13
41
12
Prev_Val
Time_PSDL
Cur_Val
Dif1_C(N)
PSDL [2]
PSDL [1]
DCL
Prev_Hash
Timestamp
11
TX_Root
Nonce

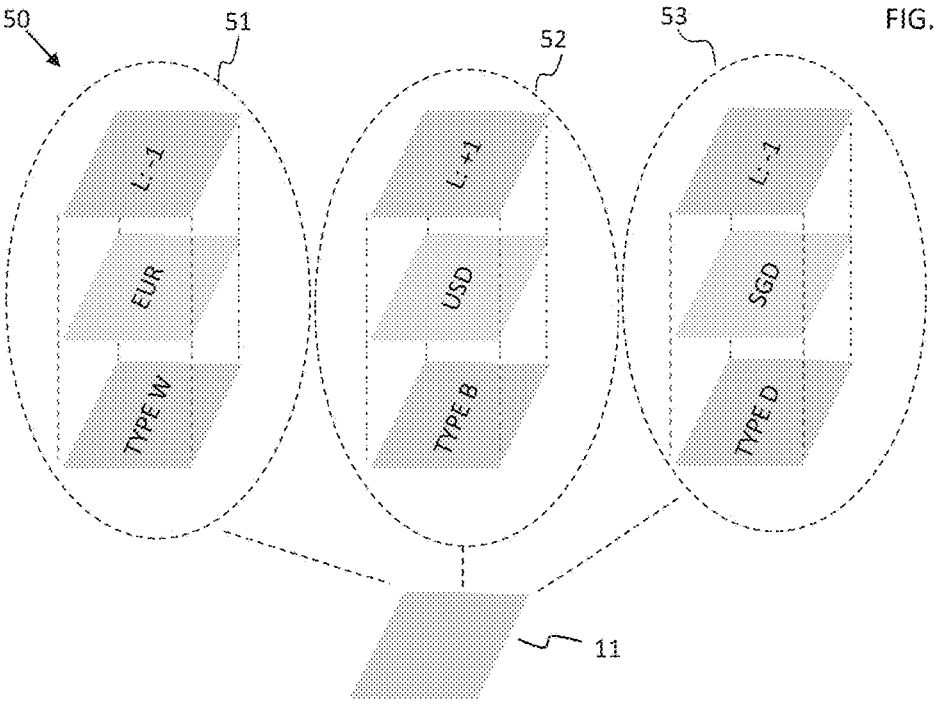

50
51
52
53
L: -1
EUR
TYPE W
L: +1
USD
TYPE B
L: -1
SGD
TYPE D
11

FIG. 5

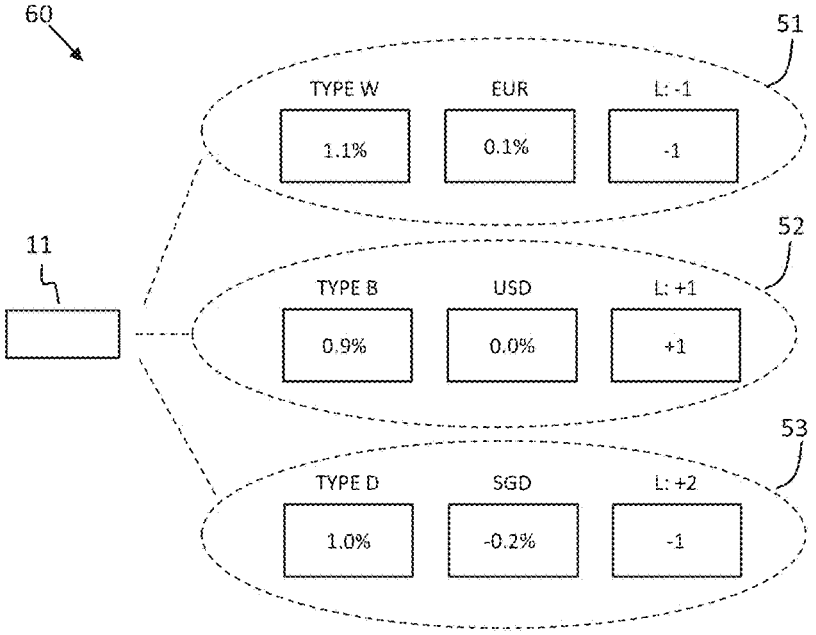

60
11
51
TYPE W
EUR
L: -1
1.1%
0.1%
-1
52
TYPE B
USD
L: +1
0.9%
0.0%
+1
53
TYPE D
SGD
L: +2
1.0%
-0.2%
-1

FIG. 6

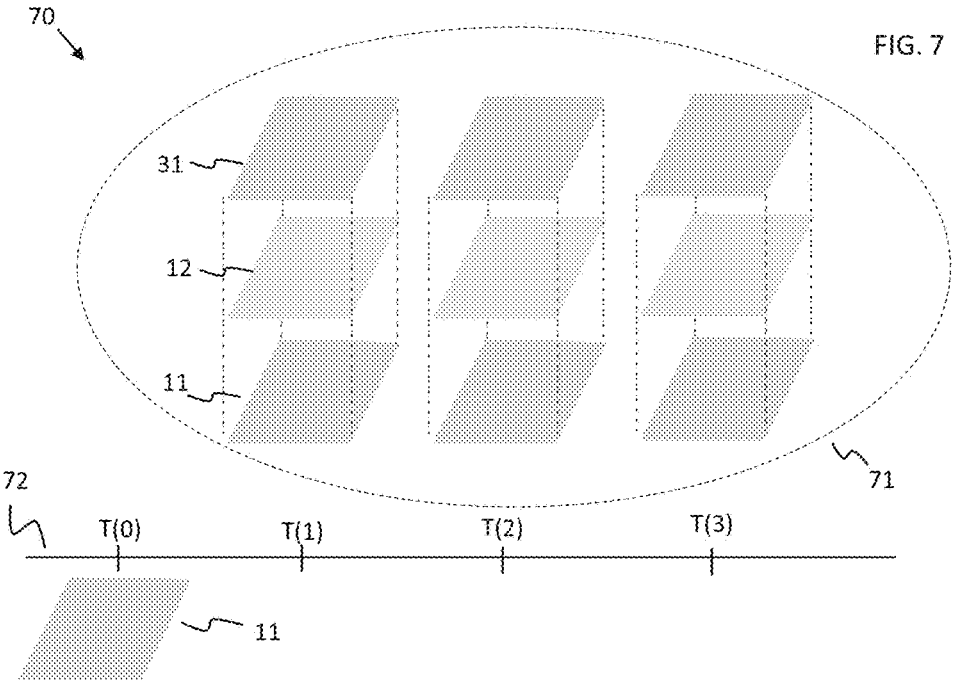


FIG. 7

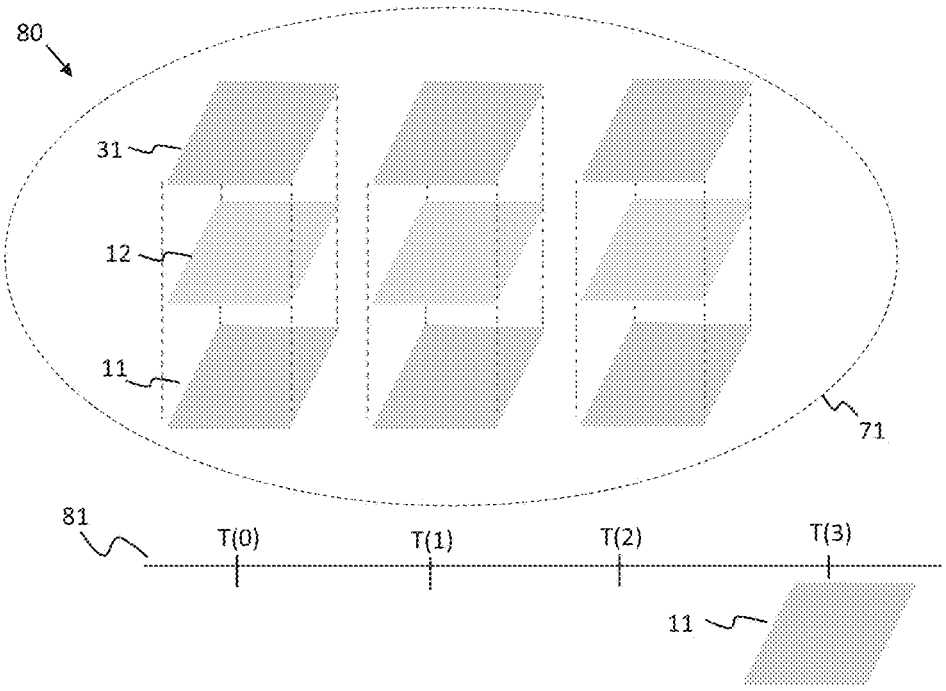


FIG. 8

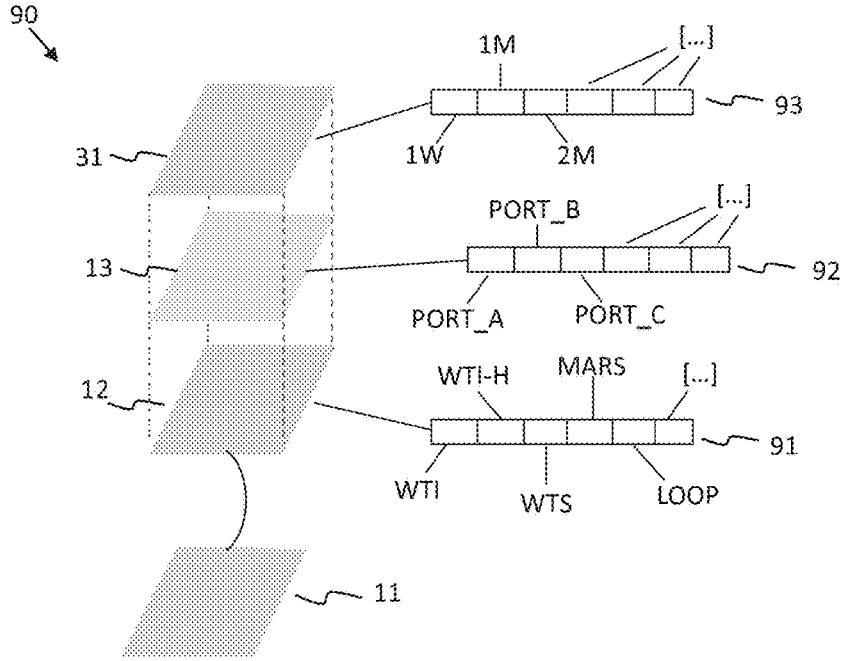

90
1M
[...]
93
1W
2M
31
PORT_B
[...]
13
92
PORT_A
PORT_C
WTI-H
MARS
[...]
12
91
WTI
WTS
LOOP
11

FIG. 9

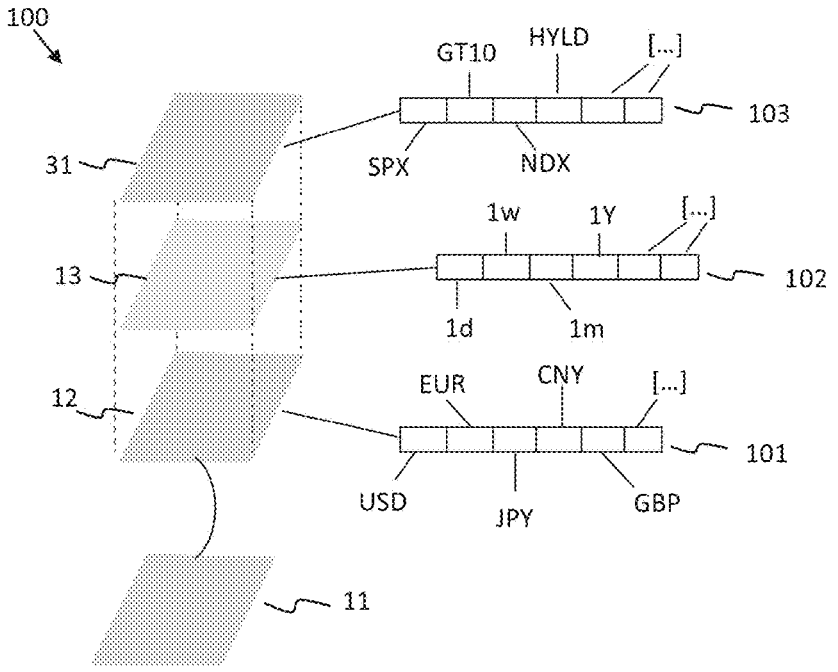
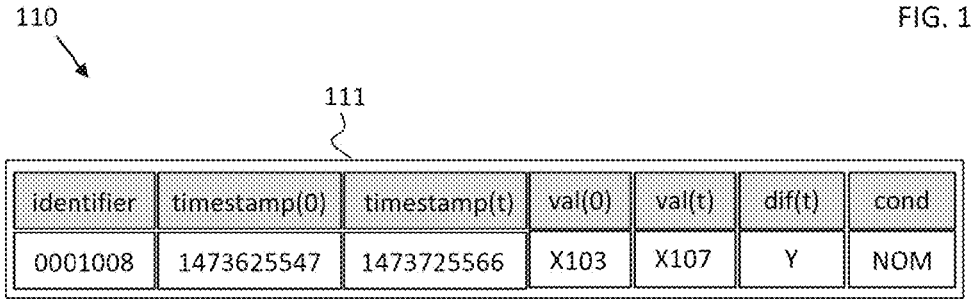

100
GT10
HYLD
[...]
103
31
SPX
NDX
1w
1Y
[...]
13
102
1d
1m
EUR
CNY
[...]
12
101
USD
JPY
GBP
11

FIG. 10

FIG. 11



identifier	timestamp(0)	timestamp(t)	val(0)	val(t)	dif(t)	cond
0001008	1473625547	1473725566	X103	X107	Y	NOM

FIG. 12

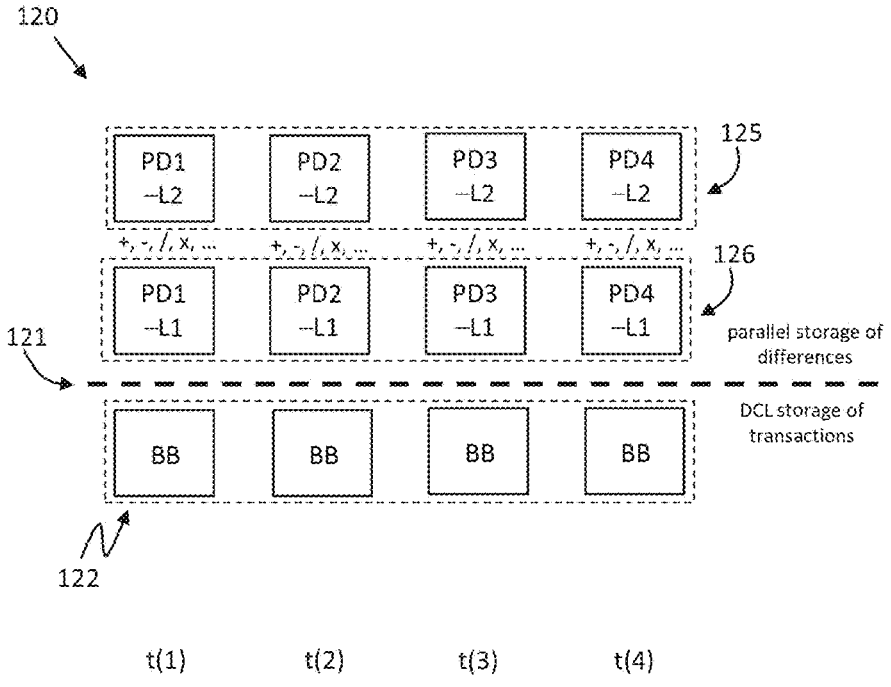


FIG. 13

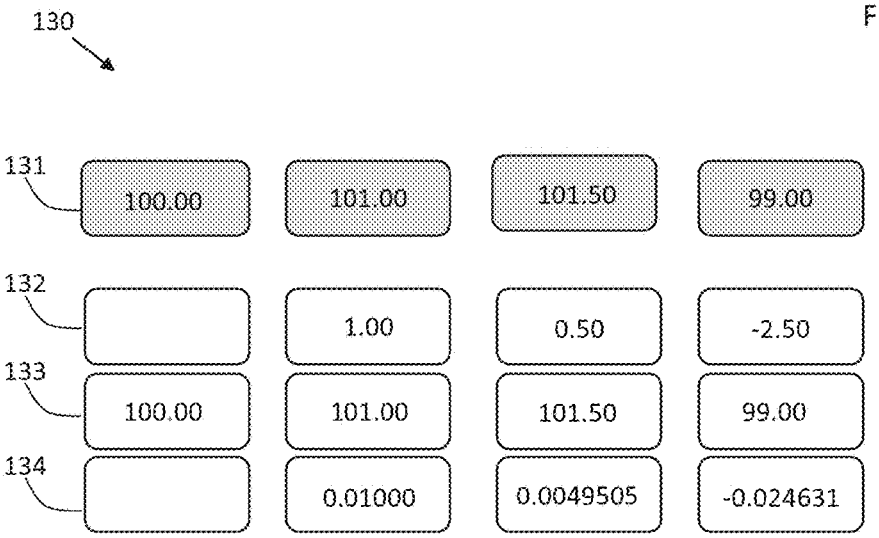


FIG. 14

140

VAL	DIF1	DIF2
USD	USD	10000
EUR	EUR	01000
JPY	JPY	00100
CNY	CNY	00010
GBP	GBP	00001

FIG. 15

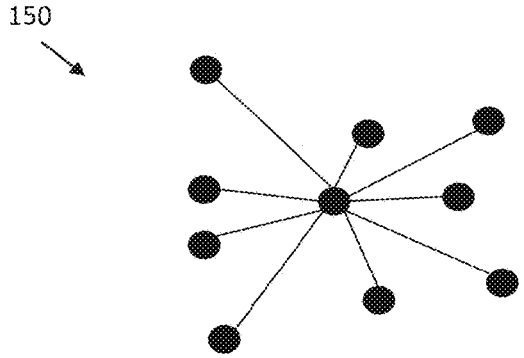


FIG. 16

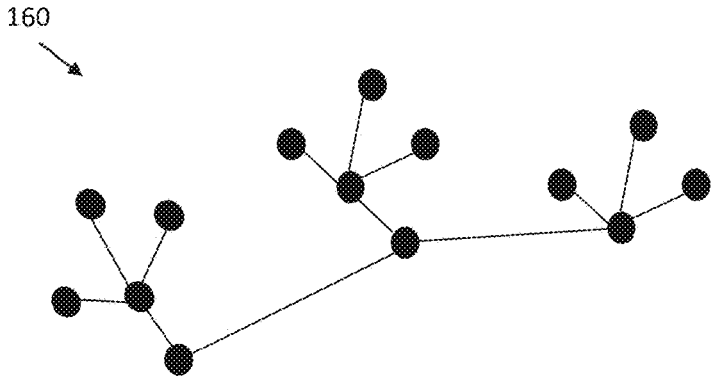


FIG. 17

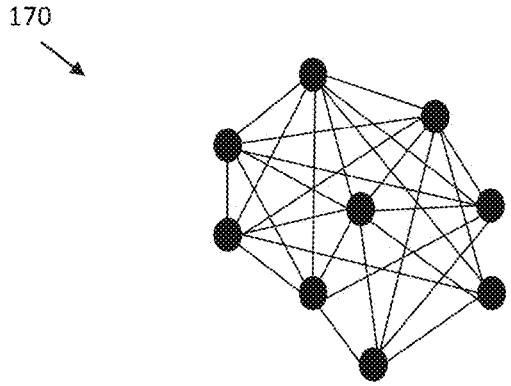
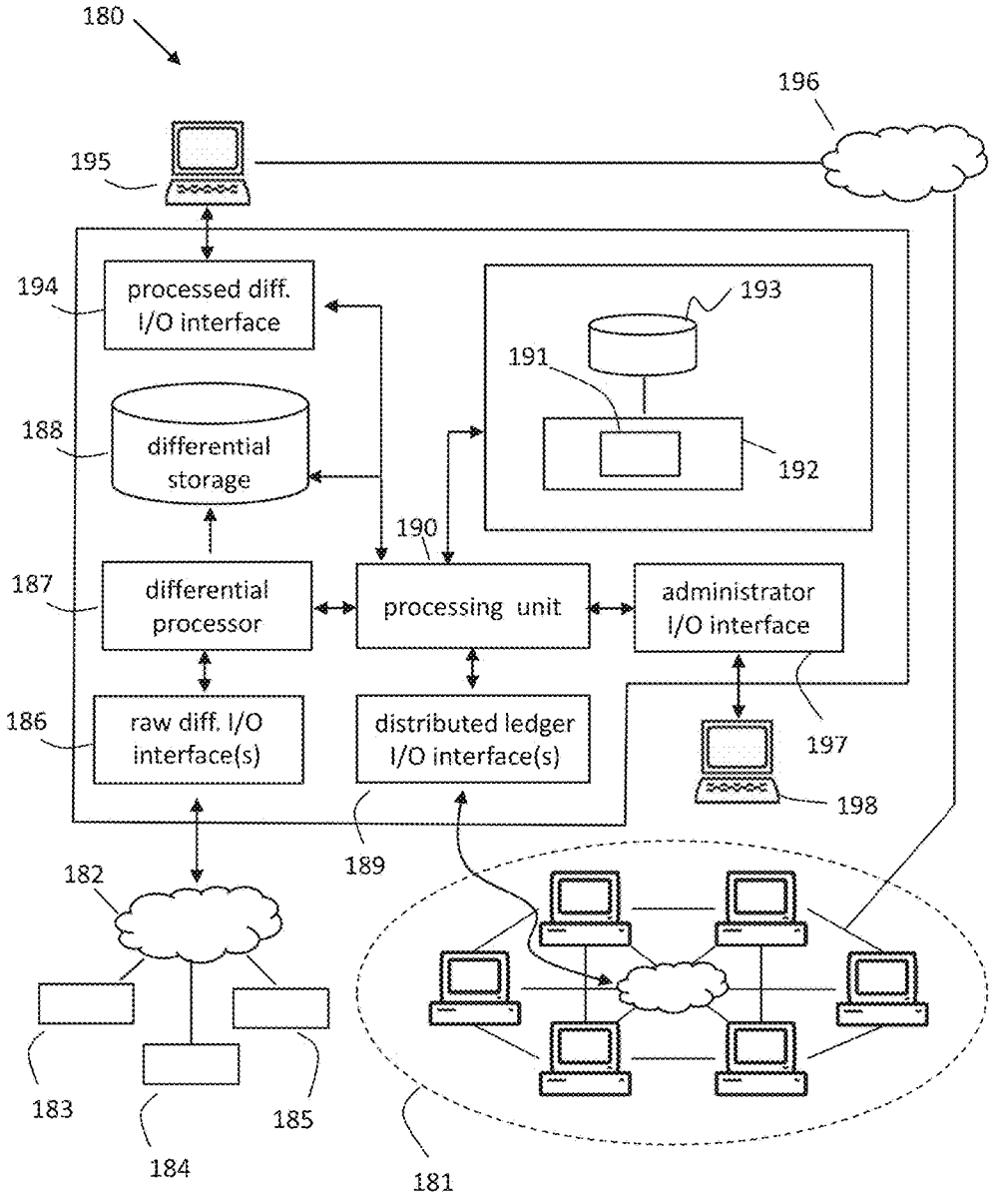


FIG. 18


180
196
195
194
processed diff. I/O interface
193
191
188
differential storage
192
190
187
differential processor
processing unit
administrator I/O interface
186
raw diff. I/O interface(s)
distributed ledger I/O interface(s)
197
198
189
182
183
184
185
181

METHOD AND SYSTEM FOR SEPARATING STORAGE AND PROCESS OF A COMPUTERIZED LEDGER FOR IMPROVED FUNCTION

CROSS-REFERENCE TO RELATED APPLICATION

The present application claims the benefit of the filing date of U.S. Provisional Patent Application No. 62/634,321, filed Feb. 23, 2018, the disclosure of which is hereby incorporated herein by reference.

FIELD OF USE

A novel computer system, network connectivity, and data storage software architecture improves the efficiency, utility and security of a computerized ledger. The disclosed system, method, and computer readable storage medium improve and may be utilized with a wide range of computerized ledgers, including distributed ledgers, decentralized ledgers, and centralized ledgers, where computerized ledgers store and report encrypted, or otherwise secured, electronic transactions, and provide a universal solution to improve the efficiency, utility, and security of computerized ledgers.

BACKGROUND

Generally, computerized ledgers are databases operated on one or more servers by a specialized computer, or operated on a specialized network and controlled by separate computers. A computerized ledger records encrypted or otherwise secured records of transactions, and a computerized ledger can be, among other things, centralized, decentralized, or distributed. Briefly a centralized computerized ledger system is where all nodes connect to a central hub. The management and modifications to the computerized ledger in a centralized environment are generally performed by a centralized computer system and there is usually only one official (or consensus) copy of the computerized ledger. A distributed computerized ledger (DCL) system is where all nodes are independently connected to each other, and the management and modifications to the computerized ledger in a distributed environment are generally performed by separate computers and each computer usually stores its own official copy of the computerized ledger which is proofed for accuracy by a consensus system running on the decentralized network.

The use of distributed computerized ledgers is gaining acceptance and popularity in a number of industrial uses including health care, international trade, and electronic (or crypto) currencies. Distributed ledgers are believed to have a number of advantages over other storage and transaction recording systems. Among the advantages of distributed ledgers are the ability to perform simultaneous updates across multiple fully independent nodes, decreased risk of data loss and corruption through widely distributed consensus-proofed copies, and the ability to create peer-to-peer environments where network validated transactions can be executed with or without a central intermediary. In theory, removing central intermediaries and more directly connecting counterparties through an instantaneous updating and tamper-proof ledger has the promises of improved speed, transparency, and efficiency in related computer systems and networks.

Through supporting systems and internet connectivity, computerized ledgers typically write, encrypt, store, access,

and transmit stored and modified records to specialized computers or specialized networks of computers. DCLs are expected to deliver a number of benefits over alternative storage and access systems including, high levels of security, immutability of transaction records, automated integrity processing, and concurrent read/write capability across multiple nodes. While implementations are currently limited, industry forecasters continue to expect DCLs to store and process transactions relating to commercial goods, health records, tangible property, financial instruments, and other items.

Developers of DCL technology face a number of competing tradeoffs and challenges in function and practical implementation. For example, some of these competing tradeoffs and challenges include secrecy of data, privacy of transactions, speed of recording transactions, speed in updating records, speed in storage and transmission, and full security of the transactions record trail. Typically DCLs engage in redundant movement of transaction data on a peer-to-peer basis such that there is independent processing at every relevant node to facilitate different forms of consensus control and audit, often without the services of a central administrator.

One of the most common data structure formats for DCLs is a block format, in which transactions are aggregated and processed within distinct computer timestamp measured periods of time, and where each aggregation of properly authenticated transactions is written to the DCL in the form of an appended block or comparable structure.

Where a DCL relies on multimode consensus, audit trails and sequencing control may include a range of cryptographic techniques including so-called mining processes based on cryptography or processing power (also known as "proof of work") or proof of stake processes based on holders and holdings within the records providing some validation. Even in some of the least data rich DCLs, such as the block chain implementations of cryptocurrencies (including Bitcoin, Ethereum and the like), the computational burden of basic transactional data in DCLs is slowing networks and jeopardizing recordkeeping, accuracy and potential growth. Cryptocurrencies typically contain only the data necessary to maintain transaction records; as industry attempts to expand the types of DCL applications, higher data requirements are certain to further frustrate processing and transmission speeds.

Most decentralized electronic ledgers (including those used for electronic currencies) are limited in functionality in that their representational blocks are homogenous and their use of timestamped sequencing is limited to curing the "double spend" problem; that is, the transacting of a ledger item which has already been transacted. The most promising known solutions to higher functionality involve pushing more data or computer code through already limited blocked data arrangements.

The promise of DCLs is big, but the industry is still challenged by the barebones data requirements of cryptocurrencies; using known techniques including colored coins and smart contracts to put real estate, health records, commercial transactions, and financial instruments on DCLs is likely to exacerbate current speed and block size challenges. The addition of smart contracts is already introducing serious security concerns.

Expanded implementation of DCLs, for example beyond homogeneous block cryptocurrencies, has been slower than many professionals in computer science, government, and commerce had anticipated. The simplified homogeneous blocks of electronic currencies are already proving difficult

to transact, transmit, and secure; news reports regularly cite problems including delays in validations and settlements, and excessive transaction costs. Proposed extensions of DCLs are generally directed at techniques such as colored coins and smart contracts, however these types of implementations also have many drawbacks including they will: (i) demand continuously revised and customized systems, (ii) add additional pressure to networks and computer systems relating to processing, storage, and transmission, and (iii) introduce vulnerabilities where operative code or descriptors is openly accessible or widely distributed.

Data heavy DCLs (including colored coins and smart contracts) will have a number of drawbacks including: (i) the need for purpose built architecture, operations, and interfaces for new applications and implementations, (ii) the need to coordinate the storage of application specific data and the operations of that data with all possible contributors and users, and (iii) the technological limitations relating to increasing file and block sizes which hampers processing efficiency, decreases practical applications for many users, and severely limits a universal application approach. For example, bitcoin's architecture of 1 gigabyte block sizing and 10-minute block-creation intervals has created an aggregate block chain size of approximately 150 gigabytes, and transaction frequencies limited to fewer than 10 per second; these limitations inherent in the known DCL architecture preclude or severely limit its use in high frequency applications such as retail sales and financial markets. Some systems designers have proposed "trusted systems" for speeding up transactions in which a parallel transaction settlement system is run in conjunction with the block chain; in these systems a trusted intermediary executes rapid transaction settlements on a centralized network, and then the intermediary's transactions are released in bulk to the distributed ledger.

However, these types of parallel transaction settlement systems are known to have many drawbacks. For example, one drawback is that running two systems in parallel demands twice the resources to accomplish the same work as a single system. Another drawback is that, since input errors are always a possibility, the probability of an error increases because the amount of data being input doubles.

Thus there still remains a need in the art for a system and method that provides the advantages of current such computer ledger systems without the above drawbacks. Furthermore there also remains a need in the art for a system and method that is compatible with current technology.

SUMMARY

The present invention solves the problems of current state of the art and provides many more benefits. The disclosed improves the storage efficiency, computational processing, transmission speeds, and functional utility of distributed computerized ledgers (DCLs). Included in the disclosed embodiment is a multiple, parallel, and modular system of storage and processing used to improve the function of the computers running on a network with a DCL. The DCL industry is trying to achieve the reality of a peer-to-peer distributed ledger with immutable records of transactions, commercial practicality, rapid communication and transaction times, and universal techniques to operate over a wide range of applications. Known DCL systems have largely been directed at very narrow applications of electronic currencies (or cryptocurrencies), and those electronic currency applications have highlighted many limitations in transmission and processing speeds in known DCL formats. The proposed system and method is universally compatible with current technologies as well as compatible with current alternative settlement arrangements.

In addition to the data storage and requirements for customization, the disclosed has an important advantage over smart contract and similar coding solutions relating to security and tampering. Because the known methods of smart contracts and colored coins are based on distributed computer code, they are more subject to error, loss of security, and hacking. The networks over Ethereum block chain implementation based on smart contracts were recently hacked, and implementations based on distributed code are particularly vulnerable (see "The Ether Thief", Bloomberg, Matthew Leising, Jun. 30, 2017). The disclosed eliminates the risks relating to distributed code, and eliminates the customization required to create, store, and operate new functionality.

The disclosed system, method and computer readable storage medium also solves certain internet based challenges not previously addressed in the industry, including expanding the capabilities of computerized ledgers and repurposing existing narrowly specified computerized ledgers through at least one processing engine capable of running and storing results in either a parallel or integrated storage architecture of automated system entries for purposes including the creation of new electronic property types, and improving the speed and security of computer based transactional networks. The disclosed embodiment utilizes unconventional architecture and processes not routinely integrated in computerized ledger systems.

Known and deployed DCLs have been largely limited to cryptocurrencies and limited networks to track the shipment of goods. Importantly, known cryptocurrency DCLs carry no data other than cryptographic markers (sometimes operating as unique block indicators) and transaction records; the unit counts (or ledger entries) are treated as the item of value and there is no other value attribution, linkage, and generally no convertibility into tangible property or items. DCLs directed at trade and shipping reside on very limited networks where a small number of permissioned parties perform simultaneous write and read operations to a shared ledger; these limited networks are generally specially purposed and have limited scalability. While there have been high expectation for banks and financial exchanges to employ DCLs across many businesses, practical development and actual implementations have been surprisingly low. In the recently published journal article, "*Blockchain and Financial Market Innovation*", the Federal Reserve Bank of Chicago Economic Perspectives, Vol. 41, No. 7, 2017, the federal bank writes "In order to achieve their full potential, implementations of block chain technology will likely be accompanied by smart contracts. Smart contracts are legal contracts written in computer code that execute automatically once certain conditions, specified in the contract, are fulfilled. Smart contracts can be added to distributed ledgers to self-execute on the basis of information in the ledger . . . ". The Federal Reserve Bank of Chicago's position is representative of the consensus view that writing more data into a DCL framework is the future of growth and expansion.

The other known method for achieving non-homogeneity in the units of a DCL is called "colored coins". The website bitcoinwiki <https://en.bitcoin.it> defines colored coins as: " . . . a class of methods for representing and managing real world assets on top of the bitcoin block chain. While originally designed to be a currency, Bitcoin's scripting language allows to [sic] store small amounts of metadata on the block chain, which can be used to represent asset

manipulation instructions." <https://en.bitcoin.it/wiki/Colored_Coins>. Colored coins are a set of limited and preset encoding techniques for simple re-denominations of coins or for initiating self-executing transfers. The implementation of colored coins puts more overhead demands on already challenged DCL ledger processing by expanding the quantities of redundantly distributed data. Known colored coins are also limited in application, if for no other reason than their data demands become impractical when applied to universal solution sets.

The disclosed embodiment departs from consensus in that it is based on alternative storage processes and architecture. The disclosed embodiment is directed at separating the processes and storage of DCL computers, networks and systems, where only those items required for transaction record keeping are maintained in the fully distributed ledger, and all other data, functionality, and processing is stored in a system of decentralized or centralized storage and processing, linked to the distributed ledger through a combination including timestamps, cryptographic strings, cryptographic nonces, or identifying keys. The disclosed embodiment is directed at a material leap in functionality, utility and speed, and it can be applied to both existing DCL data architectures and newly purposed DCLs. Further, the storage architecture of the disclosed embodiment readily extends to highly efficient mixed real-world use, through a modular system of parallel stored and processed differentials; entirely new applications and enhancements can be added with no additional overhead in the transaction ledgers.

In addition to improving the functionality of the hardware and networking components, the disclosed embodiment is directed at transforming the properties and expanding the functionality entries in a DCL in many unconventional ways. To date DCLs have been limited to currencies of simple multi-node database applications. In order to make DCL technologies widely useful, their underlying entries and transaction records need storage and processing technologies to become representative of, convertible into, or based on real world items of markets and commerce, where those real world items of markets and commerce are represented by a mix of datasets of prices, volumes, dates, settlement particulars and other details. For example, an application of the disclosed embodiment relating to the storage, trade, and transport of steel can carry all of the necessary parameters without additional DCL overhead, where examples of the parameters include: type (carbon, alloy, stainless, tool), grade (SAE—Society of Automotive Engineers codes), application (structural, high tensile, pipe), delivery (date, port), and settlement price and currency (US dollars, Euro, Japanese Yen). Further, the same DCL system design and architecture can immediately accommodate any other application including health care, international trade, or financial instrument applications without recoding or providing for additional overhead.

The Achilles heel of the industry is achieving the combination of speed, functionality, and application versatility, while maintaining the benefits of secure, immutable, and distributed transaction records; the disclosed embodiment solves for the long felt needs of higher efficiency and greater functionality with a new data storage, data processing, and computer functionality. Applied to implementations, the system, method, and computer readable storage medium of the disclosed embodiment are capable of creating entirely new computerized ledger arrangements without additional processing headroom or distributed storage.

The disclosed embodiment employs a non-routine system in which a base DCL, or other computerized ledger, records transactions in sequence; the base computerized ledger transaction records may encrypted and distributed over an internet connected network or encrypted and stored on connected decentralized devices. Among the distinctions between the disclosed embodiment and known and conventional methods is that the disclosed embodiment operates at least one parallel, modular, and separate linked computer storage which accesses one or more exogenous published variables or at least one descriptor difference for the purpose of creating new functionality to a base DCL or other computerized ledger. At least one system created differential is generated from exogenous electronic published data, variables or descriptor, where the time frequency of differential generation and storage (while the electronic published data or variables are changing) is at least as frequent as DCL block creations (or similar transaction recording and sequencing formats) of the base computerized ledger. A system of parallel, modular and separate linked storage of generated differentials expands and redefines the use and application of a base computerized ledger without additional overhead or storage requirements for the base computerized ledger. Transactional records may be transmitted through a network with redundancies, but the differences, measurements, or descriptors are stored in parallel, modular and linked arrangements and not within the transaction records.

The above objects are met by the present invention. In addition the above objects and yet other objects and advantages of the present invention will become apparent from the hereinafter-set forth Brief Description of the Drawings, Detailed Description, and claims appended herewith. These features and other features are described and shown in the following drawings and detailed description.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** illustrates one example of data storage and data structure, where a distributed computerized ledger, a time sequence, and parallel storage of difference layers are indicated on the x, y, and z axes;

FIG. **2** illustrates one example of the population of data items in the three dimensional storage array;

FIG. **3** an example of a data storage array, where the transactional records which are transmitted between and among nodes are distinguished and separated from differential items processed and stored separately from the transactional ledger;

FIG. **4** illustrates an example of electronic linkage between and across a distributed computerized ledger storing transaction records and a related array of centrally or decentralized stored differentials;

FIG. **5** illustrates an example implementation of a distributed transaction ledger and a mixed type storage array of values and differentials;

FIG. **6** illustrates examples of three real world applications;

FIG. **7** illustrates an example of a time sequenced storage and linkage relationship between a base ledger and a related storage array of differentials;

FIG. **8** illustrates a second example of a time sequenced storage and linkage relationship between a base ledger and a related storage array of differentials;

FIG. **9** illustrates an example of a multilevel mixed differences storage array in an example of shipping, goods in transit, and international trade;

FIG. **10** illustrates an example of a multilevel mixed differences storage array in an example of an electronically tradable instrument.

FIG. **11** is a table illustration that is an example of exogenous differential array storage of a single item with an example record format;

FIG. **12** illustrates relative storage locations and functional alignments between distributed computerized ledgers and parallel storage of differences layers;

FIG. **13** illustrates a sample range of parallel storage of difference layer data types;

FIG. **14** is a table that illustrates descriptive and binary string storage;

FIG. **15** is a diagram a basic centralized computerized ledger;

FIG. **16** is a diagram a basic decentralized computerized ledger;

FIG. **17** is a diagram a basic distributed computerized ledger; and

FIG. **18** is a schematic diagram which depicts the internet and network connectivity of the system hardware including the differentials processor, differentials storage, and system non-transitory computer medium and internet linkage to nodes and networks.

DETAILED DESCRIPTION

In general, the present invention overcomes the disadvantages of current computer ledgers. In addition, the present invention works in an entirely new and different way than prior attempts to overcome computer ledger drawbacks. The disclosed embodiment is an unconventional system, method, and computer readable storage medium which creates and operates a high efficiency computerized ledger capable of universal application to a range of industrial, commercial and financial implementations. Included in the disclosed embodiment are computer storage, related processes, and methods which change and expand the use and applications of known computerized ledgers. The disclosed embodiment is directed at non-centralized electronic ledgers in which transaction records are distributed to multiple nodes within a network, and where multiple copies of transaction records (in summary or detail) are maintained on separate computers connected to a network. Such non-centralized networks include both decentralized networks and distributed networks, and also include networks in which participants engage in direct or indirect peer-to-peer transactions. The disclosed embodiment improves the functioning of the related computers, computer systems, the network communication equipment and related processes. Further, the disclosed embodiment expands the usefulness and versatility of existing ledgers, transactional storage systems and distributed ledger communication networks.

The disclosed embodiment reduces the storage requirements of known methods, improves network transaction speeds, reduces the amounts of transmitted data relating to transactions, and expands the utility and functionality of computerized ledgers. The detailed embodiments disclosed are examples, and the systems, methods, and computer readable storage medium can be embodied in many forms. The specific design, structure, and method details described herein are not meant to be limiting. In addition, terms and descriptors used herein are not intended to be limiting, but rather to illustrate the concepts.

Computerized ledgers are generally used to record the transaction records of interests or units, where the interests or units are purchased, sold, transferred, conveyed, split or where ownership stakes are otherwise altered. To date, large scale applications of computerized ledgers have been largely limited to electronic or so-called cryptocurrencies. A cryptocurrency operating on a network has extremely limited data demands; coins are identical and fungible, and the only requirement to track movements in the network is a basic sequenced listing record of transactions of the homogeneous units; all units or interests on the DCL network are identical and fungible and require no stored or process descriptors. The DCL has many benefits in the context of a single coin network, in that the ledger can be fully distributed and transparent. Coin based DCL networks track a single transaction type where validation and immutability of the post-validation record trail require basic cryptographic techniques to properly sequence the peer-to-peer transactions. However, as the industry has attempted expansions beyond homogeneous coins, the simple architecture of the basic ledger has introduced limitations. Widely promoted add-ons such as colored coins and smart contracts, are very limited in implementation, require implementation-specific customization, put much larger demands on data transmission and data storage, and introduce unique security risks.

A number of DCLs are currently in implemented as block chain operations over cryptocurrencies. Block chain (or block chain) ledgers are typically DCL formats where transactions are sequenced and bundled into discrete units or blocks, aggregated by time, where the blocks are sequenced through a combination of timestamps and internal generated cryptographic codes. The basic design objectives of the block chain DCLs are to ensure that transactions which get recorded in a block are validated through a consensus process, that blocks are properly sequenced as they are appended onto the chain, and that their sequencing and recording is immutable.

Known computerized ledgers are principally designed and built for electronic currencies, which did not previously exist, and only exist within the framework of the DCL block chain. Known systems and methods have attempted to extend the DCL to other types of items and applications, most of which require a high degree of detailed specification and data overhead. Systems developers are finding that extending the known DCL methods to applications requiring specification requires overly complex bespoke solutions for each application, and that the bespoke solutions create material burdens on networks.

The disclosed embodiment is a departure in systems, storage, method, and data architecture. The disclosed embodiment changes design and methods of data storage and the functionality of a DCL. The disclosed embodiment is partially based on the concepts: (i) electronic transactions within a DCL can be independent and separately processed from the data items required to specify a value, disposition, distribution, or resolution of a unit of the DCL, (ii) direct processing of a DCL and available network and system capacity must be directed at the highest levels of transaction and execution speed, rather than DCL internal specification, and (iii) many real world applications of DCL will relate to already specified real world objects, and the articulation of those items can generally be dynamically imputed to the DCL interests through linked and modular storage systems.

Included in the purposes of the system's separated parallel storage of differentials dynamically linked to a base DCL are a reduction in redundant data transmission and faster processing of transactions; a base DCL can be entirely repurposed through the disclosed embodiment with no increase in base DCL storage requirements or computational overhead. Also included in the purposes of the modular structure of separated parallel storage of differentials is increased utility and functionality of the systems running basic DCLs; the disclosed embodiment can repurpose a base DCL, by cre-

ating new electronically tradable items without any customization to the DCL, and the modularity of the disclosed embodiment allows rapid and dynamic changes without increased data transmission or data storage overhead.

Beginning with FIG. **1** diagram **10**, the diagram is a simplified illustration of system storage. A base electronic computerized ledger **11** is illustrated as the single layer where computerized ledger transaction records are created and modified including the actions of writing, appending, and reading, where the base electronic ledger resides on a distributed or decentralized ledger. All of the transaction records reside on the base electronic ledger **11**, and transaction records may be appended to the ledger grouped within blocks or appended individually. Transaction records will be identifiable by at least one of a system timestamp, a network timestamp, a unique system generated identifier, or a cryptographic identifier. When written or appended in groups or blocks, the related group or block and component transaction records may be identified collectively in some implementations (the disclosed embodiment reads and processes transaction record data in time sequenced groups), and may be identified independently in other implementations (the disclosed embodiment reads and processes transaction record data based on individual transactions).

Continuing with FIG. **1** diagram **10**, the parallel storage of differences is indicated by parallel storage of difference layer (PSDL) **12** and PSDL **13**. An implementation of the system includes at least one PSDL. Each PSDL will store at least one system written and system accessible time sequenced differential or descriptor, where differentials are created by the system from exogenous and electronically published data streams, and where at least one differences processing engine running on the system computes and stores time sequenced differences from values in the published data stream. Differentials recorded on a PSDL may also include descriptive differentials which can indicate difference types, grades, timeframes or other discriminatory identifiers; descriptive differentials may be utilized with or without data stream differentials. In certain implementations, a descriptive differential is an indirect reference to electronically published data streams; for example a descriptive differential which indicates a certain type of steel of a certain grade to a DCL unit imparts a delivery obligation or value which aligns with one or more electronically published data streams.

The differences residing on a PDSL are applied to the units (or interests) of a DCL upon a system occurrence of an action or process including a value polling, a distribution, a resolution or settlement, or other processes requiring the supplementary data in the PSDL. Cryptographic encoding, transaction validation, and consensus proofing process operations on the DCL may or may not access PSDLs. The system may apply each PDSL to the related units in sequence (i.e. PSDL**1**, then PSDL**2**) or simultaneously (i.e. PSDL**1** and PSDL**2** at the same time). Examples of the time sequenced exogenous and electronically published data include: (i) the prices of computer memory storage devices, (ii) prices of crude oil of differing grades, at different delivery points, denominated in different currencies, (iii) voter counts in statewide election by demographic, party affiliation, and geographic location.

Continuing with FIG. **1** diagram **10**, in DCL implementations of fully distributed or decentralized transaction record storage, the base electronic computerized ledger **11** may be transmitted across a network of computers, where multiple original copies reside on multiple network nodes. The PDSL data **12** and PSDL data **13** (and related processes) are not distributed to every node within the network, and in certain implementations of the system, all PSDL items and processes are maintained on a centralized devices or limited decentralized storage devices. The system's interaction with and between system's stored data layers, and the modifications to the base DCL ledger units (based on the PSDL data storage) may include any mathematical or logic computer operator or operation.

Moving to FIG. **2** diagram **20**, the diagram is an extension of diagram **10**, where diagram **20** illustrates an example of recording transaction records to the base DCL (which is distributed with multiple original copies in a distributed or decentralized network), and the simultaneous, advanced, or lagged writing of differences (on the PSDL) from one or more exogenous electronically published internet data streams or descriptors. The modularity of the linked and mixed storage are illustrated within the three dimensional axes of: (x) time as measured by network computer clocks, or other highly accurate timekeeping devices, (y) the layer or depth of items which relates to the content detail of each appended record in the PSDLs **12** and **13**, and (z) the changeable number and modularity of layers, where storage layers can be added, removed or otherwise modified, and where such modifications directly interoperate with the base DCL without the addition of network or system overhead, and without the security risks of distributed code. Illustrative detailed examples of transaction records are illustrated on the base DCL, where example appended blocks are denoted as "B", and the values "001" through "N" denote the sequencing of the appended blocks. Illustrative examples of PSDL stored differences are denoted on PSDL **12** and PSDL **13**, where "DIF1" and "DIS1" indicate differential items, and the values "001" through "N" illustrate a time alignment of PSDL items with base DCL transaction records.

Diagram **30** of FIG. **3** illustrates the system's separation of the base DCL transaction records **11**, from the parallel storage of difference layers **12**, **13**, and **31**, where a third storage layer in the PSDL is indicated at **31**. Diagram **30** is an example illustration of three separate PSDLs operating over a single base DCL. It can be noted that only the base DCL **11** is redundantly distributed (each may be an original and consensus proofed copy) over individual nodes of the network, while the PSDL data is stored in a centralized or decentralized network. FIG. **3** diagram **30** illustrates an example where multiple modular layers of stored (and operative) differences (the PSDLs) are time sequenced, and where time sequences are aligned with system writing and appending of transaction records in the base DCL individually or in groups (or blocks). Diagram **30** is an example of the modularity of the system and an illustration in the system's efficiency in storage operations. The system's PSDLs are modular, and implementations of the system can create entirely new computerized storage of entirely new functional electronic ledger items using already implemented or new DCLs. Further, because the system separates the storage requirements and computer processing overhead related to the PSDL data, there are no additional DCL storage requirements, and no additional computing overhead on the transaction record keeping network and systems; all while the disclosed embodiment transforms the utility and function of the base DCL.

Referring briefly to FIG. **3**, a base computerized ledger is indicated at **11** where appended blocks (of records) or individual records are added to the ledger in sequence (from left to right in the figure). A modular layer one (**12**), a modular layer two (**13**), and a modular layer three (**31**) are system generated stored differentials or descriptors which

are collectively applied to the base computerized ledger to create a useful, convertible and real world applications where entries within the computerized ledger **11** can have attributable and constructive discriminatory properties, and where differentials relate to any number of objects or transactions including, shipping records, commodity delivery, or a bespoke financial instrument. The base computerized ledger **11** may be redundantly transmitted and modified across a distributed network similar to the manner in which known electronic currencies operate, but the operative modifying modular layers are generated and stored once or in other limited decentralized distribution, and linked to the base computerized ledger **11** through a computer generated timestamp, a timestamp sequenced key, a unique character string, a cryptographic nonce, or similar unique identifier; records of the modular layers which differ in value or descriptor and time will have a unique alignment with records in the base DCL. Complex and multifactor arrangements may be stored, processed, and transmitted without the introduction of additional overhead in the transacting network and without the customization which is required with the implementation of techniques such as colored coins or smart contracts.

FIG. **4**, diagram **40** is an illustration of an example of system data linkages of the distributed or decentralized base DCL to the centralized or decentralized PSDL. Beginning with the base DCL **11**, basic details relating to a transaction or a group of transactions are indicated in the base DCL diagram **11**. Beginning with "Timestamp" and moving clockwise, "Timestamp" is generally a unique entry in the DCL, recording the system time when a block was found (in certain cryptographic mining processes) or recording the system time when one or more transactions are written on or appended to the base DCL dataset. "Nonce" is generally a one-time use number added to a group or block record in certain base DCL applications. The "Nonce" serves many purposes including cryptographic security, immutable record sequencing and it is part of the consensus validation in certain applications involving cryptographic mining. The "Tx_Root" is generally the connection to recorded transactions, and "Prev_Hash" is a hash from the immediately preceding block, which ensures that each block is immutably tied to previous block.

Continuing with FIG. **4** diagram **40**, example detail is added to PSDL **12** and PSDL **13**. Differing implementations may include different PSDL detail, and importantly, there is no requirement that the data stored in each PSDL be identical. "Time PSDL" is an example of the principle linkage, where the field stores a unique identifier which may be in the form of a string, or value such that the value is linked to a concurrent, delayed, or advanced identifying entry in the DCL. "Dif1_C(N)" is a difference measuring a subject data items over a defined period of time, where "Dif1_C(N)" can be descriptive or arithmetic. "Cur_Val" and "Prev_Val" are descriptive or arithmetic values stored on PSDL **12** and PSDL **13** and relate to exogenous items, where "Prev_Val" indicates the immediately preceding or early system time, and "Cur_Val" indicates a later time or current time. Both the DCL and PSDL have independent record sequences, and each record, block, or transaction within a DCL will have an attribution to a location in the stored PSDL, and the DCL-to-PSDL connector **41** links the distributed DCL to the centralized or decentralized PSDLs based on unique codes, time indicators, or similar items.

FIG. **5** diagram **50** is an example of the increased functionality of a base DCL format through the storage of the disclosed embodiment. Three PSDLs are illustrated as examples relating to using the disclosed embodiment in the context of commodities transport and trade. The base DCL **11** is shown, and the PSDLs **51**, **52**, and **53** illustrate example specifications for a PSDL linked to the base DCL. In each illustrated example (of three sets of PSDLs), the lower layer indicates a shipped commodity type (Type W, B, and D), the middle layer indicates the currency in the data (EUR=Euro, USD=U.S. Dollar, and SGD=Singapore Dollar), and the topmost layer indicates descriptive differences (−1, +1 and −1, from right to left at **51**, **52**, and **53** respectively). PSDL **51** is an example where the exogenous data streams are Type W (WTI Crude Oil), and Euro currency (EUR), and a descriptive difference of −1, where −1 may indicate an obligation to deliver WTI Crude Oil denominated in Euros. PSDL **52** is an example where the exogenous data streams are Type B (Brent Crude Oil), and U.S. Dollars currency (USD), and a descriptive difference of +1, where +1 may indicate an obligation (or operative entitlement) to take delivery of Brent Crude Oil denominated in USD. PSDL **53** is an example where the exogenous data streams are Type D (Dubai Crude Oil), and Singapore Dollars currency (SGD), and a descriptive difference of −1, where −1 may indicate an obligation (or operative entitlement) to make delivery of Dubai Crude Oil denominated in SGD. The lowest two layers of each PSDL illustrate stored data and operations are over a time sequenced exogenous data item; values are drawn from one or more internet data streams containing the requisite commodities and currency values. The topmost layer is a descriptive difference layer, where −1 may be used to indicate a delivery obligation, and where +1 may be used to indicate an obligation to take delivery; descriptive difference layers may be used to change the direction and responsiveness to one or more exogenous data items. The disclosed embodiment is not expected to alter the quantities, prices, or types of commodities which might be transacted and monitored in a computer system organized to manage and monitor commercial trade. However, the disclosed embodiment is expected to materially decrease the storage and transmission requirements of related distributed ledger networks and computer systems. Further, the centralized or decentralized storage of the PSDL data and operations is expected to material increase the network security of the computer systems and networks.

Moving to FIG. **6** diagram **60**, values of differences are illustrated for PSDL **51**, PSDL **52**, and PSDL **53**. The value of differences is generated by the system from one of more internet data streams, and where practical, the values of differences are generated, stored, and linked with a frequency which matches or exceeds the frequency used for appending transactions records to the base DCL **11** during periods in which the values of difference published in an internet data streams are changing. In each case, the value differences illustrated for TYPE W, TYPE B, TYPE D, EUR, USD, and SGD are differences in percentage changes from the immediately preceding period; in alternate implementations, absolute values or other measured changes may be generated, stored and applied. The value differences in the right-most layer of each PSDL are descriptive value differences which may indicate a relative directionality or degree of difference application. For clarity, one PSDL is generally applied to only one base DCL, however a single PSDL may have more stored layers than are indicated in the figures.

Continuing with FIG. **6** diagram **60**, one example of applying a PSDL to the units of the base DCL **11** is through the use of computer mathematic operators, where each layer's numerical difference storage layer is applied to

produce an aggregate impact. Beginning with PSDL **51**, the base DCL **11** units can be modified in one example by using a multiplication operator for each layer, where the units are modified by the product of: 1.1% (Type W), 0.1% (EUR), and −1 (L:−1) for a modification of −1.012011% (1.011× 1.001×−1). Applying the same operator to PSDL **52** and PSDL **53** results in 1.009%, and −1.00798% respectively. For clarity, the base DCL is stored and transacted independent of the PSDL storage. Also for clarity, PSDL storage may impart absolute values (where absolute values are subject to differences over time sequences), descriptive characteristics, or a mix of absolute values, relative values, and descriptors. The mathematical functions, operators and results are incidental to the disclosed embodiment's storage and processing and not central to the operation of the specialized system or storage design. The PSDL modifications may be electronically published over a network or internet such that holders or transactors can monitor aggregate modifying impacts, and the impact of PSDL modification may be coupled with the base DCL to direct reporting to holders, transactors, and other participants. Importantly, the system data, differences, and processes of the PSDLs is not propagated through or contained in the transaction ledger.

FIG. **7** diagram **70** is an example of alternative alignments of PSDL data storage **71**, and base DCL data storage **11**. Timeline **72** indicates a starting time of T(**0**) and three forward time intervals indicated as T(**1**), T(**2**), and T(**3**). PSDL data storage **71** is an aggregation of individual PSDL layers **12**, **13**, and **31**. In diagram **70**, the units of the base DCL **11** are impacted by forward-looking or advance values and differences of three PSDLs. Importantly, the PSDL data and processes are not stored in the base DCL nor is the PSDL data and processes redundantly distributed throughout nodes of the network as are known methods such as smart contracts and colored coins.

FIG. **8** diagram **80** is a second example of an alternative alignment of PSDL storage **71**, and base DCL storage **11**. Timeline **81** indicates a starting time of T(**0**) and two additional arrears time intervals indicated as T(**1**), T(**2**), and a current time of T(**3**). PSDL data storage **71** is an aggregation of individual PSDL layers **12**, **13**, and **31**. In diagram **80**, the units of the base DCL **11** are impacted by backward-looking or arrears values and differences of three PSDLs. Relating to diagrams **70** and **80**, because non-transaction record data is not distributed through all nodes of the network, but rather is separately and centrally stored in a modular framework, functionality and utility is materially increased because PSDL are limited in distribution and modular, transaction and speeds are improved because only transaction records need be fully distributed, and security is enhanced because transaction records can benefit from an immutable sequencing on the ledger, and operative differences PSDL data and processes can be stored on secured centralized or decentralized systems.

Referring briefly to FIG. **9**, a base computerized ledger (**11**) is operatively linked to parallel and modular differences of factors one through three (**91**, **92**, and **93** respectively, stored in parallel storage of differences layers **11**, **12**, and **31** respectively) over a multi-time period measured and stored differences, only the single record indicated at base computerized ledger **11** is transmitted across the multiple nodes of the electronic network; the parallel and modular differences of factors are centrally stored, not distributed with the transactions ledger, and accessed only when units (or interests) underlying the base computerized ledger (**11**) are subject to a resolution, disposition, valuation, settlement, distribution or another action requiring differentials and descriptors. FIG. **9** is an illustration of one particular implementation where highly reduced storage and high functionality is achieved. Based on 3 modular levels of parallel storage, each containing 6 stored attributes, 6^3 or 216 unique applications are available with one base computerized ledger data structure **11**, and data transmission requirements and storage redundancy is minimized. In contrast, units of known distributed electronic ledgers are homogeneous; one bitcoin or Ethereum coin is identical to any other and the addition of features using known methods where additional items are added to the blocks (or other sequential records) expand the block storage and transmission overhead exponential and cause an increase is the amount of redundant data which propagates throughout network nodes.

FIG. **9**, diagram **90** illustrates an example of the storage of illustrative PSDLs **12**, **13**, and **31**. Diagram **90** is an example of the PSDL storage and operation over base DCL **11**, where a PSDL may contain value differences or descriptive differences. The PSDLs in the example relate to a real world example of oil trade in which PSDL **12** indicates a type of crude oil (WTI, WTI-H, WTS, MARS, LOOP indicated at **91**), PSDL **13** indicates a port (or delivery) location (PORT_A, PORT_B, PORT_C indicated at **92**), and PSDL **31** indicates a delivery time or delivery cycle (1 W, 1 M, 2 M indicated at **93**). PSDL **12**, PSDL **13**, and PSDL **31** all operate over the units or interests of the base DCL **11**, however all storage relating to the PSDL data is not stored in the base DCL, but rather is stored centrally and linked to the resultant transaction records of the base DCL. One example of value differentials for oil type **91** is a storage of time-sequenced price values where the prices of one or more of the oil types is recorded in the PSDL for the storage and application of differences. Another example for oil type differentials **91** is the use of descriptive differentials relating to oil types **91**, where the system stores and applies strings or binary encoding rather than basic numerical values; a string example is "WTI-H", and the binary descriptive coding is 01000 to indicate the second oil type in a list of five.

FIG. **10** diagram **100** is an extended example of diagram **90**, which illustrates an implementation in the context of the creation a range of bespoke instruments over a base DCL **11** with 3 parallel storage of difference layers. In diagram **100**, the lowest storage layer **12** is a differences of currencies, where the currencies indicated in a selector **101** are U.S. Dollar (USD), Euro (EUR), Japanese Yen (JPY), Chinese Yuan (CNY), and British Pound Sterling (GBP); storage layer **12** can contain descriptive differences (e.g. which among the group of currencies is indicated), stored value absolute differences for relating to absolute values such as foreign exchange rates in a time sequence, or relative change or percentage differentials in a time sequences; descriptive differentials may be stored and indicated as indicators, indicator flags, binary values, or character strings. PSDL **13**, the second storage layer is used to indicate tenor of an actual or synthetic position desired for the aggregate PSDL (where the aggregate PSDL is the combination of storage levels **12**, **13**, and **31**). Relating to PSDL **13**, selector **102** can indicate "1 d", "1 w", "1 m", "1 Y" (1 day, 1 week, 1 month, and 1 year respectively). Relating to PSDL **31**, selector **103** references specific instruments which can be stored as description differences or time sequenced differences of absolute values or relative percentage changes. Selector **103** can indicate and store SPX, GT10, NDX, or HYLD (an equity market index, a bond index, a tech equity index, and a credit index respectively).

FIG. **11** diagram **110** is an example of a system storage record which may be stored within a PSDL. Table **111** indicates an example of a particular record where the fields are indicated as "identifier", "timestamp(0)", timestamp(t)" "val(0)", "val(t)", "dif(t)", and "cond". Identifier is an example of an encoding which is used to identify the subject of the stored differentials or descriptors. Timestamp(0) and timestamp(1) indicate the beginning and ending timestamps of a computer system or network time (and the related period) over which the stored difference is based. Val(0) and val(1) indicate the respective values of the subject item taken from an electronic published data stream, where val(0) relates to timestamp(0) in observation time, and where val(1) related to timestamp(1) in observation time. Dif(t) is an example of a numerical difference over the subject data stream object from timestamp(0) to timestamp(1). The field "cond." may be used to indicate successful retrieval from an internet data stream. The values and forms of values in table **111** are examples; implementations may present different record architecture, different stored data items, and different types of values.

FIG. **12** diagram **120** illustrates the separation of storage of the base DCL **122** and the PSDL layers **125** and **126**. Line **121** illustrates the dividing line of storage where items above the line are stored on the system in centralized or decentralized storage, and those items below the line are stored in a distributed ledger. Diagram **120** is an example of the modularity of the system, and two PSDLs are illustrated at **125** and **126** respectively. The single base DCL (labelled "BB" or base block records) is indicated at **122**. In each of the four columns of diagram **120**, the base DCL **122** may be impacted by each PSDL **126** and PSDL **125** vertical pair individually, where each time-sequenced entry in the aggregate PSDL is independent from the others, or the base DCL **122** may be impacted by the PSDL **126** and PSDL **125** aggregate PSDL in a cumulative or compounding manner, where the impact is cumulative as time moves from left to right as indicated by the time indicators t(**1**), t(**2**), t(**3**), and t(**4**).

FIG. **13** diagram **130** illustrates an example of differentials stored and processed by the system, where differentials are based on any mathematical or computer operator. Row **131** illustrates a time sequenced vector of values equal to 100, 101, 101.5, and 99. Rows **132**, **133**, and **134** are illustrations of differentials stored and processed by the system. In the example of row **132**, the system stores and processes differentials based on value change; value changes may be based on differences or absolute values. In the example of row **133**, the system stores and processes changing time sequenced values as the value of the time sequenced different and changing values. In the example if row **134**, the system stores and processes the values as percentage differences based on the immediately preceding value.

FIG. **14** diagram **140** illustrates an example of differentials where differentials are descriptive. As illustrated in the table of diagram **140**, the first column entitled "VAL" lists a set of unique string values "USD", "EUR", "JPY", "CNY", and "GBP". As illustrated in the example of the column entitled "DIF1", the differentials stored and processed by the system may be the actual string values which differ within the set of values (i.e. the differential descriptor). As illustrated in example of the column entitled "DIF2", the differentials stored and processed may be an indicator, flag, or binary string which indicates a value out of a set; where the set is comprised of 6 values, "100000" may be used to indicate the first value ("USD"), and "000001" may be used to indicate the last value "GBP". The remote and linked storage of descriptive items can be used to transform the nature of units within a base DCL or to strategically discriminate within the units for specialized remote processing or handling; units become representative of other items such as an industrial good, a commodity, or an airline ticket, and units within the block can be further discriminated by time or other transaction record parameters.

FIG. **15** diagram **150** is an example of a centralized network, as such term is used in the disclosed embodiment. Generally, all nodes in the network connect in a centralized manner to a central point or hub. Security in a centralized network is generally based on securing the media, device, and operating processes at the central point, and limiting write, modification and certain read access at the centralized point. Centralized networks enjoy the security advantage of having limited points or vulnerability, however, centralized networks can become highly compromised if the central point is breached.

FIG. **16** diagram **160** is an example of a decentralized network as such term is used in the disclosed embodiment. Generally, there are a number of central or connecting points at which individual nodes may connect. Similar to a centralized network, security in a decentralized network is generally based on securing the media, device, and operating processes at the common connection points, and limiting write, modification, and certain read access at the decentralized points.

FIG. **17** diagram **170** is an example of a distributed network as such term is used in the disclosed embodiment. Generally, each individual node is connected (or capable of being connected) to every other node in the network. Implementations of distributed computerized ledgers are generally configured in a manner similar to FIG. **17**. Security in a distributed network is often achieved through cryptographic techniques within a ledger in combination with a consensus system for validating transaction records written and appended to the ledger. Further, diagram **170** is an illustrating example of the configuration of one base DCL, where a ledger of transaction records is distributed among connected computer nodes. For clarity, intermediary nodes may be used as surrogates for discrete nodes, where such intermediary nodes are exchanges or other intermediary service providers who hold interests in a base DCL for limited periods of time or for rapid collective settlement.

Referring to FIG. **18** diagram **180**, it is to be understood that the disclosed embodiment includes storage of differentials and processing of differentials computer node **191**, where differentials and storage processing software is stored on system RAM **192**. The system software instructs the system continuously relating to all aspects of the differentials storage, including processing and storage of each modular PSDL, interconnectivity with at least one time-sequenced electronically published data stream or descriptive differential, and linkage to at least one base DCL. The differentials computer node **191** and related RAM may also be linked to a differentials computer node storage device **193**.

Continuing with FIG. **18**, an example of a base DCL is indicated at **181** where the separate computing nodes of a network are interconnected through the internet or a network, and where a ledger of transactions is distributed across the network. An example of the base DCL connection with the rest of the system is indicated at the distributed ledger I/O interface(s) **189**, where the distributed ledger I/O interface(s) **189** may operate in both a transmission and receiving mode. The distributed ledger I/O interface(s) **189** is further connected to a system processing unit **190**.

Continuing with FIG. **18**, an example of a raw differential I/O interface(s) **186** is connected through an internet connection **182** for the purposes of retrieving one or more time-sequenced electronically published data streams or a descriptive differentials, where a time-sequenced data stream may relate to prices, trade flows, trade variables, shipping details, economic variables, performance measures or other numerical or descriptive data. Examples of source nodes connected to the internet include: (i) a commercial trade, tracking, or shipping network run by a company, industry group, or governmental entity **183**, (ii) an electronic exchange **184** which publishes a price data stream of changing market prices, and (iii) an electronic news outlet **185** which publishes electronic data relating to changing news. Continuing to the differential processor **187**, the differential processor **187** assimilates at least one value or descriptive differential through a computer or mathematical operation, forming the data into a useable format where it can be stored on differential storage unit **188**, for simultaneous or subsequent application to the units or interests of a base DCL with network **181**. Continuing with the processing of differentials, the system processing unit **190** is connected to the differential processor and the differentials computer node **191** such that the system can effect differential data transmissions and responses to differential data queries through a processed differential data I/O interface **194** which is controlled by a differentials administration gateway terminal **195**. An example of processed differential transmission, dissemination, and query management is illustrated at internet connection **196** in which the system can both broadcast processed differential data and respond to queries. When a unit or interest of the base DCL requires valuation, settlement, exchange, or resolution, the system can be polled for a valuation or impact on a unit, record, or interest of the DCL as of a particular time, or over a particular period of time.

Continuing with diagram **180**, an example of an administrator interface **197** is controlled by an administrator console **198**, where the specifications of PSDLs, related exogenous data streams, and connectivity to a base DCL is established, controlled, and modified.

The disclosed embodiment is a system, method, and computer readable storage medium related to the systems, network connectivity, software, and data storage architecture in applications over computerized ledgers. The disclosed system, method and computer readable storage medium is directed at a range of computerized ledgers, including distributed ledgers, decentralized ledgers, and centralized ledgers, where computerized ledgers store and report encrypted, or otherwise secured, electronic transactions. The disclosed system, including its data storage features, computer readable storage medium, and methods are directed at universal solutions to improve the efficiency and utility of computers and networks operating computerized ledgers.

The above disclosed embodiments are not intended to limit the scope of the invention but are examples thereof. Although the invention herein has been described with reference to particular embodiments, it is to be understood that these embodiments are merely illustrative of the principles and applications of the present invention. The above disclosed embodiments were chosen and described to most clearly explain the principles of the invention and practical applications, and to enable others skilled in the art to understand the invention for various embodiments. It is therefore to be understood that numerous modifications may be made to the illustrative embodiments and that other arrangements may be devised without departing from the scope and spirit of the present invention as defined by the appended claims.

What is claimed:

1. A computer based method comprising:

creating at least one electronic parallel storage of a differences layer linked to a distributed computer ledger (DCL); the DCL contains an electronic transaction record by a time-sequenced value or a time-sequenced string;

accessing and storing a value through the at least one electronic parallel storage of the differences layer, the value from a group comprising of at least one time-sequenced electronically published data stream and at least one descriptive differential, wherein at least one differences processing engine running on a specialized computer system creates and stores parameters from a group comprised of a measurement differences and a descriptive differences;

storing the DCL containing an electronic transactions record on at least one of a distributed network of connected independent computers or a decentralized network of computers wherein the electronic transaction record is time sequenced, and a writing or an appending of the electronic transaction records is performed on the distributed network of connected independent computers or the decentralized network of computers;

storing the at least one electronic parallel storage of the differences layer on at least one of a centralized storage device controlled by the specialized computer system or a decentralized storage device controlled by the specialized computer system for increasing functionality and utility of the DCL, reducing data storage requirements, eliminating transmission of redundant data, and improving data security;

linking the electronic transaction record in the DCL to records of the at least one electronic parallel storage of the differences layer utilizing at least one time sequenced value, string, code, or key; and

imputing at least one measured differential with a descriptive identifier or at least one descriptive identifier to the electronic transaction record of the DCL through data storage and processing on the at least one electronic parallel storage of the differences layer.

2. The method of claim **1**, wherein records of the at least one electronic parallel storage of the differences layer are written and stored separately from the distributed electronic ledger containing electronic transaction records, where the records of the at least one electronic parallel storage of the differences layer are encoded for time-sequenced alignment with the electronic transaction records when values from a group comprised of the at least one time-sequenced electronically published data stream and the at least one descriptive differential change in value or specification.

3. The method of claim **1**, wherein values and descriptors from a group comprised of the at least one time-sequenced electronically published data stream and the at least one descriptive differential alter the functionality and transactional value of the electronic transaction records of the distributed electronic ledger.

4. The method of claim **1**, wherein values and descriptors from a group comprised of the at least one time-sequenced electronically published data stream and the at least one descriptive differential define the functionality and operative entitlement of the electronic transaction records of the distributed electronic ledger.

5. The method of claim 1, wherein values from a group consisting of at least one time-sequenced electronically published data stream and at least one descriptive differential are linked to the electronic transaction records within the distributed electronic ledger and the electronic transaction records are homogeneous on the distributed electronic ledger as identified by a timestamp or other unique record identifier.

6. The method of claim 1, wherein layers of the at least one electronic parallel storage of the differences layer linked are modular and changeable independent of the distributed electronic ledger containing electronic transaction records.

7. A system comprising:

a system having a memory device, the memory device further including a Random Access Memory (RAM);

a processor connected to the memory device, the processor is configured to:

create at least one electronic parallel storage of a differences layer linked to a distributed computer ledger (DCL), both the electronic parallel storage of the differences layer and the DCL containing a respective electronic transaction record, a time-sequenced value, or a time-sequenced string;

access a value from a group comprising of at least one time-sequenced electronically published data stream and at least one descriptive differential;

store the values from a group comprising of at least one time-sequenced electronically published data stream and at least one descriptive differential on the at least one electronic parallel storage of the differences layer;

align and link a stored value record of the at least one electronic parallel storage of the differences layer to the electronic transaction record of the DCL utilizing at least one time sequenced value, string, code, or key; and

impute at least one measured differential with a descriptive identifier or at least one descriptive identifier to the electronic transaction record of the DCL.

8. The system of claim 7, wherein the memory device includes a separation of storage of the differences layer.

9. The system of claim 8, wherein the separation of storage is between the electronic transaction record of the DCL and the differences layer.

10. The system of claim 9, wherein a plurality of differences layer is parallel stored to create a parallel storage of differences layer (PSDL).

11. The system of claim 7, wherein the difference layer is stored on a centralized storage or a decentralized storage apart from the electronic transaction record of the DCL.

12. The system of claim 11, wherein the electronic transaction record of the DCL is impacted by a parallel storage of differences layer.

13. The system of claim 12, wherein impact is done from each of the parallel storage of differences layer (PSDL) in an individual manner.

14. The system of claim 13, wherein the parallel storage of differences layer (PSDL) has a time-sequence entry, and each time-sequenced entry is independent in the PSDL.

15. The system of claim 12, wherein impact is done from the parallel storage of differences layer (PSDL) in a cumulative manner, or a compounding manner, wherein impact is cumulative based on a time indicator.

16. The system of claim 15, wherein the parallel storage of differences layer (PSDL) has a time-sequence entry, and each time-sequenced entry is independent or dependent in the PSDL.

17. The system of claim 7, wherein the difference layer is stored on a distributed network, a centralized network, or a decentralized network, and wherein the difference layer is stored apart from the electronic transaction record of the DCL.

18. The system of claim 17, wherein the electronic transaction record of the DCL is impacted by the differences layer.

19. A non-transitory computer readable storage medium, comprising storage, retrieval, modification, and linking system software which instructs at least one computer processor residing on a specialized computer system to implement a process to:

create at least one electronic parallel storage of a differences layer linked to a distributed computer ledger (DCL) containing an electronic transaction record arranged by a time-sequenced value or time-sequenced string, wherein the at least one electronic parallel storage of the differences layer accesses and stores values from a group consisting of at least one time-sequenced electronically published data stream and a list of descriptive differentials, and wherein at least one differences processing engine running on a specialized computer system creates and stores parameters from a group consisting of measurement differences and descriptive differences;

store the DCL containing the electronic transactions records on at least one of a distributed network of connected independent computers or a decentralized network of computers wherein the electronic transaction records are time sequenced, and the writing or appending of the electronic transaction records is performed on the distributed network of connected independent computers or the decentralized network of computers;

store the at least one electronic parallel storage of the differences layer on at least one of a centralized storage device controlled by the specialized computer system or a decentralized storage device for increasing functionality and utility of the DCL, reducing data storage requirements, eliminating transmission of redundant data, and improving data security;

link the transaction records in the DCL to the at least one electronic parallel storage of the differences layer utilizing at least one time sequenced value, string, code, or key; and

impute at least one measured differential with a descriptive identifier or at least one descriptive identifier to the electronic transaction records of the DCL, wherein a data storage and a processing of the imputing resides on a centralized device or a decentralized device controlled by the specialized computer system.

20. The non-transitory computer readable storage medium of claim 19, wherein the difference layer is stored apart from the electronic transaction record of the DCL, and the electronic transaction record of the DCL is impacted by the differences layer.

* * * * *

EXHIBIT B

The Maker Protocol: MakerDAO's Multi-Collateral Dai (MCD) System

Abstract

The Maker Protocol, also known as the Multi-Collateral Dai (MCD) system, allows users to generate Dai by leveraging collateral assets approved by "Maker Governance." Maker Governance is the community organized and operated process of managing the various aspects of the Maker Protocol. Dai is a decentralized, unbiased, collateral-backed cryptocurrency soft-pegged to the US Dollar. Resistant to hyperinflation due to its low volatility, Dai offers economic freedom and opportunity to anyone, anywhere.

This white paper is a reader-friendly description of the Protocol, which is built on the Ethereum blockchain. Technically savvy users might want to head directly to Introduction to the Maker Protocol in the Maker Documentation Portal for an in-depth explanation of the entire system.

About MakerDAO

MakerDAO is an open-source project on the Ethereum blockchain and a Decentralized Autonomous Organization[1] created in 2014. The project is managed by people around the world who hold its governance token, MKR.

[1] Note that Decentralized Autonomous Organizations, or DAOs, are understood in the Ethereum community as largely social and technical communities centered around a particular mission or project, and does not necessarily imply the existence of traditional corporate forms.

Through a system of scientific governance involving Executive Voting and Governance Polling, MKR holders manage the Maker Protocol and the financial risks of Dai to ensure its stability, transparency, and efficiency. MKR voting weight is proportional to the amount of MKR a voter stakes in the voting contract, DSChief. In other words, the more MKR tokens locked in the contract, the greater the voter's decision-making power.

About the Maker Protocol

The Maker Protocol, built on the Ethereum blockchain,[2] enables users to create currency. Current elements of the Maker Protocol are the Dai stablecoin, Maker Collateral Vaults, Oracles, and Voting. MakerDAO governs the Maker Protocol by deciding on key parameters (e.g., stability fees, collateral types/rates, etc.) through the voting power of MKR holders.

The Maker Protocol, one of the largest decentralized applications (dapps) on the Ethereum blockchain, was the first decentralized finance (DeFi) application to earn significant adoption.

About the Maker Foundation

The Maker Foundation, which is part of the global Maker community, built and launched the Maker Protocol in conjunction with a number of outside partners. It is currently working with the MakerDAO community to bootstrap decentralized governance of the project and drive it toward complete decentralization.

About the Dai Foundation

The Dai Foundation, based in Denmark, is self-governing and independent of the Maker Foundation. It was formed to house the Maker community's key intangible assets, such as trademarks and code copyrights, and it operates solely on the basis of objective and rigid statutes that define its mandate. Its purpose, as noted in the Dai Foundation Trust Deed, is to safeguard what cannot be technologically decentralized in the Maker Protocol.

[2] https://ethereum.org/

Introduction

Beginning in 2015, the MakerDAO project operated with developers around the globe working together on the first iterations of code, architecture, and documentation. In December 2017, the first MakerDAO formal white paper was published, introducing the original Dai (now Sai) Stablecoin System.

The white paper described how anyone could generate Dai using that system by leveraging Ethereum (ETH) as collateral through unique smart contracts known as Collateralized Debt Positions (CDPs). Given that ETH was the only collateral asset accepted by the system, the Dai generated was called Single-Collateral Dai (SCD), or Sai. That white paper also included a plan to upgrade the system to support multiple collateral asset types in addition to ETH. What was then an intention, became a reality in November 2019.

The Dai Stablecoin System, today called the Maker Protocol, now accepts as collateral any Ethereum-based asset that has been approved by MKR holders, who also vote on corresponding Risk Parameters for each collateral asset. Voting is a critical component of the Maker decentralized governance process.

Welcome to **Multi-Collateral Dai (MCD)**.

In MCD We Trust

Blockchain technology provides an unprecedented opportunity to ease the public's growing frustration with—and distrust of—dysfunctional centralized financial systems. By distributing data across a network of computers, the technology allows any group of individuals to embrace transparency rather than central-entity control. The result is an unbiased, transparent, and highly efficient permissionless system—one that can improve current global financial and monetary structures and better serve the public good.

Bitcoin was created with this goal in mind. But, while Bitcoin succeeds as a cryptocurrency on a number of levels, it is not ideal as a medium of exchange because its fixed supply and speculative nature results in volatility, which prevents it from proliferating as mainstream money.

The Dai stablecoin, on the other hand, succeeds where Bitcoin fails precisely because Dai is designed to *minimize price volatility.* A decentralized, unbiased, collateral-backed cryptocurrency that is soft-pegged to the US Dollar, Dai's value is in its stability.

Since the release of Single-Collateral Dai in 2017, user adoption of the stablecoin has risen dramatically, and it has become a building block for decentralized applications that help expand the DeFi (decentralized finance) movement. Dai's success is part of a wider industry movement for stablecoins, which are cryptocurrencies designed to maintain price value and function like money.

For example, in February 2019, JPMorgan became the first bank in the United States to create and test a digital coin that represents 1 USD.[3] As the cryptocurrency industry grows, other banks, financial services companies, and even governments will create stable digital currencies (e.g., Central Bank Digital Currencies), as will large organizations outside of the finance sector. Facebook, for example, announced its plans for Libra, "a stable digital cryptocurrency that will be fully backed by a reserve of real assets,"[4] in June 2019. However, such proposals forfeit the core value proposition of blockchain technology: global adoption of a common infrastructure without a central authority or administrator that may abuse its influence.

An Overview of the Maker Protocol and Its Features

The Maker Protocol

The Maker Protocol is one of the largest dapps on the Ethereum blockchain. Designed by a disparate group of contributors, including developers within

[3] https://www.jpmorgan.com/global/news/digital-coin-payments
[4] https://libra.org/en-US/wp-content/uploads/sites/23/2019/06/LibraWhitePaper_en_US.pdf

the Maker Foundation, its outside partners, and other persons and entities, it is the first decentralized finance (DeFi) application to see significant adoption.

The Maker Protocol is managed by people around the world who hold its governance token, MKR. Through a system of scientific governance involvingExecutive Voting and Governance Polling, MKR holders govern the Protocol and the financial risks of Dai to ensure its stability, transparency, and efficiency. One MKR token locked in a voting contract equals one vote.

The Dai Stablecoin

The Dai stablecoin is a decentralized, unbiased, collateral-backed cryptocurrency soft-pegged to the US Dollar. Dai is held in cryptocurrency wallets or within platforms, and is supported on Ethereum and other popular blockchains.

Dai is easy to generate, access, and use. Users generate Dai by depositing collateral assets into Maker Vaults within the Maker Protocol. This is how Dai is entered into circulation and how users gain access to liquidity. Others obtain Dai by buying it from brokers or exchanges, or simply by receiving it as a means of payment.

Once generated, bought, or received, Dai can be used in the same manner as any other cryptocurrency: it can be sent to others, used as payments for goods and services, and even held as savings through a feature of the Maker Protocol called the Dai Savings Rate (DSR).

Every Dai in circulation is directly backed by excess collateral, meaning that the value of the collateral is higher than the value of the Dai debt, and all Dai transactions are publicly viewable on the Ethereum blockchain.

What Properties of Dai Function Similarly to Money?

Generally, money has four functions:

1. A store of value
2. A medium of exchange

3. A unit of account
4. A standard of deferred payment

Dai has properties and use cases designed to serve these functions.

Dai as a Store of Value

A store of value is an asset that keeps its value without significant depreciation over time. Because Dai is a stablecoin, it is designed to function as a store of value even in a volatile market.

Dai as a Medium of Exchange

A medium of exchange is anything that represents a standard of value and is used to facilitate the sale, purchase, or exchange (trade) of goods or services. The Dai stablecoin is used around the world for all types of transactional purposes.

Dai as a Unit of Account

A unit of account is a standardized measurement of value used to price goods and services (e.g., USD, EUR, YEN). Currently, Dai has a target price of 1USD (1 Dai = 1 USD). While Dai is not used as a standard measurement of value in the off-chain world, it functions as a unit of account within the Maker Protocol and some blockchain dapps, whereby Maker Protocol accounting or pricing of dapp services is in Dai rather than a fiat currency like USD.

Dai as a Standard of Deferred Payment

Dai is used to settle debts within the Maker Protocol (e.g., users use Dai to pay the stability fee and close their Vaults). This benefit separates Dai from other stablecoins.

Collateral Assets

Dai is generated, backed, and kept stable through collateral assets that are deposited into Maker Vaults on the Maker Protocol. A collateral asset is a digital asset that MKR holders have voted to accept into the Protocol.

To generate Dai, the Maker Protocol accepts as collateral any Ethereum-based asset that has been approved by MKR holders. MKR

holders also must also approve specific, corresponding Risk Parameters for each accepted collateral (e.g., more stable assets might get more lenient Risk Parameters, while more risky assets could get stricter Risk Parameters). Detailed information on Risk Parameters is below. These and other decisions of MKR holders are made through the Maker decentralized governance process.

Maker Vaults

All accepted collateral assets can be leveraged to generate Dai in the Maker Protocol through smart contracts called Maker Vaults. Users can access the Maker Protocol and create Vaults through a number of different user interfaces (i.e., network access portals), including Oasis Borrow and various interfaces built by the community. Creating a Vault is not complicated, but generating Dai does create an obligation to repay the Dai, along with a Stability Fee, in order to withdraw the collateral leveraged and locked inside a Vault.

Vaults are inherently non-custodial: Users interact with Vaults and the Maker Protocol directly, and each user has complete and independent control over their deposited collateral as long the value of that collateral doesn't fall below the required minimum level (the Liquidation Ratio, discussed in detail below).

Interacting with a Maker Vault

- **Step 1: Create and Collateralize a Vault**
  A user creates a Vault via the Oasis Borrow portal or a community-created interface, such as Instadapp, Zerion, or MyEtherWallet, by funding it with a specific type and amount of collateral that will be used to generate Dai. Once funded, a Vault is considered collateralized.
- **Step 2: Generate Dai from the Collateralized Vault**
  The Vault owner initiates a transaction, and then confirms it in her unhosted cryptocurrency wallet in order to generate a specific amount of Dai in exchange for keeping her collateral locked in the Vault.
- **Step 3: Pay Down the Debt and the Stability Fee**
  To retrieve a portion or all of the collateral, a Vault owner must pay down or completely pay back the Dai she generated, plus the Stability

Fee that continuously accrues on the Dai outstanding. The Stability Fee can only be paid in Dai.

- **Step 4: Withdraw Collateral**
  With the Dai returned and the Stability Fee paid, the Vault owner can withdraw all or some of her collateral back to her wallet. Once all Dai is completely returned and all collateral is retrieved, the Vault remains empty until the owner chooses to make another deposit.

Importantly, each collateral asset deposited requires its own Vault. So, some users will own multiple Vaults with different types of collateral and levels of collateralization.

Liquidation of Risky Maker Vaults

To ensure there is always enough collateral in the Maker Protocol to cover the value of all outstanding debt (the amount of Dai outstanding valued at the Target Price), any Maker Vault deemed too risky (according to parameters established by Maker Governance) is liquidated through automated Maker Protocol auctions. The Protocol makes the determination after comparing the Liquidation Ratio to the current collateral-to-debt ratio of a Vault. Each Vault type has its own Liquidation Ratio, and each ratio is determined by MKR voters based on the risk profile of the particular collateral asset type.

Maker Protocol Auctions

The auction mechanisms of the Maker Protocol enable the system to liquidate Vaults even when price information for the collateral is unavailable. At the point of liquidation, the Maker Protocol takes the liquidated Vault collateral and subsequently sells it using an internal market-based auction mechanism. This is a **Collateral Auction**.

The Dai received from the Collateral Auction is used to cover the Vault's outstanding obligations, including payment of the Liquidation Penalty fee set by MKR voters for that specific Vault collateral type.

If enough Dai is bid in the Collateral Auction to fully cover the Vault obligations plus the Liquidation Penalty, that auction converts to a **Reverse Collateral Auction** in an attempt to sell as little collateral as possible. Any leftover collateral is returned to the original Vault owner.

If the Collateral Auction does not raise enough Dai to cover the Vault's outstanding obligation, the deficit is converted into Protocol debt. Protocol debt is covered by the Dai in the Maker Buffer. If there is not enough Dai in the Buffer, the Protocol triggers a **Debt Auction**. During a Debt Auction, MKR is minted by the system (increasing the amount of MKR in circulation), and then sold to bidders for Dai.

Dai proceeds from the Collateral Auction go into the Maker Buffer, which serves as a buffer against an increase of MKR overall supply that could result from future uncovered Collateral Auctions and the accrual of the Dai Savings Rate (discussed in detail below).

If Dai proceeds from auctions and Stability Fee payments exceed the Maker Buffer limit (a number set by Maker Governance), they are sold through a **Surplus Auction**. During a Surplus Auction, bidders compete by bidding decreasing amounts of MKR to receive a fixed amount of Dai. Once the Surplus Auction has ended, the Maker Protocol autonomously destroys the MKR collected, thereby reducing the total MKR supply.

Example (Collateral Auction Process):

A large Vault becomes undercollateralized due to market conditions. An Auction Keeper then detects the undercollateralized Vault opportunity and initiates liquidation of the Vault, which kicks off a Collateral Auction for, say, 50 ETH.

Each Auction Keeper has a **bidding model** to assist in winning auctions. A bidding model includes a price at which to bid for the collateral (ETH, in this example). The Auction Keeper uses the token price from its bidding model as the basis for its bids in the first phase of a Collateral Auction, where increasing Dai bids are placed for the set amount of collateral. This amount represents the price of the total Dai wanted from the collateral auction.

Now, let's say the Auction Keeper bids 5,000 Dai for the 50 ETH to meet this amount. The Dai bid is transferred from the Vault Engine to the Collateral Auction contract. With enough Dai in the Collateral Auction contract to cover the system's debt plus the Liquidation Penalty, the first phase of the Collateral Auction is over.

In order to reach the price defined in its bidding model, the Auction Keeper submits a bid in the second phase of the Collateral Auction. In this phase, the objective is to return as much of the collateral to the Vault owner as the market will allow. The bids that the Auction Keepers place are for fixed Dai amounts and decreasing amounts of ETH. For instance, the bidding model of the Keeper in this example seeks a bid price of 125 Dai per ETH, so it offers 5000 Dai for 40 ETH. Additional Dai for this bid is transferred from the Vault Engine to the Collateral Auction contract. After the bid duration limit is reached and the bid expires, the Auction Keeper claims the winning bid and settles the completed Collateral Auction by collecting the won collateral.

Key External Actors

In addition to its smart contract infrastructure, the Maker Protocol involves groups of external actors to maintain operations: Keepers, Oracles, and Global Settlers (Emergency Oracles), and Maker community members. Keepers take advantage of the economic incentives presented by the Protocol; Oracles and Global Settlers are external actors with special permissions in the system assigned to them by MKR voters; and Maker community members are individuals and organizations that provide services.

Keepers

A Keeper is an independent (usually automated) actor that is incentivized by arbitrage opportunities to provide liquidity in various aspects of a decentralized system. In the Maker Protocol, Keepers are market participants that help Dai maintain its Target Price ($1): they sell Dai when the market price is above the Target Price, and buy Dai when the market price is below the Target Price. Keepers participate in Surplus Auctions, Debt Auctions, and Collateral Auctions when Maker Vaults are liquidated.

Price Oracles

The Maker Protocol requires real-time information about the market price of the collateral assets in Maker Vaults in order to know when to trigger Liquidations.

The Protocol derives its internal collateral prices from a decentralized Oracle infrastructure that consists of a broad set of individual nodes called Oracle Feeds. MKR voters choose a set of trusted Feeds to deliver price information to the system through Ethereum transactions. They also control how many Feeds are in the set.

To protect the system from an attacker attempting to gain control of a majority of the Oracles, the Maker Protocol receives price inputs through the Oracle Security Module (OSM), not from the Oracles directly. The OSM, which is a layer of defense between the Oracles and the Protocol, delays a price for one hour, allowing Emergency Oracles or a Maker Governance vote to freeze an Oracle if it is compromised. Decisions regarding Emergency Oracles and the price delay duration are made by MKR holders.

Emergency Oracles

Emergency Oracles are selected by MKR voters and act as a last line of defense against an attack on the governance process or on other Oracles. Emergency Oracles are able to freeze individual Oracles (e.g., ETH and BAT Oracles) to mitigate the risk of a large number of customers trying to withdraw their assets from the Maker Protocol in a short period of time, as they have the authority to unilaterally trigger an Emergency Shutdown.

DAO Teams

DAO teams consist of individuals and service providers, who may be contracted through Maker Governance to provide specific services to MakerDAO. Members of DAO teams are independent market actors and are not employed by the Maker Foundation.

The flexibility of Maker Governance allows the Maker community to adapt the DAO team framework to suit the services needed by the ecosystem based on real-world performance and emerging challenges.

Examples of DAO team member roles are the Governance Facilitator, who supports the communication infrastructure and processes of governance, and Risk Team members, who support Maker Governance with financial risk

research and draft proposals for onboarding new collateral and regulating existing collateral.

While the Maker Foundation has bootstrapped Maker Governance to date, it is anticipated that the DAO will take full control, conduct MKR votes, and fill these varied DAO team roles in the near future.

The Dai Savings Rate

The Dai Savings Rate (DSR) allows any Dai holder to earn savings automatically and natively by locking their Dai into the DSR contract in the Maker Protocol. It can be accessed via the Oasis Save portal or through various gateways into the Maker Protocol. Users aren't required to deposit a minimum amount to earn the DSR, and they can withdraw any or all of their Dai from the DSR contract at any time.

The DSR is a global system parameter that determines the amount Dai holders earn on their savings over time. When the market price of Dai deviates from the Target Price due to changing market dynamics, MKR holders can mitigate the price instability by voting to modify the DSR accordingly:

- If the market price of Dai is above 1 USD, MKR holders can choose to gradually decrease the DSR, which will reduce demand and should reduce the market price of Dai toward the 1 USD Target Price.
- If the market price of Dai is below 1 USD, MKR holders can choose to gradually increase the DSR, which will stimulate demand and should increase the market price of Dai toward the 1 USD Target Price.

Initially, adjustment of the DSR will depend on a weekly process, whereby MKR holders first evaluate and discuss public market data and proprietary data provided by market participants, and then vote on whether an adjustment is necessary or not. The long-term plan includes implementation of the DSR Adjustment Module, an Instant Access Module that directly controls both the DSR and the Base Rate. This module allows for easy adjustment of the DSR (within strict size and frequency boundaries set by MKR holders) by an MKR holder on behalf of the larger group of MKR holders. The motivation behind this plan is to enable nimble responses to

rapidly changing market conditions, and to avoid overuse of the standard governance process ofExecutive Voting and Governance Polling.

Governance of the Maker Protocol

Use of the MKR Token in Maker Governance

The MKR token—the governance token of the Maker Protocol—allows those who hold it to *vote* on changes to the Maker Protocol. Note that anyone, not only MKR holders, can *submit* proposals for an MKR vote.

Any voter-approved modifications to the governance variables of the Protocol will likely not take effect immediately in the future; rather, they could be delayed by as much as 24 hours if voters choose to activate the Governance Security Module (GSM). The delay would give MKR holders the opportunity to protect the system, if necessary, against a malicious governance proposal (e.g., a proposal that alters collateral parameters contrary to established monetary policies or that allows for security mechanisms to be disabled) by triggering a Shutdown.

Polling and Executive Voting

In practice, the Maker Governance process includes proposal polling and Executive Voting. Proposal polling is conducted to establish a rough consensus of community sentiment before any Executive Votes are cast. This helps to ensure that governance decisions are considered throughtfully and reached by consensus prior to the voting process itself. Executive Voting is held to approve (or not) changes to the state of the system. An example of an Executive Vote could be a vote to ratify Risk Parameters for a newly accepted collateral type.

At a technical level, smart contracts manage each type of vote. A Proposal Contract is a smart contract with one or more valid governance actions programmed into it. It can only be executed once. When executed, it immediately applies its changes to the internal governance variables of the Maker Protocol. After execution, the Proposal Contract cannot be reused.

Any Ethereum Address can deploy valid Proposal Contracts. MKR token holders can then cast approval votes for the proposal that they want to elect

as the Active Proposal. The Ethereum address that has the highest number of approval votes is elected as the Active Proposal. The Active Proposal is empowered to gain administrative access to the internal governance variables of the Maker Protocol, and then modify them.

The MKR Token's Role in Recapitalization
In addition to its role in Maker Governance, the MKR token has a complementary role as the recapitalization resource of the Maker Protocol. If the system debt exceeds the surplus, the MKR token supply may increase through a Debt Auction (see above) to recapitalize the system. This risk inclines MKR holders to align and responsibly govern the Maker ecosystem to avoid excessive risk-taking.

MKR Holder Responsibilities
MKR holders can vote to do the following:

- Add a new collateral asset type with a unique set of Risk Parameters.
- Change the Risk Parameters of one or more existing collateral asset types, or add new Risk Parameters to one or more existing collateral asset types.
- Modify the Dai Savings Rate.
- Choose the set of Oracle Feeds.
- Choose the set of Emergency Oracles.
- Trigger Emergency Shutdown.
- Upgrade the system.

MKR holders can also allocate funds from the Maker Buffer to pay for various infrastructure needs and services, including Oracle infrastructure and collateral risk management research. The funds in the Maker Buffer are revenues from Stability Fees, Liquidation Fees, and other income streams.

The governance mechanism of the Maker Protocol is designed to be as flexible as possible, and upgradeable. Should the system mature under the guidance of the community, more advanced forms of Proposal Contracts could, in theory, be used, including Proposal Contracts that are bundled. For example, one proposal contract may contain both an adjustment of a Stability Fee and an adjustment of the DSR. Nonetheless, those revisions will remain for MKR holders to decide.

Risk Parameters Controlled by Maker Governance

Each Maker Vault type (e.g., ETH Vault and BAT Vault) has its own unique set of Risk Parameters that enforce usage. The parameters are determined based on the risk profile of the collateral, and are directly controlled by MKR holders through voting.

The Key Risk Parameters for Maker Vaults are:

- **Debt Ceiling**: A Debt Ceiling is the maximum amount of debt that can be created by a single collateral type. Maker Governance assigns every collateral type a Debt Ceiling, which is used to ensure sufficient diversification of the Maker Protocol collateral portfolio. Once a collateral type has reached its Debt Ceiling, it becomes impossible to create more debt unless some existing users pay back all or a portion of their Vault debt.
- **Stability Fee**: The Stability Fee is an annual percentage yield calculated on top of how much Dai has been generated against a Vault's collateral. The fee is paid in Dai only, and then sent into the Maker Buffer.
- **Liquidation Ratio**: A low Liquidation Ratio means Maker Governance expects low price volatility of the collateral; a high Liquidation Ratio means high volatility is expected.
- **Liquidation Penalty**: The Liquidation Penalty is a fee added to a Vault's total outstanding generated Dai when a Liquidation occurs. The Liquidation Penalty is used to encourage Vault owners to keep appropriate collateral levels.
- **Collateral Auction Duration**: The maximum duration of Collateral auctions is specific to Maker Vaults. Debt and Surplus auction durations are global system parameters.
- **Auction Bid Duration:** Amount of time before an individual bid expires and closes the auction.
- **Auction Step Size**: This Risk Parameter exists to incentivize early bidders in auctions, and prevent abuse by bidding a tiny amount above an existing bid.

Risk and Mitigation Responsibilities of Governance

The successful operation of the Maker Protocol depends on Maker Governance taking necessary steps to mitigate risks. Some of those risks are identified below, each followed by a mitigation plan.

A malicious attack on the smart contract infrastructure by a bad actor. One of the greatest risks to the Maker Protocol is a malicious actor—a programmer, for example, who discovers a vulnerability in the deployed smart contracts, and then uses it to break the Protocol or steal from it.

In the worst-case scenario, all decentralized digital assets held as collateral in the Protocol are stolen, and recovery is impossible.

Mitigation: The Maker Foundation's highest priority is the security of the Maker Protocol, and the strongest defense of the Protocol is Formal Verification. The Dai codebase was the first codebase of a decentralized application to be formally verified.

In addition to formal system verification, contracted security audits by the best security organizations in the blockchain industry, third-party (independent) audits, and bug bounties are part of the Foundation's security roadmap. To review the formal verification report and various Maker Protocol audits, visit Maker's Multi-Collateral Dai Security Github repository.

These security measures provide a strong defense system; however, they are not infallible. Even with formal verification, the mathematical modeling of intended behaviors may be incorrect, or the assumptions behind the intended behavior itself may be incorrect.

A black swan event
A black swan event is a rare and critical surprise attack on a system. For the Maker Protocol, examples of a black swan event include:

- An attack on the collateral types that back Dai.
- A large, unexpected price decrease of one or more collateral types.
- A highly coordinated Oracle attack.

- A malicious Maker Governance proposal.

Please note that this list of potential "black swans" is not exhaustive and not intended to capture the extent of such possibilities.

Mitigation: While no one solution is failsafe, the careful design of the Maker Protocol (the Liquidation Ratio, Debt Ceilings, the Governance Security Module, the Oracle Security Module, Emergency Shutdown, etc.) in conjunction with good governance (e.g., swift reaction in a crisis, thoughtful risk parameters, etc.) help to prevent or mitigate potentially severe consequences of an attack.

Unforeseen pricing errors and market irrationality
Oracle price feed problems or irrational market dynamics that cause variations in the price of Dai for an extended period of time can occur. If confidence in the system is lost, rate adjustments or even MKR dilution could reach extreme levels and still not bring enough liquidity and stability to the market.

Mitigation: Maker Governance incentivizes a sufficiently large capital pool to act as Keepers of the market in order to maximize rationality and market efficiency, and allow the Dai supply to grow at a steady pace without major market shocks. As a last resort, Emergency Shutdown can be triggered to release collateral to Dai holders, with their Dai claims valued at the Target Price.

User Abandonment for Less Complicated Solutions
The Maker Protocol is a complex decentralized system. As a result of its complexity, there is a risk that inexperienced cryptocurrency users will abandon the Protocol in favor of systems that may be easier to use and understand.

Mitigation: While Dai is easy to generate and use for most crypto enthusiasts and the Keepers that use it for margin trading, newcomers might find the Protocol difficult to understand and navigate. Although Dai is designed in such a way that users need not comprehend the underlying mechanics of the Maker Protocol in order to benefit from it, the documentation and numerous

resources consistently provided by the Maker community and the Maker Foundation help to ensure onboarding is as uncomplicated as possible.

Dissolution of The Maker Foundation
The Maker Foundation currently plays a role, along with independent actors, in maintaining the Maker Protocol and expanding its usage worldwide, while facilitating Governance. However, the Maker Foundation plans to dissolve once MakerDAO can manage Governance completely on its own. Should MakerDAO fail to sufficiently take the reins upon the Maker Foundation's dissolution, the future health of the Maker Protocol could be at risk.

Mitigation: MKR holders are incentivized to prepare for the Foundation's dissolution after it completes "gradual decentralization" of the project. Moreover, successful management of the system should result in sufficient funds for governance to allocate to the continued maintenance and improvement of the Maker Protocol.

General Issues with Experimental Technology

Users of the Maker Protocol (including but not limited to Dai and MKR holders) understand and accept that the software, technology, and technical concepts and theories applicable to the Maker Protocol are still unproven and there is no warranty that the technology will be uninterrupted or error-free. There is an inherent risk that the technology could contain weaknesses, vulnerabilities, or bugs causing, among other things, the complete failure of the Maker Protocol and/or its component parts.

Mitigation: See “A malicious attack on the smart contract infrastructure by a bad actor” above. The Mitigation section there explains the technical auditing in place to ensure the Maker Protocol functions as intended.

Price Stability Mechanisms

The Dai Target Price

The Dai Target Price is used to determine the value of collateral assets Dai holders receive in the case of an Emergency Shutdown. The Target Price for Dai is 1 USD, translating to a 1:1 USD soft peg.

Emergency Shutdown

Emergency Shutdown (or, simply, Shutdown) serves two main purposes. First, it is used during emergencies as a last-resort mechanism to protect the Maker Protocol against attacks on its infrastructure and directly enforce the Dai Target Price. Emergencies could include malicious governance actions, hacking, security breaches, and long-term market irrationality. Second, Shutdown is used to facilitate a Maker Protocol system upgrade. The Shutdown process can only be controlled by Maker Governance.

MKR voters are also able to instantly trigger an Emergency Shutdown by depositing MKR into the Emergency Shutdown Module (ESM), if enough MKR voters believe it is necessary. This prevents the Governance Security Module (if active) from delaying Shutdown proposals before they are executed. With Emergency Shutdown, the moment a quorum is reached, the Shutdown takes effect with no delay.

There are three phases of Emergency Shutdown:

1. **The Maker Protocol shuts down; Vault owners withdraw assets.**
   When initiated, Shutdown prevents further Vault creation and manipulation of existing Vaults, and freezes the Price Feeds. The frozen feeds ensure that all users are able to withdraw the net value of assets to which they are entitled. Effectively, it allows Maker Vault owners to immediately withdraw the collateral in their Vault that is not actively backing debt.
2. **Post-Emergency Shutdown auction processing**
   After Shutdown is triggered, Collateral Auctions begin and must be completed within a specific amount of time. That time period is determined by Maker Governance to be slightly longer than the duration of the longest Collateral Auction. This guarantees that no auctions are outstanding at the end of the auction processing period.
3. **Dai holders claim their remaining collateral**
   At the end of the auction processing period, Dai holders use their Dai to claim collateral directly at a fixed rate that corresponds to the calculated value of their assets based on the Dai Target Price. For example, if the ETH/USD Price Ratio is 200, and a user holds 1000 Dai at the Target Price of 1 USD when Emergency Shutdown is activated,

The user will be able to claim exactly 5 ETH from the Maker Protocol after the auction processing period. There is no time limit for when a final claim can be made. Dai holders will get a proportional claim to each collateral type that exists in the collateral portfolio. Note that Dai holders could be at risk of a haircut, whereby they do not receive the full value of their Dai holdings at the Target Price of 1 USD per Dai. This is due to risks related to declines in collateral value and to Vault owners having the right to retrieve their excess collateral before Dai holders may claim the remaining collateral. For more detailed information on Emergency Shutdown, including the claim priorities that would occur as a result, see the published community documentation.

The Future of the Maker Protocol: Increased Adoption and Full Decentralization

Addressable Market

A cryptocurrency with price stability serves as an important medium of exchange for many decentralized applications. As such, the potential market for Dai is at least as large as the entire decentralized blockchain industry. But the promise of Dai extends well beyond that into other industries.

The following is a non-exhaustive list of current and immediate markets for the Dai stablecoin:

- **Working capital, hedging, and collateralized leverage**. Maker Vaults allow for permissionless trading by users, who can use the Dai generated against Vault collateral for working capital. To date, there have been numerous instances where Vault owners use their Dai to buy additional ETH (same asset as their collateral), thereby creating a leveraged but fully collateralized position.
- **Merchant receipts, cross-border transactions, and remittances**. Foreign exchange volatility mitigation and a lack of intermediaries mean the transaction costs of international trade are significantly reduced when using Dai.
- **Charities and NGOs** when using transparent distributed ledger technology.

- **Gaming.** For blockchain game developers, Dai is the currency of choice. With Dai, game developers integrate not only a currency, but also an entire economy. The composability of Dai allows games to create new player behavior schemes based around decentralized finance.
- **Prediction markets**. Using a volatile cryptocurrency when making an unrelated prediction only increases one's risk when placing the bet. Long-term bets become especially infeasible if the bettor must also gamble on the future price of the volatile asset used to place the bet. That said, the Dai stablecoin would be a natural choice for use in prediction markets.

Asset Expansion

Should MKR holders approve new assets as collateral, those assets will be subject to the same risk requirements, parameters, and safety measures as Dai (e.g., Liquidation Ratios, Stability Fees, Savings Rates, Debt Ceilings, etc.).

Evolving Oracles

MakerDAO was the first project to run reliable Oracles on the Ethereum blockchain. As a result, many decentralized applications use MakerDAO Oracles to ensure the security of their systems and to provide up-to-date price data in a robust manner. This confidence in MakerDAO and the Maker Protocol means that Maker Governance can expand the core Oracle infrastructure service to better suit the needs of decentralized applications.

Conclusion

The Maker Protocol allows users to generate Dai, a stable store of value that lives entirely on the blockchain. Dai is a decentralized stablecoin that is not issued or administered by any centralized actor or trusted intermediary or counterparty. It is unbiased and borderless —available to anyone, anywhere.

All Dai is backed by a surplus of collateral that has been individually escrowed into audited and publicly viewable Ethereum smart contracts.

Anyone with an internet connection can monitor the health of the system anytime at daistats.com.

With hundreds of partnerships and one of the strongest developer communities in the cryptocurrency space, MakerDAO has become the engine of the decentralized finance (DeFi) movement. Maker is unlocking the power of the blockchain to deliver on the promise of economic empowerment today.

For more information, visit the MakerDAO website.

APPENDIX

Dai Use-Case Benefits and Examples

The Maker Protocol can be used by anyone, anywhere, without any restrictions or personal-information requirements. Below are a few examples of how Dai is used around the world:

Dai Offers Financial Independence to All

According to the World Bank's Global Findex Database 2017, about 1.7 billion adults around the world are unbanked.[5] In the US alone, according to a 2017 survey by the FDIC, around 32 million American households are either unbanked or underbanked,[6] meaning that they either have no bank account at all or they regularly use alternatives to traditional banking (e.g., payday or pawn shop loans) to manage their finances. Dai can empower every one of those people; all they need is access to the internet.

As the world's first unbiased stablecoin, Dai allows anyone to achieve financial independence, regardless of their location or circumstances. For example, in Latin America, Dai has provided an opportunity for individuals and families to hedge against the devaluation of the Argentine peso[7] and the Venezuelan bolívar. On the islands of Vanuatu in the South Pacific, where residents pay very high money transfer fees, Oxfam International, a U.K.-based non-profit; Australian startup, Sempo; and Ethereum startup

[5] https://globalfindex.worldbank.org/
[6] https://www.fdic.gov/householdsurvey/
[7] https://slideslive.com/38920018/living-on-defi-how-i-survive-argentinas-50-inflation

ConsenSys have successfully piloted a cash-assistance program through which 200 residents on the island of Efate were each given 50 Dai to pay a local network of vendors.[8]

Self-Sovereign Money Generation

Oasis Borrow allows users to access the Maker Protocol and generate Dai by locking their collateral in a Maker Vault. Notably, users do not need to access any third-party intermediary to generate Dai. Vaults offer individuals and businesses opportunities to create liquidity on their assets simply, quickly, and at relatively low cost.

Savings Earned Automatically

Dai holders everywhere can better power their journeys to financial inclusion by taking advantage of the Dai Savings Rate, which, as detailed earlier, builds on the value of Dai by allowing users to earn on the Dai they hold and protect their savings from inflation.

For example, if Bob has100,000 Dai locked in the DSR contract, and the DSR set by Maker Governance is 6% per year, Bob will earn savings of 6,000 Dai over 12 months. Additionally, because exchanges and blockchain projects can integrate the DSR into their own platforms, it presents new opportunities for cryptocurrency traders, entrepreneurs, and established businesses to increase their Dai savings and Dai operating capital. Due to this attractive mechanism, Market Makers, for example, may choose to hold their idle inventory in Dai and lock it in the DSR.

Fast, Low-cost Remittances

Cross-border remittances, whether for the purchase of goods or services or to simply send money to family and friends, can mean high service and transfer fees, long delivery timelines, and frustrating exchange issues due to inflation. The Dai stablecoin is used around the world as a medium of exchange because people have confidence in its value and efficiency.

Remittance users benefit from Dai in the following ways:

[8] https://www.coindesk.com/oxfam-trials-delivery-of-disaster-relief-using-ethereum-stablecoin-dai

- **Low-cost domestic and international transfers.** Dai provides immediate cost savings, as low gas fees replace high bank and wire service fees. Low cost allows for more frequent transactions.
- **Anytime service.** Dai doesn't rely on bank-like hours of operation. The Maker Protocol can be accessed 24/7/365.
- **Convenient on/off ramps.** Users can take advantage of the many fiat on and off ramps that exchange fiat currencies to Dai. These options allow users to bridge the gap between the fiat and cryptocurrency world, and easily cash out Dai holdings in their local currencies.
- **Increased security and confidence.** The blockchain offers high levels of security and consumer trust.

Stability in Volatile Markets

As noted above, Dai is both a readily accessible store of value and a powerful medium of exchange. As such, it can help protect traders from volatility. For example, it provides traders with a simple way to maneuver between positions smoothly and remain active in the market without having to cash out and repeat an on-ramp/off-ramp cycle.

Dai as an Ecosystem Driver and DeFi Builder

As more and more users become aware of Dai's value as a stablecoin, more developers are integrating it into the dapps they build on the Ethereum blockchain. As such, Dai is helping to power a more robust ecosystem. In short, Dai allows dapp developers to offer a stable method of exchange to their users who would rather not buy and sell goods and services using speculative assets.

Additionally, because Dai can be used to pay for gas in the Ethereum ecosystem, by creating DeFi dapps that accept Dai instead of ETH, developers offer users a smoother onboarding experience and a better overall experience.

Glossaries

- MakerDAO Glossary of Terms

- Maker Protocol Glossary (terms, variables, functions, and more)

System and Community Resources

- MakerDAO on GitHub
- MakerDAO Documentation
- MakerDAO.com
- The MakerDAO Blog
- The MakerDAO Forum
- The MakerDAO Chat
- MakerDAO on Reddit
- MakerDAO on Twitter

EXHIBIT C

Search (Press "/" to focus)

Home > Learn > Oracles > **Security**

Security

How the Maker Protocol handles the security of oracles

The oracle system for the Maker Protocol uses decentralized reporting to defend against fraudulent price data.

There are multiple organizations and individuals who report price-data, they are called `Feeds` .

At the launch of Multi-Collateral Dai, oracles received data from a total of 20 Feeds which consisted of 15 individuals and five public organizations.

Oracles use the median of the reported prices for each asset as the reference price. Using a median instead of an average makes it harder to manipulate the reference price since control over half of the data providers is needed in order for a fraudulent price to be pushed through.

Additionally, using a median naturally filters out irregular price data.

Feeds

Feeds are a mix of pseudonymous individuals and public organizations. There are two types of Feeds; Dark Feeds run by anonymous individuals, and Light Feeds run by public organizations. Individuals consist of people internal to Maker, influential people in the greater crypto community, as well as some community members.

Organizations involved in being Feeds at the launch of MCD can be found in the **Feeds list.** ↗

no formal way for Feeds to be added to the Maker Protocol.

As of October 7th, 2019, the **Oracle Team** ↗ was mandated, granting the Interim Oracle Team the responsibility of being the intermediary between the Feeds and governance. In the coming months, the process of becoming a Feed will become more clear.

Oracle Security Module

The **Oracle Security Module(OSM)** ↗ safeguards the process by delaying price-feed data for one hour.

Oracle Security Modules (OSMs) delay the publishing of new reference prices for a predefined set of time. This parameter is called the Oracle Security Module Delay and was set to be one hour at the launch of MCD.

This allows **MKR token** holders and other stakeholders the time to react to bugs or attacks on the Oracles. An OSM is active on each Oracle in the Maker Protocol.

This allows MKR governors and other stakeholders the time to identify bugs or attacks on the price-feed system. An OSM is active on all Oracles in the Maker Protocol.

Medianizer

A Medianizer is a type of smart-contract in the Maker Protocol's Oracle system that collects price-data from Feeds and calculates a reference price by calculating a median.

The Medianizer maintains a white-list of Feeds that can be controlled by MakerDAO governance. Every time a new set of price updates is received, the reference price is recalculated and queued into the Oracle Security Module which publishes the price after a delay period.

Each time a relayer pushes a new set of medians to a Medianizer, it recalculates the median and updates the queued reference price in the Oracle Security Module(OSM). The OSM applies a time-delay and acts as the final publisher of the reference price.

For technical documentation, please visit the **Medianizer** ↗ section of our **Documentation Portal.** ↗

information on how a Secure Scuttlebutt Network works visit this **informative page** ↗ on **scuttlebot.io.** ↗

Search (Press "/" to focus)

EXHIBIT D

Maker Docs

MakerDAO Documentation

The Maker Protocol is the platform through which anyone, anywhere can generate the Dai stablecoin against crypto collateral assets. Learn how it works.

Introduction

MakerDAO is a decentralized organization dedicated to bringing stability to the cryptocurrency economy. The Maker Protocol employs a two-token system. The first being, Dai, a collateral-backed stablecoin that offers stability. The Maker Foundation and the MakerDAO community believe that a decentralized stablecoin is required to have any business or individual realize the advantages of digital money. Second, there is MKR, a governance token that is used by stakeholders to maintain the system and manage Dai. MKR token holders are the decision-makers of the Maker Protocol, supported by the larger public community and various other external parties.

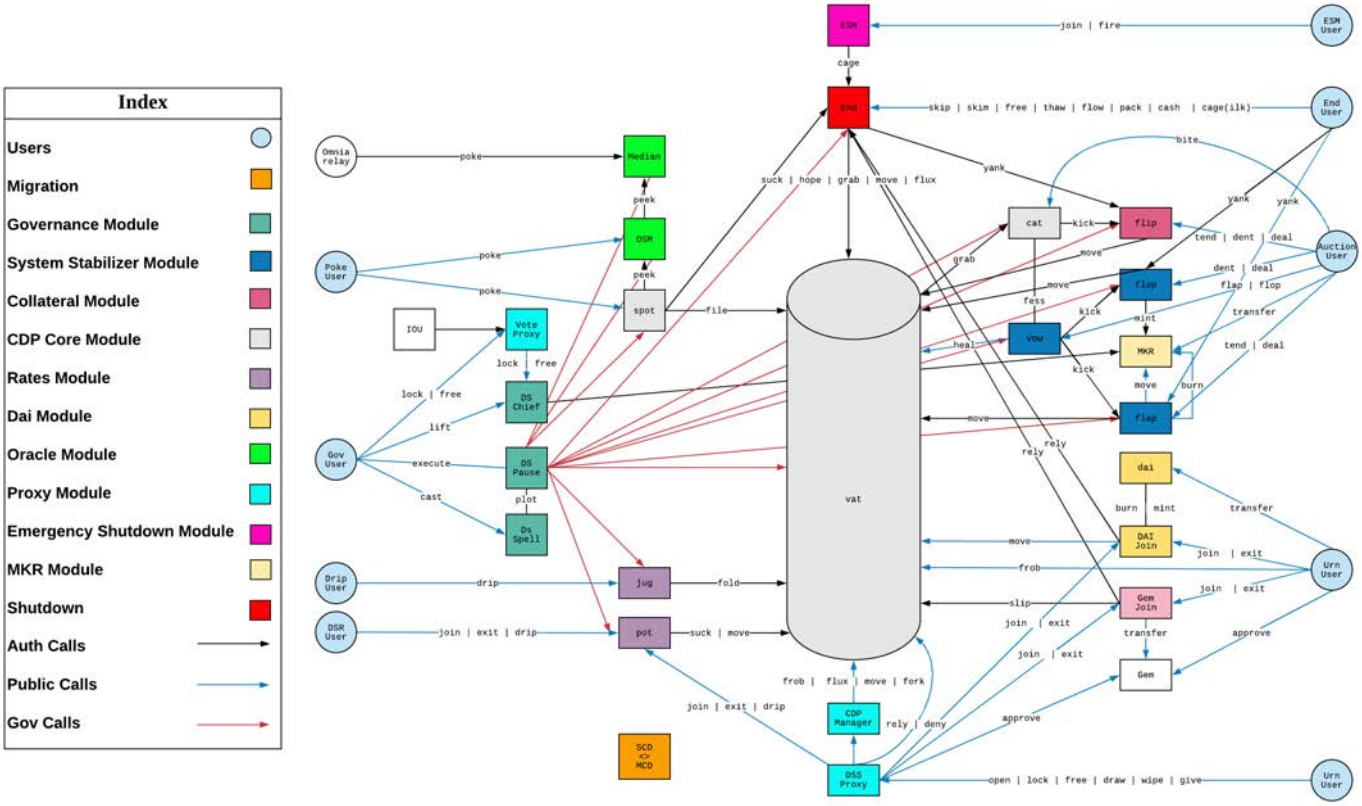
Maker is unlocking the power of decentralized finance for everyone by creating an inclusive platform for economic empowerment; enabling everyone with equal access to the global financial marketplace.

With the new version of the **Maker Protocol**, Multi Collateral Dai (MCD), being released and live on the main Ethereum network, we wanted to go over a few of the changes and features that it comes with. The biggest change to the Maker Protocol is that it now accepts any Ethereum-based asset as collateral to generate Dai given that it has been approved by MKR holders and has been given specific, corresponding Risk Parameters through the Maker decentralized governance process.

Additionally, there are a few other newly introduced features that come with the MCD upgrade. **These new features include:**

- New Dai token ($DAI)
- Support for multiple Vault collateral types (Launching with ETH and BAT)
  - To open a Vault, head to **Oasis Borrow**
- Dai Savings Rate (DSR)
  - To use the DSR, head to **Oasis Save**
- More robust peg ensuring mechanisms (MKR acting as backstop)
- Stability fees paid every block, rather than on Dai repayment
- New Maker Terminology
- MKR and governance remains the same
- Oasis Trade

The Maker Protocol Smart Contract Modules System



The Maker Protocol System Diagram

Search (Press "/" to focus)

English

No Available Languages
Need another language?
Join translation team

English

No Available Languages
Need another language? Join translation team

These are legacy guides and will not be maintened. You may be looking for the page on **Oracles**

Oracles

What is an Oracle?

An Oracle makes both off-chain and on-chain data available for use in smart-contracts. In the Maker Protocol, Oracles enable the use of price data of various assets to determine a number of important things like when to **Liquidate** a **Vault** or how much Dai a given Vault can generate. MakerDAO Oracles receive data from a number of independent Feeds that consist of individuals and organizations. Each Oracle corresponds to a single asset and its reference price.

How does the Oracle system work?

In the Maker Protocol, each collateral type has a corresponding Oracle that publishes a reference price that the system uses. Each

Each Feed uses a tool called **Setzer** ↗ which pulls the median price from a set of exchanges and then pushes it to a **Secure Scuttlebutt Network** ↗ that has relayers reading from it. Relayers aggregate the price data and send a transaction to the Medianizer. The Medianizer then takes the median of the **mul**tiple reported medians and publishes it as a queued reference price. This price is then delayed by the Oracle Security Module before it is finally used by the system.

Feeds may configure Setzer to pull from any exchanges of their choosing. Relayers are able to configure parameters around when to push price data to the Medianizer. Only MKR **governance** can configure or change the Medianizer and Oracle Security Module.

How is the Oracle system made secure?

To defend against fraudulent price-data, the reporting is decentralized; there are **mul**tiple organizations and individuals who report price-data, they are called Feeds. At the launch of **Mul**ti-Collateral Dai, Oracles received data from a total of 20 Feeds which consisted of 15 individuals and five public organizations. The Oracles use the median of the reported prices for each asset as the reference price. Using a median instead of an average makes it harder to manipulate the reference price since control over half of the data providers is needed in order for a fraudulent price to be pushed through. Additionally, using a median naturally filters out irregular price data.

In addition to this, the **Oracle Security Module(OSM)** ↗ safeguards the process by delaying price-feed data for one hour. This allows MKR governors and other stakeholders the time to identify bugs or attacks on the price-feed system. An OSM is active on all Oracles in the Maker Protocol.

Search (Press "/" to focus)

Oracle Security Modules(OSMs) ↗ delay the publishing of new reference prices for a predefined set of time. This parameter is called the `Oracle Security Module Delay` and was set to be one hour at the launch of MCD. This allows MKR token holders and other stakeholders the time to react to bugs or attacks on the Oracles. An OSM is active on each Oracle in the Maker Protocol.

Can MakerDAO governance change the time of the Oracle Security Module's delay?

Yes. This parameter is called the `Oracle Security Module Delay` and can be adjusted by MKR token holders.

What is a Medianizer?

A **Medianizer** ↗ is a type of smart-contract in the Maker Protocol's Oracle system that collects price-data from Feeds and calculates a reference price by calculating a median. The Medianizer maintains a white-list of Feeds that can be controlled by MakerDAO **governance.** Every time a new set of price updates is received, the reference price is recalculated and queued into the Oracle Security Module which publishes the price after a delay period.

How often does the Medianizer publish an updated reference price?

What is a Secure Scuttlebutt Network?

Secure Scuttlebutt is a database protocol for unforgeable append-only message feeds. For more information on how a Secure Scuttlebutt Network works visit this **informative page** ↗ on **scuttlebot.io** ↗.

Why are Oracles an attack target for malicious actors?

In the Maker Protocol, if the reference price for an asset was determined by a single party, then they could fraudulently report an incorrect price and cause severe issues. For instance, if the price of ETH was reported to be fraudulently low, say $0.01, then every single ETH **Vault** in the system would be **Liquidated**. On the other hand, if the price of ETH was reported to be artificially high, say $1,000,000.00, then any ETH Vault owner would be able to issue an excessive amount of Dai since the system thinks there is more Collateral value than there actually is. Oracle attacks can be very profitable for a successful attacker and can be very disruptive to MakerDAO and its users.

Who are the Feeds?

The Feeds are a mix of pseudonymous individuals and public organizations. Individuals consist of people internal to Maker, influential people in the greater crypto community, as well as some community members. The organizations involved in being Feeds at the launch of MCD can be found in the ratified **DeFi Feeds proposal** ↗.

55

pseudonymous?

From their onset, the individuals running Feeds have been pseudonymous out of necessity, to protect the individuals from the risk of extortion and blackmail. Pseudonymous Feeds also have the benefit of making it harder to coordinate an Oracle attack since the Feeds don't know who the others are. Organizations running Feeds, however, are different. Organizations are much more resilient against coercion, have the resources to combat malicious actors, and have their reputations at stake. This makes them much better equipped to be Feeds with public identities. A hybrid model is optimal, one that incorporates the benefits of both individual and organizational Feeds.

What is the process for becoming a Feed?

All new Feeds go through MakerDAO's **governance** in order to be added in. There is currently no formal way for Feeds to be added to the Maker Protocol. As of October 07th, 2019, the ratified **Oracle Team Mandate** ↗ grants the Interim Oracle Team the responsibility of being the intermediary between the Feeds and governance. In the coming months, the process of becoming a Feed will become more clear.

Is the Oracle system resistant to Sybil attacks?

To quote from Wikipedia's page on **Sybil Attacks** ↗, "In a Sybil attack, the attacker subverts the reputation system of a peer-to-peer network by creating a large number of pseudonymous identities, using them to gain a disproportionately large influence. A reputation system's vulnerability to a Sybil attack depends on how cheaply identities can be generated, the degree to which the

The short answer is yes, the Oracle system is resistant to Sybil attacks because of the existence of a whitelist for Feeds. It's not simple to become a Feed, they need to be approved by MKR **governance.** Therefore, an attacker cannot gain a majority influence by creating many pseudo-feeds.

What happens if there is a flash crash on an exchange?

Since the reference price published by the Oracles is a median of the median prices that are reported by at least 20 different Feeds, outliers are automatically filtered out. In practice, this means if a single exchange experiences a flash crash the set of prices will look something like this:

```
[0.70, 104.80, 104.82, **104.88**, 104.90, 105.02, 10
```

The median of this set still reflects the real market price of the asset. Flash crashes on single exchanges do not affect the published reference price.

Where can I find more technical information about Oracles?

Visit our **Documentation Portal** ↗ for all technical documentation of the Maker Protocol. Technical documentation of Oracles can be found in the **Oracle Module** ↗ section of our Documentation Portal.

Search (Press "/" to focus)

Resources

Whitepaper

FAQs

Privacy Policy

Brand Assets

Feeds

Service Status

Products

Oasis

Migrate

Ecosystem

Governance

Developers

Documentation

Dai.js

Developer Guides

Foundation

Contact

Blog

EXHIBIT E

Maker Docs

Median - Detailed Documentation

The Maker Protocol's trusted reference price

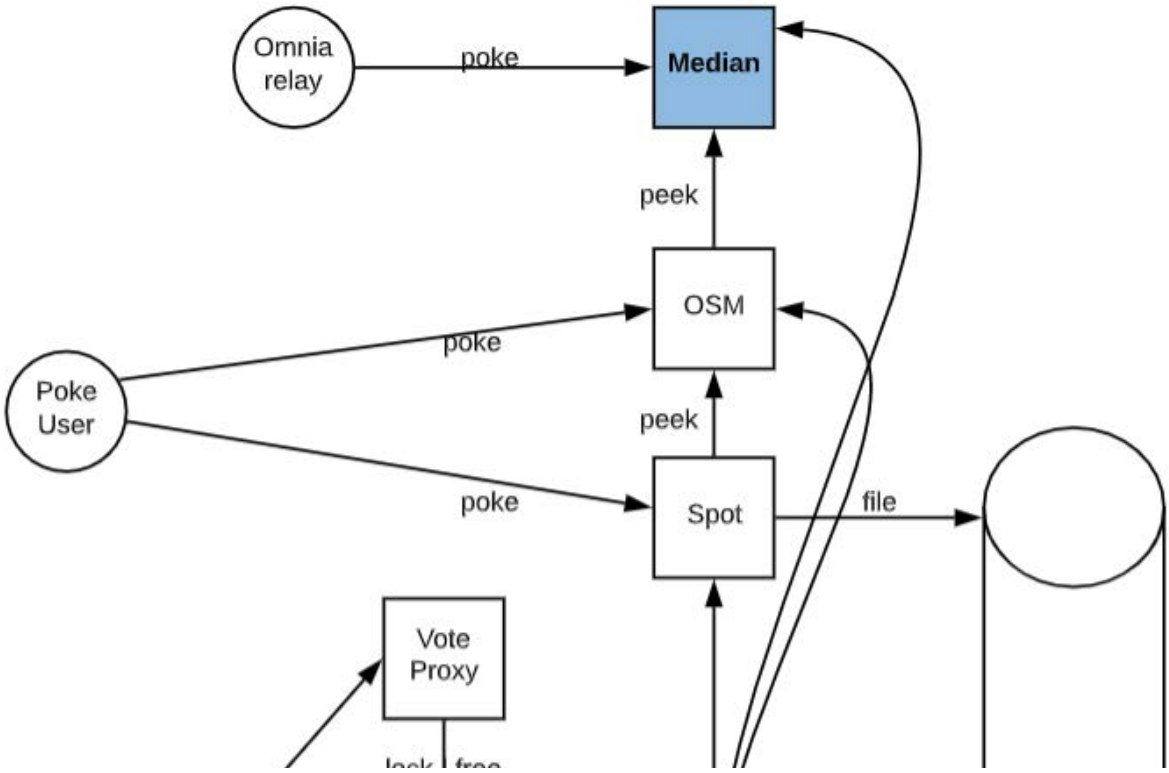
- **Contract Name:** median.sol
- **Type/Category:** Oracles Module
- **Associated MCD System Diagram**
- **Contract Source**

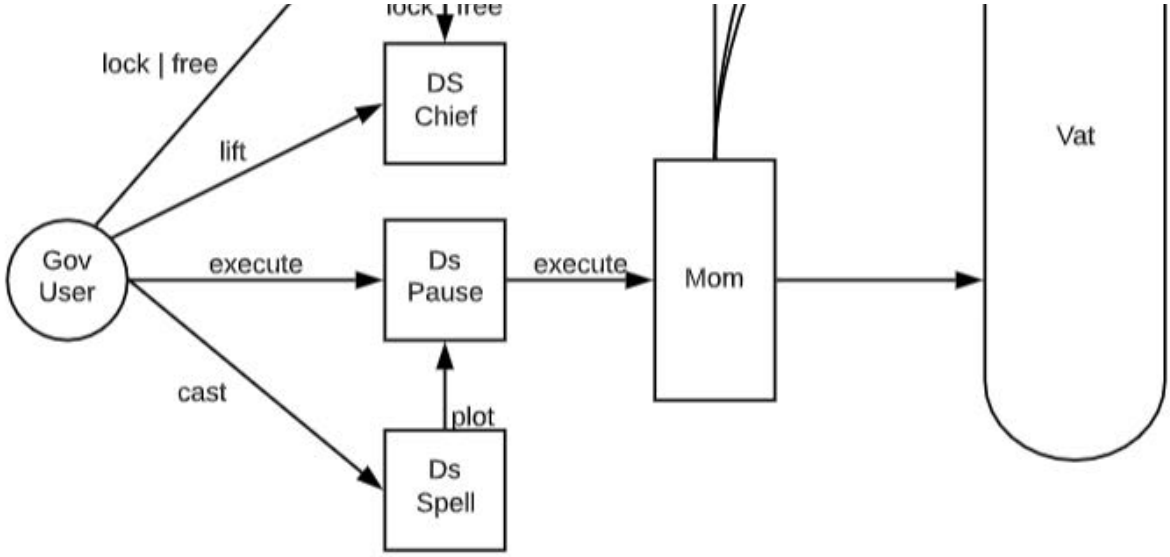
---

1. Introduction (Summary)

The median provides Maker's trusted reference price. In short, it works by maintaining a whitelist of price feed contracts which are authorized to post price updates. Every time a new list of prices is received, the median of these is computed and used to update the stored value. The median has permissioning logic which is what enables the addition and removal of whitelisted price feed addresses that are controlled via governance. The permissioning logic allows governance to set other parameters that control the Median's behavior—for example, the `bar` parameter is the minimum number of prices necessary to accept a new median value.

A High-level overview diagram of the components that involve and interact with the median:





Note: All arrows without labels are governance calls.

---

2. Contract Details

Median (Glossary)

Key Functionalities (as defined in the smart contract)

- `read` - Gets a non-zero price or fails.
- `peek` - Gets the price and validity.
- `poke` - Updates price from whitelisted providers.
- `lift` - Adds an address to the writers whitelist.
- `drop` - Removes an address from the writers whitelist.
- `setBar` - Sets the `bar`.
- `kiss` - Adds an address to the reader's whitelist.
- `diss` - Removes an address from the readers whitelist.

Note: `read` returns the `value` or fails if it's invalid & `peek` gives back the `value` and if the `value` is valid or not.

Other

- `wards(usr: address)` - Auth mechanisms.
- `orcl(usr: address)` - `val` writers whitelist / signers of the prices (whitelisted via governance / the authorized parties).
- `bud(usr: address)` - `val` readers whitelist.
- `val` - the price (private) must be read with `read()` or `peek()`

- `val` - the price (private) must be read with `read()` or `peek()`
- `age` - the Block timestamp of last price `val` update.
- `wat` - the price oracles type (ex: ETHUSD) / tells us what the type of asset is.
- `bar` - the Minimum writers quorum for `poke` / min number of valid messages you need to have to update the price.

---

3. Key Mechanisms & Concepts

As mentioned above, the `median` is the smart contract that provides Maker's trusted reference price. Authorization **(auth)** is a key component included in the mechanism of this contract and its interactions. For example, the price (`val`) is intentionally kept not public because the intention is to only read it from the two functions `read` and `peek` , which are whitelisted. This means that you need to be authorized, which is completed through the `bud` . The `bud` is modified to get whitelisted authorities to read it on-chain (permissioned), whereas, everything of off-chain is public.

The `poke` method is not under any kind of `auth` . This means that anybody can call it. This was designed for the purpose of getting Keepers to call this function and interact with Auctions. The only way to modify its state is if you call it and send it valid data. For example, let's say this oracle needs 15 different sources. This means that we would need it to send 15 different signatures. It will then proceed to go through each of them and validate that whoever sent the the data has been `auth` 'd to do so. In the case of it being an authorized oracle, it will check if it signed the message with a timestamp that is greater than the last one. This is done for the purpose of ensuring that it is not a stale message. The next step is to check for order values, this requires that you send everything in an array that is formatted in ascending order. If not sent in the correct order (ascending), the median is not calculated correctly. This is because if you assume the prices are ordered, it would just grab the middle value which may not be sufficient or work. In order to check for uniqueness, we have implemented the use of a `bloom` filter. In short, a bloom filter is a data structure designed to tell us, rapidly and memory-efficiently, whether an element is present in a set. This use of the bloom filter helps with optimization. In order to whitelist signers, the first two characters of their addresses (the first `byte`) have to be unique. For example, let's say that you have 15 different price signers, none of the first two characters of their addresses can be the same. This helps to filter that all 15 signers are different.

Next, there are `lift` functions. These functions tell us who can sign messages. Multiple messages can be sent or it can just be one but they are put into the authorized oracle). However, there is currently nothing preventing someone from `lift` 'ing two prices signers that start with the same address. This is something for example, that governance needs to be aware of (see an example of what a governance proposal would look like in this case in the **Gotchas** section).

Due to the mechanism design of how the oracles work, the **quorum** has to be an odd number. If it is an even number, it will not work. This was designed as an optimization (`val = uint128(val_[val_.length >> 1]);`); this code snippet outlines how it works, which is by taking the array of values (all the prices that each of the prices signers reported, ordered from 200-215) and then grabbing the one in the middle. This is done by taking the length of the array (15) and shifting it to the right by 1 (which is the same as dividing by 2). This ends up being 7.5 and then the EVM floors it to 7. If we were to accept even numbers this would be less efficient. This presents the issue that you should have a defined balance between how many you

require and how many signers you actually have. For example, let's say the oracle needs 15 signatures, you need at least 17-18 signers because if you require 15 and you only have 15 and one of them goes down, you have no way of modifying the price, so you should always have a bit more. However, you should not have too many, as it could compromise the operation.

---

4. Gotchas

Emergency Oracles

- They can shutdown the price feed but cannot bring it back up. Bringing the price feed back up requires governance to step in.

Price Freeze

- If you void the oracles Ethereum module, the idea is that you cannot interact with any Vault that depends on that ilk.
  - **Example:** ETHUSD shutdown (can still add collateral and pay back debt - increases safety) but you cannot do anything that increases risk (decreases safety - remove collateral, generate dai, etc.) because the system would not know if you would be undercollateralized.

Oracles Require a lot of Upkeep

- They need to keep all relayers functioning.
- The community would need to self-police (by looking at each price signer, etc.) if any of them needs to be replaced. They would need to make sure they are constantly being called every hour (for every hour, a transaction gets sent to the OSM, which means that a few transactions have already been sent to the median to update it as well. In addition, there would need to be a transaction sent to the `spotter`, as DSS operates in a pool-type method (doesn't update the system/write to it, you tell it to read it from the OSM).

There is nothing preventing from `lift`'ing two prices signers that start with the same address

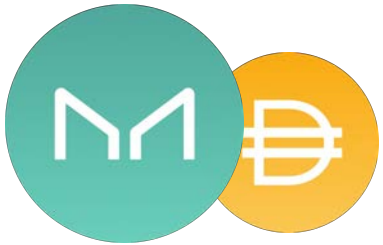
- The only thing that this prevents is that you cannot have more than 256 oracles but we don't expect to ever have that many, so it is a hard limit. However, Governance needs to be sure that whoever they are voting in anyone that they have already voting in before with the same two first characters.
- An example of what a governance proposal would look like in this case:
  - We are adding a new oracle and are proposing (the Foundation) a list of signers (that have been used in the past) and we already have an oracle but want to add someone new (e.g. Dharma or dydx). We would say that they want to be price signers, so these are their addresses and we want to lift those two addresses. They would vote for that, and we would need to keep a list of the already existing addresses and they would need to create an address that doesn't conflict with the existing ones.

5. Failure Modes (Bounds on Operating Conditions & External Risk Factors)

- By design, there is currently no way right now to turn off the oracle (failure or returns false) if all the oracles come together and sign a price of zero. This would result in the price being invalid and would return false on `peek`, telling us to not trust the value.
    - We are currently researching (Oracles ETH module) that would invalidate the price but there is no way to do this in the median today. This is due to the separation of concerns that DSS does not read directly from median, it reads from the OSM, but this may end up changing.

EXHIBIT F

Some sections (which are highlighted in red) are still a WIP
Updated Dec 8th, 2020



Maker Protocol 101

Kenton, Wouter, Soren, Tom, and Chris B.

Maker Foundation

Please submit errors to kenton@makerdao.com or @Kenton on Rocket Chat

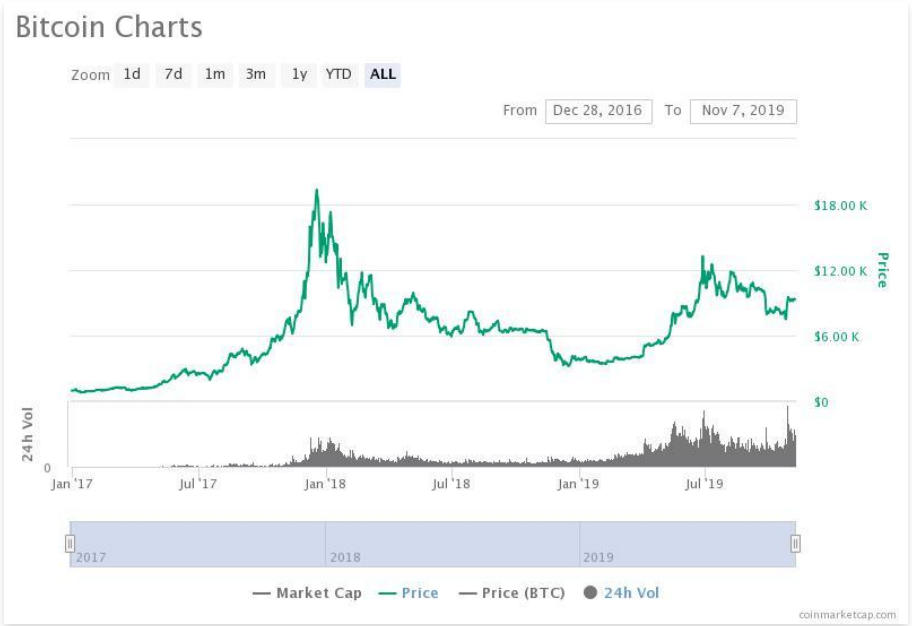
Sections

1. **30 seconds**
    a. Problem and Solution
    b. Dai
2. **3-5 minutes**
    a. Economics
    b. Vault Mechanics
3. **60 minutes**
    a. Smart contract Modules
4. **90 minutes**
    a. Advanced Concepts

Problem: Instability

The ultimate gauge of capital is denominated in the Global Currency, which is USD.

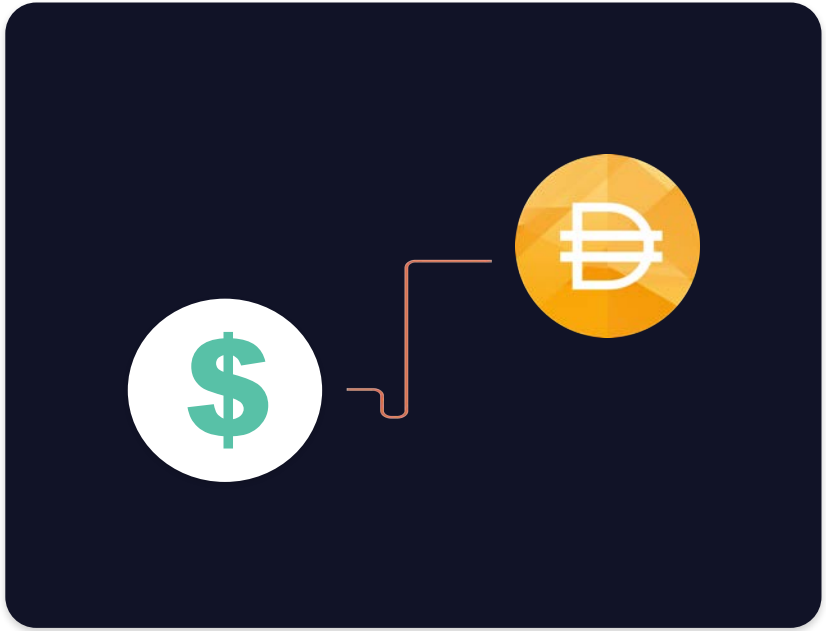
Is Bitcoin stable relative to the Global Currency?



Maker’s Flagship Product

Dai - Stablecoin

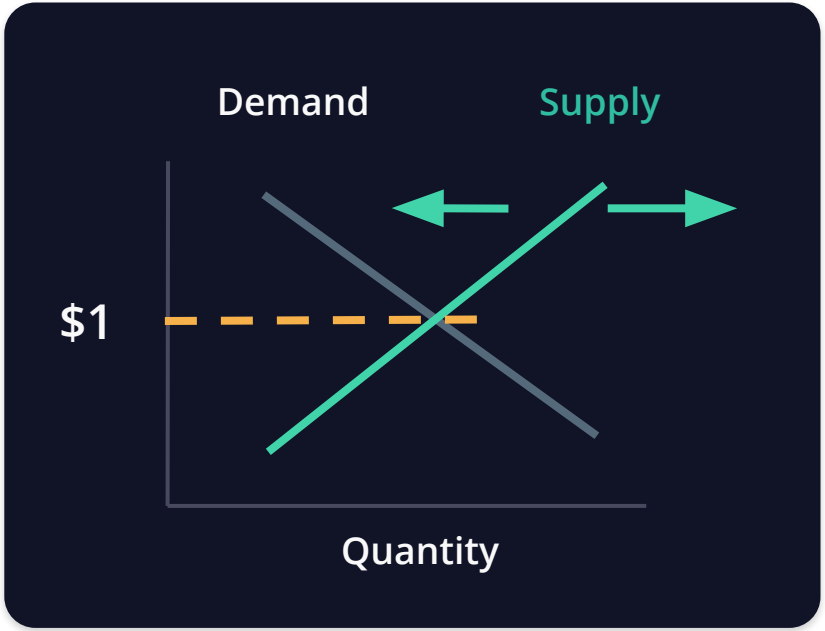
- Dai soft-pegged to USD
- Basic user
- Fully backed by collateral



Dai - Economics

How does it keep its peg?

- Demand curve can shift due to market conditions, confidence of Dai holders, etc
- Supply curve is shifted through a permissionless credit factory on Ethereum
- Any actor can vary the supply of Dai through a Maker Vault
- The system was built so that these actors are incentivized to shift the supply curve to ensure that the price is close as possible to $1



Dai - Vault

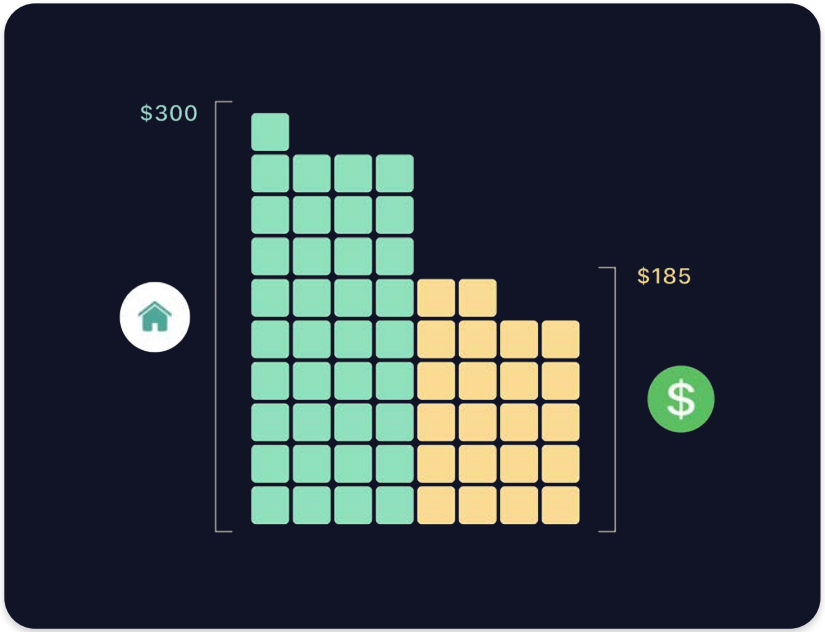
Maker Vault

- Borrow Dai through locking up crypto assets as collateral
- Repay Dai + fee to retrieve collateral
- Safe, over-collateralized Vault >

Dai - Vault

Analogous to a Mortgage

- Bank gives you a loan by “locking” ownership rights with them
- Repay debt to “free” the bank’s ownership of the house



Dai - Vault

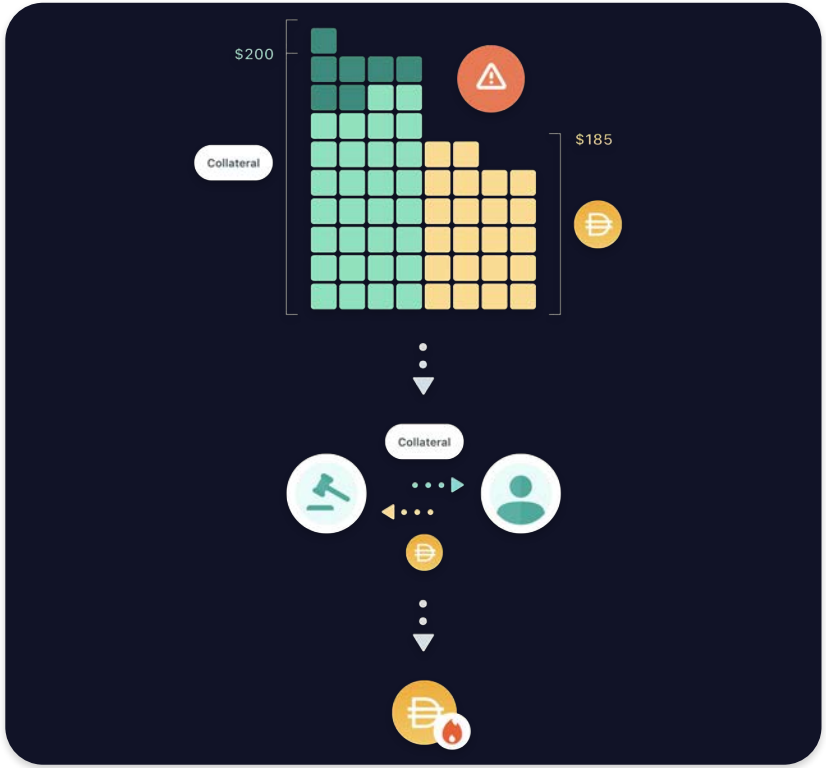
Liquidations

1. A Vault is automatically liquidated if the collateral value (in USD) falls too low

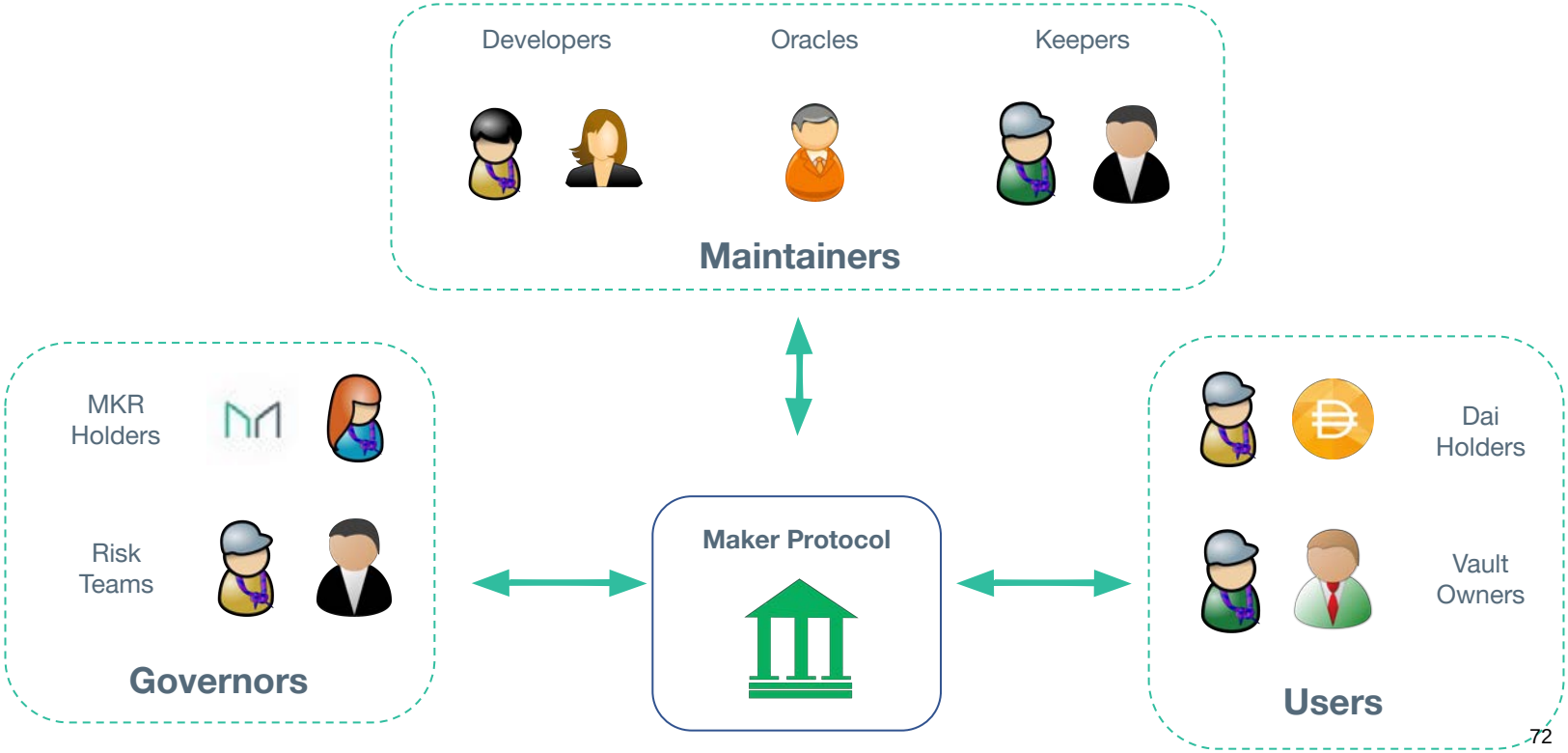
2. Part of the collateral is auctioned off by the Protocol to cover the outstanding debt + penalty fee

3. Dai is then burned by the Protocol to decrease the supply

   Vault owner receives the leftover collateral



System Interaction Diagram



System Interaction Actors

User

- Dai Holders
  - Basic - No additional knowledge to own Dai
  - Stability Seekers; Consumer; Businesses
- Vault Owners
  - Intermediate - Some knowledge
  - Risk Seekers; Speculators; Borrowers

Governor

- MKR Holders
  - Advanced - Extensive knowledge
  - Monitor, partake, and vote on upgrades/changes in the Maker Protocol
- Risk Teams
  - Advanced - Extensive knowledge
  - Collect/compile relevant data and develop risk models, assessed by MKR Holders

System Interaction Actors

Maintainer

- Developers
  - Advanced - Extensive knowledge
  - Supports system upgrades accepted by Governors
- Oracles
  - Intermediate User - Some knowledge
  - Supports data feed from real world to blockchain
- Keepers
  - Advanced User - Extensive knowledge
  - Builds/maintains systems that profit off system discrepancies and participate in auctions

Anyone with the required knowledge can freely participate in any role

Any one person (or service) can have multiple roles

System Lines of Defense

At any point, the system must have more value in collateral than value of Dai supply.

The following mechanisms help maintain system solvency:

1. Supply and Demand	Supply and demand of Vaults (and thus Dai) is influenced by the Stability Fees, Dai Savings Rate, and Debt Ceiling adjustments.
2. Liquidation	Any time the collateral value of a Vault gets closer to its debt, it becomes “risky-er”. The system liquidates Vaults that get too risky.

System Lines of Defense

3. MKR Minting/Burning

If MKR holders govern the Maker Protocol successfully, the Protocol will accrue Surplus Dai as Dai holders pay Stability Fees. On the other hand, if liquidations are inadequate, then the Protocol will accrue Bad Debt. Once this Surplus Dai / Bad Debt amount hits a threshold, as voted by MKR holders, then the Protocol will discharge Surplus Dai / Bad Debt through the Flapper / Flopper smart contract by buying and burning / minting and selling MKR, respectively.

4. Emergency Shutdown

This is a process that is used as a last resort in cases of extreme market irrationality, attacks, or coordinated upgrades. Emergency Shutdown gracefully settles the Maker Platform while ensuring that all users, both Dai holders and Vault users, receive the net value of assets they are entitled to.

Smart Contract Modules

Table of Contents

Sai vs Dai

Single Collateral DAI

- Single collateral type backing Dai
- Collateral is ETH
- Less diversified
- Liquidations occur at a fixed discount to the current collateral price
- MKR used to pay stability fee
- Decentralized governance through MKR voting rights

Multi Collateral DAI

- Multi. collateral types backing Dai
- Collateral can be any approved asset
- More diversified and stable
- Liquidations are handled using Auctions
- Dai used to pay stability fee
- Decentralized governance through MKR voting rights

Sai vs Dai - Terminology

Single Collateral DAI

- Dai Credit System
- CDP
- Boom / Bust Spread

Multi Collateral DAI

- Maker Protocol
- (Maker) Vault
- Discount / Premium Spread

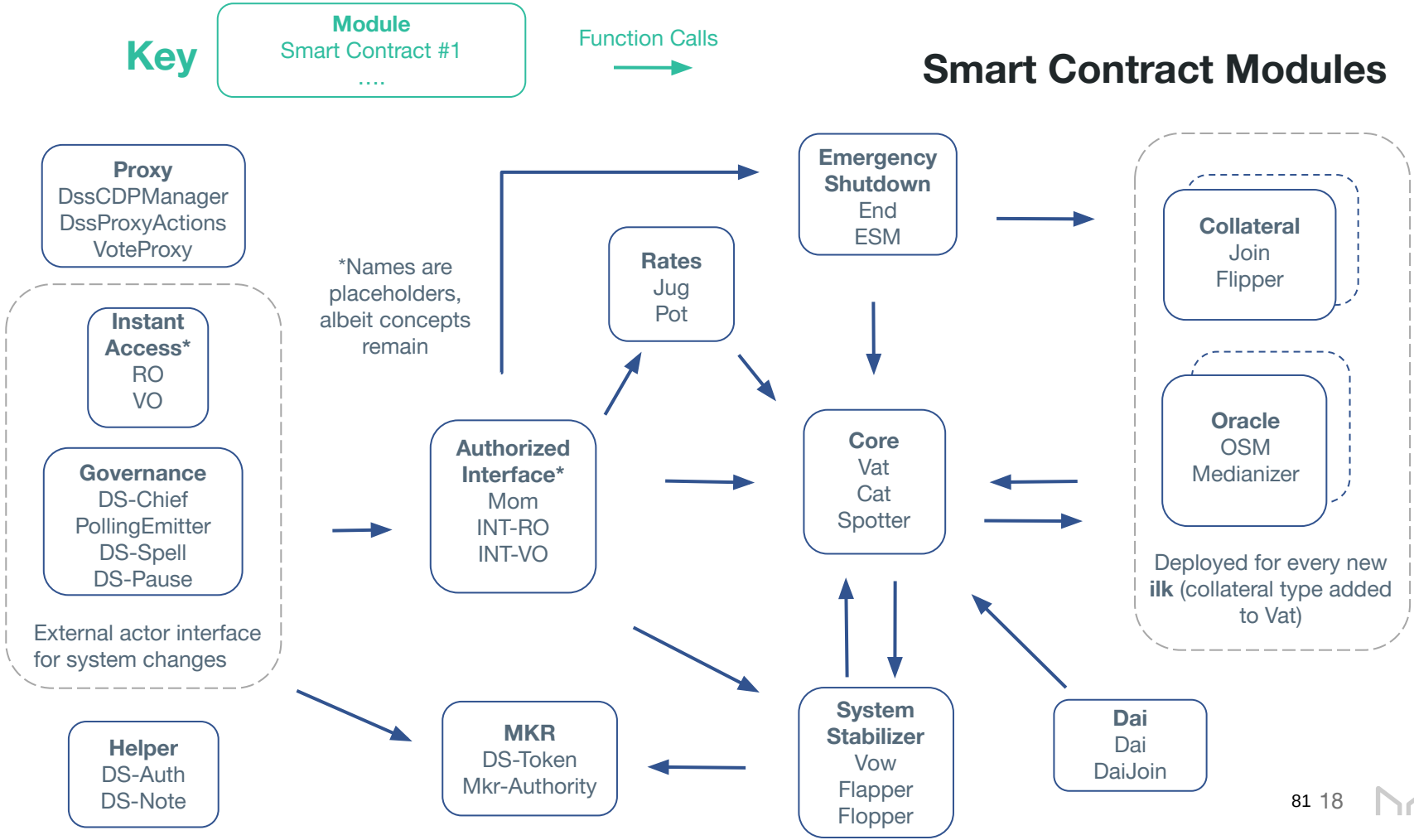
Quick Vocab - Rates

- **Risk Premium Rate** - This rate is used to calculate the risk premium fee that accrues on debt in a Vault. A unique Risk Premium Rate is assigned to each collateral type. (e.g. 2.5%/year for Collateral A, 3.5%/year for Collateral B, etc)
- **Base Rate** - This rate is used to calculate the base fee that accrues on debt in a Vault. A system wide Base Rate is assigned to all collateral types. (e.g. 0.5%/year for the Maker Protocol)
- **Stability Rate** = *Risk Premium Rate* + *Base Rate*. This rate is used to calculate the **Stability Fee**.
- **Dai Savings Rate (DSR)** - This rate is used to calculate the dai earned that accrues on Dai locked in the savings contract. A system wide Dai Savings Rate is assigned to all Dai locked in the DSR contract. (e.g. 1%/year for DSR)

All rates accrue regularly on a per second basis.

https://github.com/makerdao/dss/wiki/Glossary

Smart Contract Modules



Motivations for concise names

To sidestep terminological debates; for example, whether to say *rate of target price change* or *target rate*

To decouple financial and technical vocabularies; we can more flexibly improve one without affecting the other.

To discuss the system formally; this ability, with the financial interpretation partly suspended, has suggested insights that would have been harder to think of inside the normal language.

To better formalize the implementation; the precise and distinctive language makes the structure and logic of the implementation more apparent and easier to formalize.

To decrease verbosity; concise names make the code less verbose and the concepts easier to handle on paper, whiteboard, etc.

Quick Vocab - general

Risk Parameters - system variables adjusted through MKR governance to control various types of risk.

Important parameters for each collateral type:

- **Stability fee** - a fee that continuously accrues on debt in a Vault (e.g. 2.5% per year)
- **Debt ceiling** - max amount of Dai generated for a given collateral type (e.g. 10 million Dai)
- **Liquidation ratio** - minimum ratio of collateral value to debt per Vault (e.g. 150%)

ilk - collateral type; each has its own set of risk parameters

urn - vault; an ethereum address can control one *urn per collateral type*

gem - unlocked collateral; gem is collateral that is not yet locked in a Vault but still recorded in the system

sin - system debt unit; a debt balance that is tracked during liquidation process

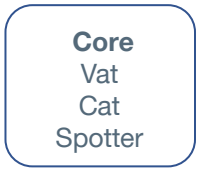
dai - stablecoin; a good debt token

https://github.com/makerdao/dss/wiki/Glossary

Core Module - what

The core module contains the state of the Maker Protocol and its central mechanisms while in normal operation.

Components



- **Vat** - **The single source of truth for the Maker Protocol.** It contains the accounting system of the core Vault, Internal Dai balances, and collateral state. The Vat has no external dependencies and maintains the central "Accounting Invariants" of the Maker Protocol. It houses the public interface for Vault management, allowing urn (Vault) owners to adjust their Vault state balances. It also contains the public interface for Vault fungibility, allowing urn (Vault) owners to transfer, split, and merge Vaults. Excluding these interfaces, the Vat is accessed through *trusted smart contract modules*.
- **Cat** - Public interface for **confiscating unsafe urns** (Vaults) and **processing seized collateral** via their respective flip (collateral) auction. With large Vaults, partial confiscations of a fixed collateral size will be processed until the urn becomes safe again.
- **Spotter** - Allows external actors to **update the price feed** in Vat for a given Ilk (collateral type).

Core Module - why Vat

Components

- **Vat** - def. “a large tank or tub used to hold liquid”

  The Vat holds the fundamental primitives of the Maker Protocol:

  - Database - Risk parameters as well as Dai, Sin, collateral, and debt balances
  - Accounting System - Basic accounting operations to update the Database
  - Vault Management - Adjustment of locked collateral and debt position (Dai creation)
  - Vault Fungibility - Ability to transfer, split and merge Vaults

  The Vat and these primitives are designed to be non-upgradable or replaceable.

  Other components of the system, such as auctions, oracles, and rate accumulators, are subject to future development. Their business logic has been contained within their own smart contract modules as an upgradable interface between the user and the Vat. Since these subsystems have access to more complex operations within the Vat, module upgrades are voted in by MKR holders. Through Vault management/fungibility, direct access to the Vat provides a strong guarantee to users that the basic semantics/interface aren’t going to change.

https://github.com/makerdao/dss/blob/master/src/vat.sol

Core Module - why Cat

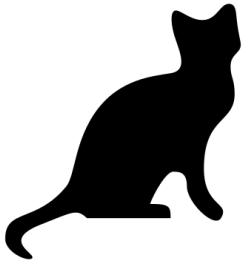
Components Continued

- **Cat**

  The Cat **“bites” Vaults that are too risky**.

  The generator of Dai is a Vault owner and the “borrower”, while the smart contract system is the “generator”. To increase the borrower’s confidence in the minted value lent out, the generator requires all minted Dai to be fully backed by an asset with value that is proven in free markets. If the value of the underlying asset dips below the required amount, the collateralization ratio (USD value of asset / USD value of Dai debt) decreases. To increase this ratio and prevent insolvency of the system, the generator takes (via Cat) the collateral and sells it for Dai in an auction (Flipper).

  Remember: 1 Dai must always be backed by more than 1 USD worth of assets.



https://github.com/makerdao/dss/blob/master/src/cat.sol

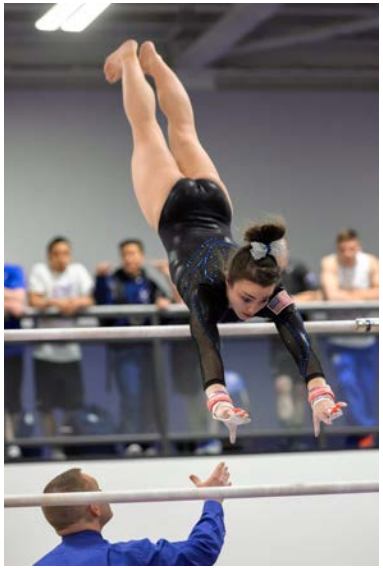
Core Module - why Spotter

Components Continued

- **Spotter** - def. “Gymnastics Spotting or to “spot” someone means to physically assist them in safely completing a skill”

  The Maker Protocol requires real time information about the market price of the assets used as collateral in Vaults. Ultimately, this market price determines the amount of Dai that can be minted, as well as the grab condition for Vault liquidations. The oracle module handles how markets prices are recorded on the blockchain.

  The Spotter is simply an **interface contract** where external actors pull the **current market price** from the Oracle module for the **specified collateral type**. The Vat reads the market price from the spotter.



https://github.com/makerdao/dss/blob/master/src/spot.sol

Collateral Module - what

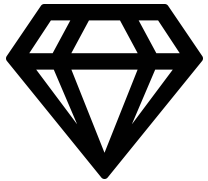
The collateral module is deployed for every new ilk (collateral type) added to Vat. It contains all adapters and auction contract for one collateral type. All token behavior is *abstracted behind* these adapters.

Components

- **Join** - The Join adapter is used to **deposit/withdraw unlocked collateral** into the Vat. Currently, GemJoin is only compatible with standard ERC20 tokens, but eventually there will be various types of Join adapters that are compatible with different Token Standards.
- **Flipper** - **Collateral auction house.** Each gem auction is unique and linked to a previously bitten urn (Vault). Investors bid with increasing amounts of DAI for a fixed GEM amount. When the DAI balance deficit is covered, bidders continue to bid for a decreasing gem size until the auction is complete. Remaining GEM is returned to the Vault owner.

Collateral Module - why

Components



- **Join** - def. of Gem - “a precious or semiprecious stone”

  To retain the security of the system, only trusted smart contracts can add/remove value to/from the Vat. A Join adapter is a trusted smart contract that is used to **deposit unlocked collateral** (gem) within the Vat. The location of collateral deposited/locked in Vaults is in the the appropriate Join adapter.

- **Flipper** - def. of Flip - “turn over or cause to turn over with a sudden sharp movement.”

  The purpose of the gem auction house is to decrease the market risk of collateral backing Dai. It sells an amount of seized gem to **purchase and burn Dai,** which will **increase the collateralization ratio** of the system, away from insolvency.

  Priorities for the Flipper:

  1. “Tend” phase: Cover the amount of total debt (minted Dai + accrued fees) of the Vault
  2. “Dent” phase: Return as much collateral back to the Vault owner as possible

https://github.com/makerdao/dss/blob/master/src/join.sol
https://github.com/makerdao/dss/blob/master/src/flip.sol

Dai Module - what

Fundamentally, 'Dai' is any token that the core system considers equal in value to its internal debt unit.

The dai module contains the dai token representation and all adapters thereof.

Components

- **Dai** - An extension from DS-Token and standard ERC20 token interface. Contains the database of Dai token owners, transfer, approval and supply logic.
- **DaiJoin** - DaiJoin is an adapter where all Dai tokens are created. The Vault owner interacts with DaiJoin to mint the Dai tokens that has been allocated for them in the Vat as well as burn Dai Tokens + fees accrued against their Vault.

Dai Module - why

Dappsys

Components

- **Dai** - Dai is an extension of DS-Token, a contract within DappSys, a safe, simple, flexible library for smart-contract systems.

  DSToken is an implementation that supports the **ERC20 Standard, but with a few additions** that complement the design of the Maker Protocol:

  - *Addition of mint and burn functions (with proper authorization)* -> to control token supply
  - *'push', 'pull' and 'move' aliases for transferFrom operations* -> improves readability
  - *Binary allowance approval* -> lower gas and higher security

  Dai is similar to DSToken, but it **omits** binary approval and contains a novel permit function that allows users to **give allowance without needing ETH**

- **DaiJoin**

  DaiJoin is a trusted smart contract that is used to deposit Dai into the Vat. All minting and burning of Dai tokens happens in DaiJoin (think "United States Mint").

https://github.com/makerdao/dss/blob/master/src/dai.sol
https://github.com/makerdao/dss/blob/master/src/join.sol

System Stabilizer Module - what

When the value of the collateral backing Dai drops below the liquidation level, then the stability of the system is at risk. The system stabilizer module sets up incentives for Keepers (incentivized external actors) to step in, push the system back to a safe state, and earn profits.

System Stabilizer
Vow
Flapper
Flopper

Components

- **Vow** - The Vow represents the **Maker Protocol's balance**, as the recipient of both system surplus and system debt. Its function is to cover deficits via debt auctions and discharge surpluses via surplus auctions.
- **Flopper** - Debt Auction house. Debt auctions are used to satisfy the Vow's debt by auctioning off MKR for a fixed amount of internal Dai. **Bidders compete with decreasing "amount requests" of MKR**. After auction settlement, the Flopper sends the received internal Dai to the Vow to cancel out its debt. The Flopper mints the MKR for the winning bidder.
- **Flapper** - Surplus Auction house. Surplus auctions are used to liquidate the Vow's surplus by auctioning off a fixed size of internal Dai for MKR. **Bidders compete with increasing amounts of MKR.** After auction settlement, the Flapper burns the winning MKR bid and sends the internal Dai to the winning bidder.

System Stabilizer Module - why

Components

- **Vow** - def. “a solemn promise.”
  The Maker Protocol deviates from equilibrium when it receives system debt and system surplus through collateral auctions, Dai Savings and Vault stability fee accumulation. The Vow houses the business logic to kick off debt and surplus auctions, which correct the system’s monetary imbalances.

  System debt: **When Vaults are bitten**, their debt is taken on by the Vow as Sin, the system debt unit, and placed in the Sin queue. If this Sin is not covered by a flip auction within some wait time, the Sin “matures” and is now considered bad debt to the Vow. This bad debt can be covered through a debt auction when it exceeds a minimum value (the `lot` size). The source of **Dai Savings Accumulation** comes from increasing Sin (system debt) in the Vow.

  System surplus: **Stability fee accumulation** occurs in the form of additional internal Dai to the Vow. This surplus is then discharged through surplus auctions.

https://github.com/makerdao/dss/blob/master/src/vow.sol

System Stabilizer Module - why pt. 2

Components continued

- **Flopper** - def. of Flop - “be completely unsuccessful; fail totally”

  The purpose of the debt auction is to cover the system deficit, which is represented by Sin. It sells an amount of minted MKR and purchases Dai to be canceled 1-to-1 with Sin.

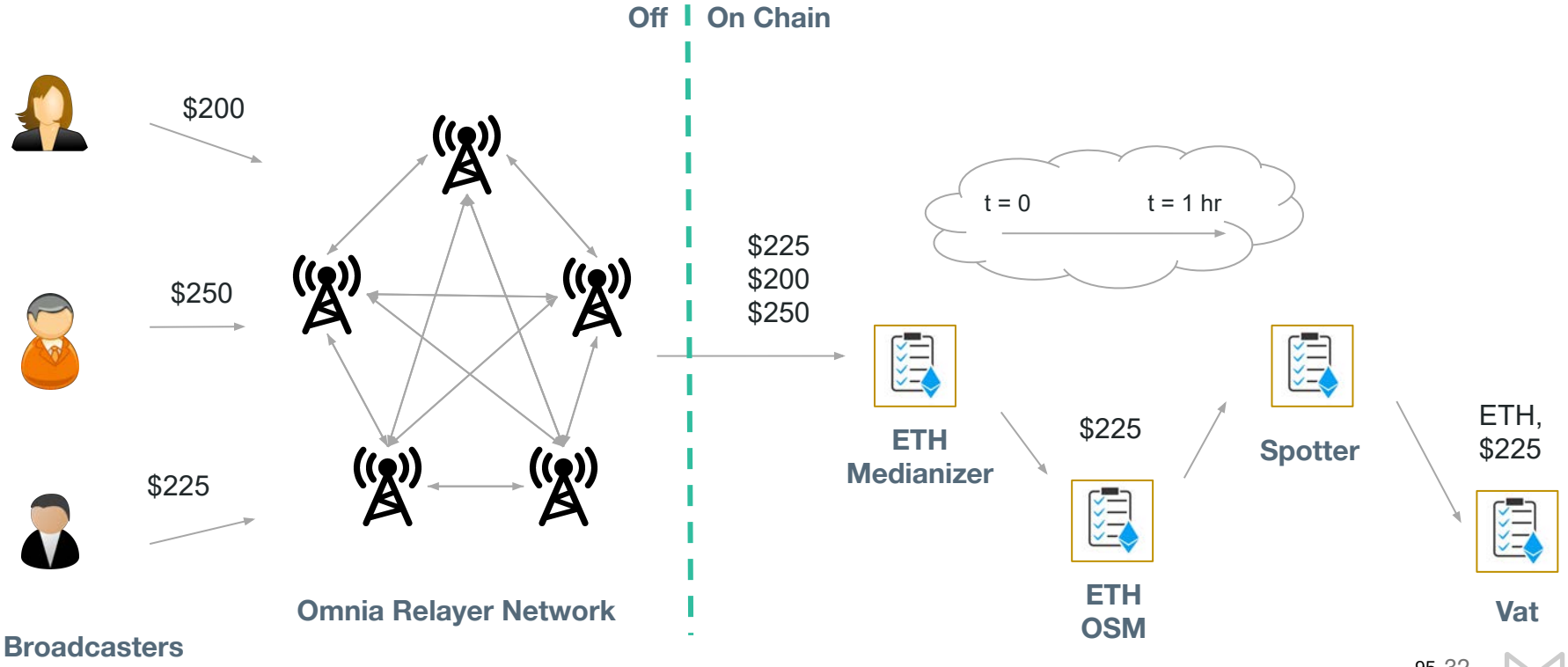
  Priorities for the Flopper:
  1. Raise an amount of Dai equivalent to the amount of bad debt **as fast as possible**
  2. Minimize the amount of MKR inflation

- **Flapper** - def of Flap - “(of a bird) move (its wings) up and down when flying or preparing to fly”

  The purpose of the surplus auction is to release Dai surplus from the Vow. It sells a fixed amount of Dai to purchase and burn a bid amount of MKR.

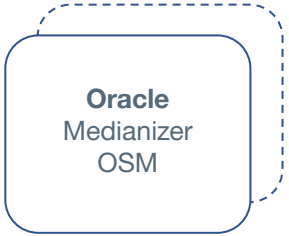
https://github.com/makerdao/dss/blob/master/src/flap.sol
https://github.com/makerdao/dss/blob/master/src/flop.sol

Oracle Module - what



Oracle Module - (what & why) pt. 2

The value of collateral in a Vault is derived from its global, free market USD price. An oracle module is deployed for each collateral type. It feeds price data for a corresponding collateral type to the Vat. Whitelisted addresses broadcast price updates off-chain, which are fed into a medianizer before being pulled into the OSM. The Spotter reads from the OSM.



Components

- **Medianizer** - For a specific Ilk, the Medianizer returns the median value of several price feeds, fed from the off-chain Omnia Relayer Network. A median value is determined to mitigate the variability in single data points.
- **OSM** (Oracle Security Module) - Authorized users are allowed to set a value after some duration of time (e.g. one hour). To protect the system from an attacker who gains control of a majority of the oracles, the OSM imposes a **1 hour delay on price feeds**, leaving enough time for the MKR governance community to analyze the data and react.

https://github.com/makerdao/medianizer
https://github.com/makerdao/osm

Mkr Module - what

The MKR module contains the MKR token representation and its custom authority

MKR
DS-Token
Mkr-Authority

Components

- **DS-Token** - An implementation supporting the ERC20 Standard; part of the DappSys (DS) library. Contains database of MKR owners, transfer and supply logic.
- **Mkr-Authority** - Custom authority contract for allowing **Maker Governance to govern the MKR** token contract. Mkr-Authority is controlled by the DSPauseProxy, which is exclusively owned by DS-Pause. Thus, DS-Pause has indirect permission to give authority to other contracts to call certain functions on the MKR token contract. In practice, this is how the Flopper contract has access to call the `mint()` function during the Protocol's Debt Auctions. Such a structure allows governance proposals voted in on the Chief to make arbitrary changes to the MKR token and its permissions, subject to a delay.

https://etherscan.io/address/0x9f8f72aa9304c8b593d555f12ef6589cc3a579a2#code
https://github.com/makerdao/mkr-authority

Mkr Module - why

The MKR module contains the MKR token representation and its custom authority

As a **utility token**: As Dai stability fees earned on Vaults accrue within the Maker Protocol, MKR holders may vote to enable the Flapper auction house to sell Dai surplus for MKR. Once the auction is complete, the Maker Protocol burns the MKR.

As a **governance token**: MKR is used by MKR holders to vote for the risk management and business logic of the Maker Protocol. Tokens are a simple representation of voting power.

As a **recapitalization resource**: In narrow circumstances, MKR can be autonomously minted by the Flopper auction house and sold for DAI, which is used to recapitalize the Maker Protocol in times of insolvency.

https://medium.com/makerdao/what-is-mkr-e6915d5ca1b3

MKR
DS-Token
Mkr-Authority

Governance Module - what

The Governance Module contains the contracts that facilitate MKR voting, proposal execution, and voting security of the Maker Protocol.

Governance
DS-Chief
PollingEmitter
DS-Spell
DS-Pause

Components

- **DS-Chief** - A basic voting contract that grants root access of the Maker Protocol to an elected “Chief” (address). Through Approval Voting, voters **lock up their MKR and vote** with a weight relative to the outstanding supply of MKR. Spells (proposals) are a type of Proposal Object and are submitted to DS-Chief as Executive Proposals, which can make a change to the protocol (adjusting risk parameters, upgrade adapters, add new collateral types, etc). Anyone can create a Spell, and MKR holders can vote on bundles of Spells, called Slates. At any point, **the Spell (proposal) with the most approval is the elected “Chief”**, has access to and can configure the Maker Protocol through Mom, the Admin interface contract for Maker Governance.

- **PollingEmitter** - Also known as the symbolic voting contract, the PollingEmitter is a **lightweight contract** that is used to **vote on Maker Governance polls**. Both the polls and the votes thereof are emitted events. The poll event contains the poll’s start date, end date, and a hash of its details, rules and metadata. The vote event contains the poll id and voter address. Anyone can create a poll and cast votes. Votes are tallied off-chain by reading the amount of MKR owned by the address in DS-Chief, held directly by the address, as well as any held in the vote proxy associated with the voter address. The checkpoint tally will be regularly performed during the poll, but the final tally will be at the block specified by the poll’s end date.

https://github.com/dapphub/ds-chief
https://github.com/makerdao/symbolic-voting

Governance Module - why

Components

- **DS-Chief** - def. of Chief - “a leader or ruler of a people or clan”

  DS-Chief is the first iteration of an on-chain tool for Maker Governance. It simply allows MKR holders to vote with a weight relative to their proportional holdings. **Votes are casted towards Proposal Objects**, which are ethereum addresses that can represent:

  - A Spell (contract with one function that does one action, one time)
  - A MegaSpell (contract with one function that does multiple actions, one time)
  - Multi-Signature Contract
  - Externally owned account, etc

This governance mechanism employs an ACL (access control list) approach, where there is a single owner that has permission to call protected functions on Mom (Admin. interface for the Maker Protocol). **At any time, this single owner is the Proposal Object that has the most votes/approval in DS-Chief**; it could change during every executive voting period, which is facilitated through off-chain coordination. Within this period, MKR holders are encouraged to vote for a Slate (bundle of Spells) that includes the previous Spell. This secures the election of the old “Chief” until enough MKR approves the new Proposal Object. In other words, this prevents the chance of an unintended Proposal Object from being elected “Chief” during the approval phase of the new Proposal Object.

https://github.com/dapphub/ds-chief

Governance Module - why

Components

- **PollingEmitter** - def. of Emitter - “a machine, device, etc., that emits something”

  In the past, DS-Chief was used to gauge MKR holder sentiment through governance proposals (polls) and votes. This had profound limitations: an inability to run multiple concurrent polls and unwanted cross-contamination between polls and executive votes. To decouple governance and executive votes, the PollingEmitter contract was created to facilitate Maker Governance Polling. It has the following characteristics: censorship resistance, cryptographic verifiability, on-chain audit-ability, and minimal on-chain computation/ gas efficiency.

  All polls and votes are simply **events emitted by this contract**, and **all tallying is done off-chain**. The tradeoff is that, without state, other smart contracts can't query it for information. However, because symbolic votes don't need to be binding on-chain, it was decided that whatever could be pushed to social layer, should be. In brief, the idea here is that voters signal their intent with on-chain events, then services and community members can, using the rules defined in each poll, tally the polls and imbue them with meaning off-chain (e.g. by summing up the amount of MKR voting for each poll option at a certain block).

https://github.com/makerdao/symbolic-voting

Governance Module - (what and why) pt. 2

Components cont.

- **DS-Spell** - def. of Spell - “a form of words used as a magical charm or incantation.”

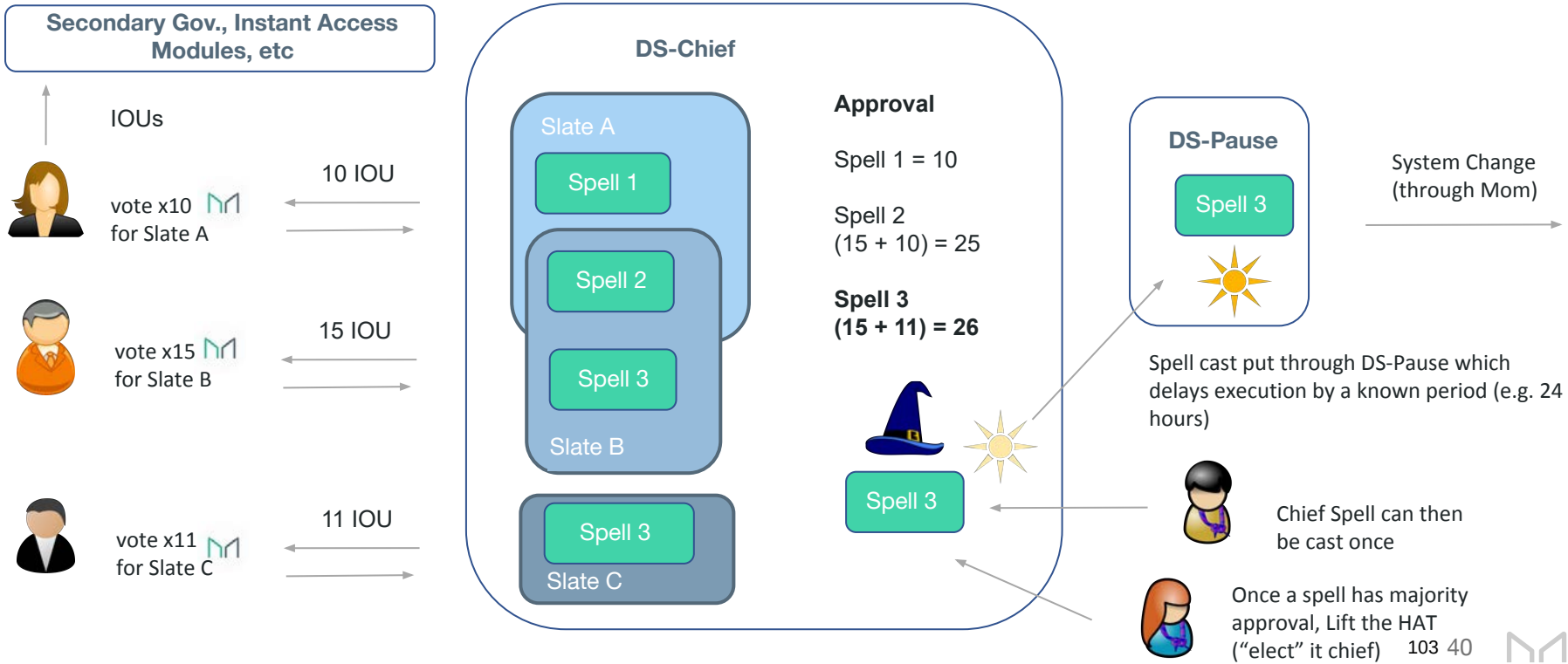
  DS-Spell is a generalized **un-owned contract that performs one action**, or series of atomic actions, one time only. The DS-SpellBook is a DS-Spell factory contract used to create Executive Spells (proposals), which perform single, preconstructed changes to the Maker Protocol. Spells can be “cast” by anyone and if the transaction is successful (no reverts or exceptional conditions), then the **spell is marked as** `done` **and cannot be re-cast**.

- **DS-Pause** - def. of Pause - “interrupt action or speech briefly.”

  Similar to the OSM, DS-Pause **schedules function calls** that can only be executed after some predetermined delay has passed. A DS-Pause implementation for Maker Governance is used to schedule Spells after their approval in DS-Chief; e.g. all Spells are executable after 24 hours. As a security component, DS-Pause ensures that those affected by governance decisions have time to react in the case of an attack. The delay period can be adjusted through an Executive Spell and extended indefinitely for coordinated upgrades to DS-Chief.

https://github.com/dapphub/ds-spell
https://github.com/dapphub/ds-pause

Governance Module - what



Rates Module - what

Compound stability fees through a rate accumulation function.

https://github.com/makerdao/dss/blob/master/src/jug.sol
https://github.com/makerdao/dss/blob/master/src/pot.sol

The Rates module is responsible for collecting stability fees on outstanding Vault debt and distributing dai earned proceeds to dai locked in the Pot (Dai Savings Rate) contract .

Rates
Jug
Pot

Components

- **Jug** - Contains `drip()`, a **public function** used to update an Ilk debt rate for an assigned stability fee. Since a debt rate is a function of time, it should be updated via `drip()` on a regular basis. Auction keepers, MKR holders, and other relevant stakeholders are **incentivized to call** `Jug.drip()`.

- **Pot** - The Pot contract is where a Dai holder would lock up Internal Dai to accrue earned dai at the Dai Savings Rate. Similar to Jug, this contract employs its own Drip function used to update it's own internal rate. This rate follows the Dai Savings Rate and is used in the exchange of a claim to the Pot's `Pie` and Internal Dai. Dai holders, MKR holders, and other relevant stakeholders are **incentivized to call** `Pot.drip()`.

  A portion of the Stability Fee dividends is allocated for the Dai Savings Rate by increasing the amount of Sin in the Vow at every `Pot.drip()` call. This Sin cancels out with Dai that would otherwise be sold off in surplus auctions, the mechanism for collecting stability fees.

Rates Module - why

Components

- **Jug** - Jug updates each Ilk's (collateral type) debt unit rate while the offsetting Dai is supplied to/by the Vow. The effect of this is to **apply accumulated positive/negative stability fees** to the outstanding Dai position of all Vaults. Since a blockchain is inherently passive, it requires an external "reminder" to collect stability fees; this "reminder" is in the form of calling Drip.

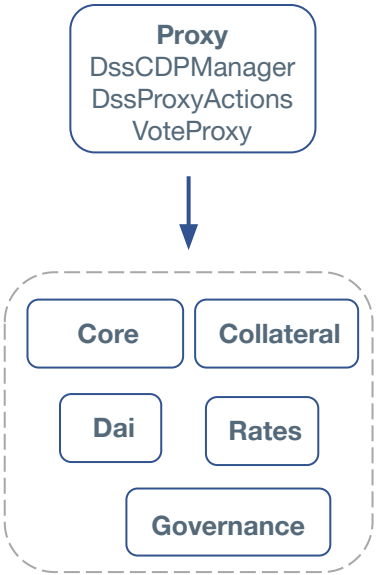
- **Pot** - Pot houses the implementation of the Dai Savings Rate, an **rate that accrues on dai that's locked in the Pot.** This rate is called the Dai Savings Rate, set by Maker Governance, and is typically less than the base stability fee to remain sustainable. This purpose of Pot is to offer another incentive in holding Dai.

The 'Why' of Rates Modules is expanded in *Section 4 - Advanced Topics*

https://github.com/makerdao/dss/blob/master/src/jug.sol
https://github.com/makerdao/dss/blob/master/src/pot.sol

Proxy Module - what and why

The Proxy module is intended to increase the usability and convenience of the Maker Protocol. It contains contract interfaces, proxies, and aliases to functions necessary for Vault management and Maker governance.



Components

- **DssCdpManager** - DssCdpManager was designed to formalize the process of **Vault transfers**, enabling Vaults to be treated more like assets that can be exchanged as non-fungible tokens (NFT). It is recommended that all Vault interaction is funneled through the CdpManager. Once unlocked collateral is deposited into the Maker Protocol, the following features are available:
    - Multi Vault ownership and numerical identification (user can own *n* number of Vaults)
    - Flexible Vault transferability

https://github.com/makerdao/dss-cdp-manager

Proxy Module - what and why

- **DssProxyActions** - **A generalized wrapper for the Maker Protocol**. Utilized by the Oasis Borrow portal, DssProxyActions is a contract with functions that are to be used via a personal ds-proxy. It is similar to the Sai-Proxy and offers functions that execute a sequence of actions atomically, such as `openLockGemAndDraw(...)`.
- **VoteProxy** - The VoteProxy is a contract that **facilitates online voting with offline MKR storage**. Through a personal VoteProxy, a linked hot wallet can pull and push MKR from the proxy's corresponding cold wallet and to DS-Chief, where voting can take place with the online hot wallet. The reason for having the voting proxy contract is two-fold: to support two different voting mechanisms and to minimize the time that MKR owners need to have their wallet online.

https://github.com/makerdao/vote-proxy
https://github.com/makerdao/dss-proxy-actions

Helper - what and why

The Helper module contains smart contract helpers that handle generic patterns, such as authorization and event logging, that are inherent to the Maker Protocol and Ethereum Blockchain, respectively.

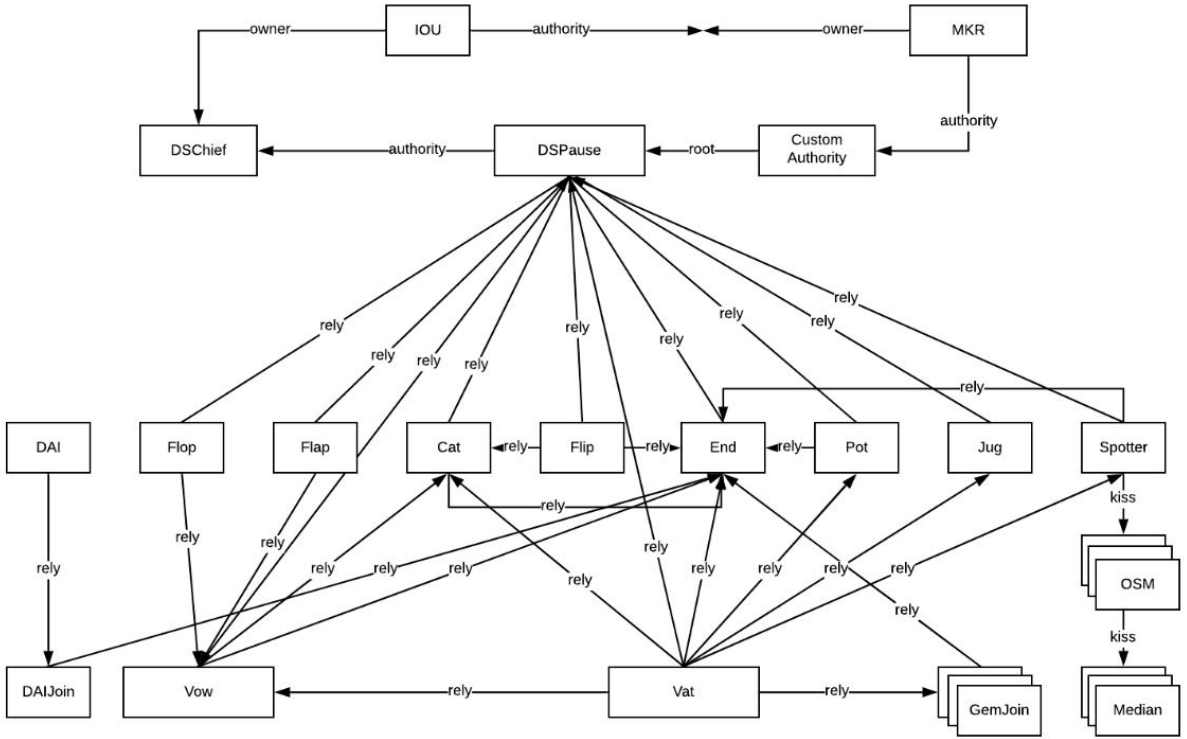
Components

Helper
DS-Auth
DS-Note

- **DS-Auth** - **Fully updatable unobtrusive auth pattern**. By design, the Maker Protocol has adopted a multi-owner authentication system, which prevents unauthorized calls that would otherwise destabilize normal operation or expose a vulnerability. Its implementation is in the form of DS-Auth, which provides a flexible and updatable `auth` modifier that restricts function-call access to the contract, contract owner, or address with granted permission via a specified `authority`.
- **DS-Note** - **Log function calls as events.** Similar to DS-Auth, DS-Note provides unobtrusive, generic function call logging by way of a `note` modifier, which triggers the capture of data as a LogNote event. Functions with this modifier will log information whenever they are called with the indexed fields being queryable by blockchain clients.

https://github.com/dapphub/ds-auth
https://github.com/dapphub/ds-note

Helper - ex of authorization within Maker Protocol



https://github.com/makerdao/dss/wiki/Auth

Example from 11/4/2019

Emergency Shutdown - what

https://github.com/makerdao/dss/blob/master/src/end.sol
https://github.com/makerdao/esm

Emergency Shutdown is the last resort to protect the system against a serious threat, such as governance attacks long-term market irrationality, hacks, and security breaches.

The Emergency Shutdown (ES) module is responsible for coordinating emergency shutdown, the process used to gracefully shutdown the Maker Protocol and properly allocate its collateral to both Vault/Dai users.

Emergency Shutdown
End
ESM

Components

- **End** - End is a smart contract that **facilitates Emergency Shutdown** within the Maker protocol; it has calling authority to much of the system and is the User's interface for claiming collateral. The ESM is authorized to initiate ES by calling the protected `cage()` function, which initiates Emergency Shutdown. It supports various scenarios, ranging from over-collateralization among all ilks to global under-collateralization, which is when the net value of all collateral types is less than the total Dai supply. In the latter edge cases, where the collateral base is limited, the Vault owner payout (in the form of excess collateral) is prioritized over the claims of Dai holders.
- **ESM** - The Emergency Shutdown Module (ESM) is **a contract with the ability to call** `End.cage()` and initiate ES. MKR holders join their funds, which are then immediately burnt. When the ESM's internal sum balance is equal to or greater than the minimum threshold, then `End.cage()` can be called.

Emergency Shutdown - why and how

Components

- **End** - Emergency shutdown is an involved, deterministic process, **requiring interaction** from all user types: Vault owners, Dai holders, Keepers, MKR governors, and other Maker Protocol Stakeholders. The high-level steps are as follows:

1. The ESM **calls `cage()` function**, which freezes the protocol and locks spot prices (collateral USD prices) for each ilk (collateral type).

2. Next, Vault holders interact with End to settle their Vault and **withdraw excess collateral**.

3. After collateral auctions have concluded or been canceled and the system has settled all large Vaults, Dai holders can begin to **claim a proportional amount** of each collateral type at a fixed rate that corresponds to Dai circulation and USD value of the asset at the time ES was initiated.

The 'Why and How' of End is expanded on in *Section 4 - Advanced Topics,* starting on **slide 63**

https://github.com/makerdao/dss/blob/master/src/end.sol

Emergency Shutdown - why and how

Components

- **ESM** - The Emergency Shutdown Module (ESM)

  Similar to DS-Chief, the ESM is a contract that locks MKR and has the ability to call `End.cage()` and initiate ES.

  It is meant to be **used by an MKR minority** to thwart two types of attack:

  1. Malicious governance
  2. An attack facilitated by a critical bug

  MKR holders **lock their MKR** in the ESM, which are then **immediately burnt**. When the ESM's internal sum balance is equal to or greater than the minimum threshold, then `End.cage()` can be called.

  If attack #1 is attempted, pledgers will have no expectation of recovering their MKR from the existing deployment (as there potentially can be a malicious majority that block the required vote). In such a case, the option is to set up an alternative fork in which the funds in which the attackers get no new MKR. If attack #2 is attempted, governance can choose to refund the ESM pledgers by minting new MKR.

https://github.com/makerdao/esm
Guide to lock MKR: https://bit.ly/2Qt8hyy

Instant Access Module - what and why

Not in scope for Nov. 18th launch of Multi-collateral Dai.
SUBJECT TO CHANGE

Instant Access Module houses the components to create direct, bounded changes to the Maker Protocol without consensus in DS-Chief.

Components

- **RO (Rates Oracle)** - Through admin access to the Rates Module, the Rates Oracle enables more dynamic updates to the Risk Premium Rates, Base Rate and Savings Rate. As a secondary governance mechanism, the Rates Oracle allows MKR holders to vote with the IOUs they receive when locking up their MKR in DS-Chief. Furthermore, to vote they have to stake some amount of Dai (e.g. $1000 or $10,000), which will be passed on to the Buffer if they vote for a losing proposal.
- **VO (NFT/LEIN Vault Oracle)** - Controlled by authorized Risk Team(s), the VO holds admin access to add an NFT/LIEN Vault type on the fly under certain restrictions.

*Names are placeholders, though concepts remain

Authorized Interface Module - what

Not in scope for Nov. 18th launch of Multi-collateral Dai.
SUBJECT TO CHANGE

The Authorized Interface Module holds the interfaces between the system and the governance contracts.

Authorized Interface*
Mom
INT-RO
INT-VO

Components

- **Mom** - Mom is a contract interface to adjust the risk parameters of the Maker Protocol. The chief in DS-Chief has the exclusive authority to call functions through Mom. The following contracts rely on Mom: Spotter, Cat, Vow, Vat, and Jug.
- **INT-RO (Rates Oracle Interface)** - Interface contract for the Rates Oracle. Accessible by the Rates Oracle. It has bounded authority over the Rates Module.
- **INT-VO (NFT/Lein Vault Oracle Interface)** - Interface contract for the NFT/LEIN Vault Oracle. It is authorized to add NFT/LEIN Vault types to the system.

*Names are placeholders, though concepts remain

Section 4

Table of Contents

System Design Rationale

Design Considerations

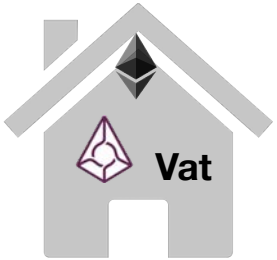
- Token Agnostic
    - System is indifferent to implementation of external tokens
    - The Join **adapters abstract away the differences** between ERC 20s, Non Fungible Tokens (NFTs), invoice tokens, etc
- Verifiable
    - Designed from bottom up to be **well suited for formal verification**; every Vat state defined and proved
    - The Vat makes ***no* external calls**, as functions in external contracts are subject to change
    - The Vat contains ***no* precision loss**; it only adds, subtracts, and multiplies
- Modular and Upgradable
    - Implementations of e.g. auctions, liquidation, Vault risk conditions, and new collateral types, to be altered on a live system through Maker Governance

Asset States within the Vat - what

Collateral

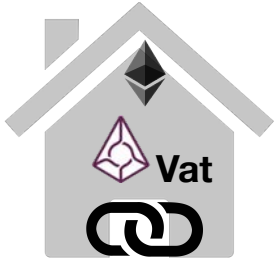
Tokens

Unencumbered
"Unlocked"



Gem credit within the Vat. A user can increase and decrease their allocation (on a per collateral type basis) without restriction

Encumbered
"Locked"



A user has the freedom of choosing how much Gem they wish to expose to liquidation in their Vault. To do this, they lock their Gem credit in a Vault and draw Dai against their now encumbered collateral called **Ink.**

Internal Dai

Dai credit within the Vat. A user can increase and decrease their allocation through the minting and burning of Dai **without restriction**

Asset States within the Vat - why

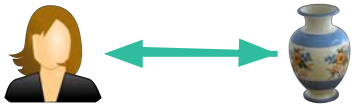
The reason why we have a separate unlocked collateral balance is for security.

To minimize dependencies, the system makes only two types of function calls to external smart contracts:

- TransferFrom( ) to deposit collateral
- Transfer( ) to free collateral

Once some collateral is deposited and registered as a gem balance, the system does not need to make any additional external calls to determine the token balance. **The user's internal unlocked balance is all they have as far as the system is concerned.**

Vault States within the Vat



Note on Step 2
Although Alice freed 7 eth from her unlocked balance, the Vault's collateralization ratio is the same in step 2 and 3, since the **Vault's debt is backed only by the amount of locked collateral.**

$$CR = \frac{Locked\ Collateral\ in\ USD}{Drawn\ Dai\ in\ USD}$$

Step	1	2	3	4
User Action	Deposit 12 Eth	Lock 5 Eth Draw 200 Dai	Free 7 eth	Wipe 200* Dai Unlock 5 Eth
Function called (Contract.method)	GemJoin.Join( )	Vat.Frob( )	GemJoin.Exit( )	Vat.Frob( )
Unlocked Collateral (Gem)	12 eth	7 eth	0	5 eth
Locked Collateral (Ink)	0	5 eth	5 eth	0
Collateral Value (USD) (Assume $100 ETH/USD)	0	$500	$500	0
Debt (Dai)	0	200	200	0
Collateralization Ratio (aka CR)	N/A	250%	250%	N/A

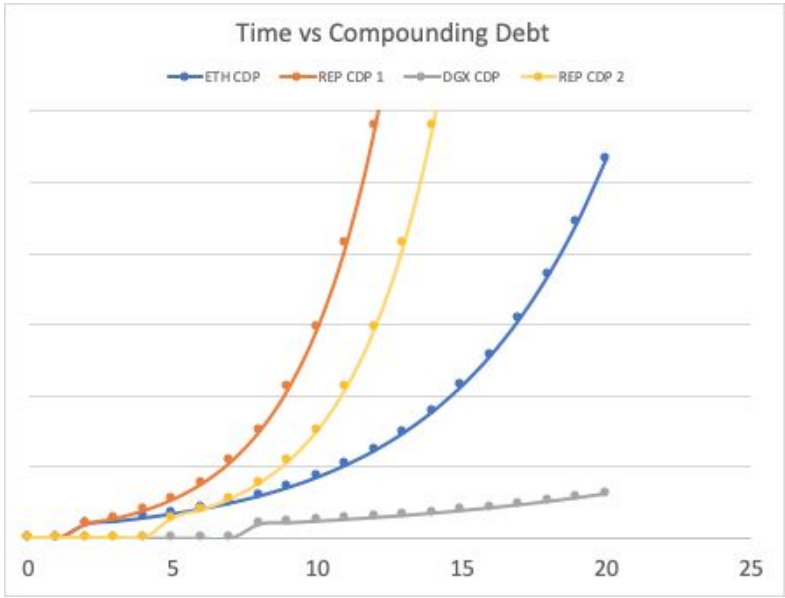
* 200 + stability fees of Dai wiped

Problem with Compounding Fees

The Maker Protocol needs to be able to calculate the total stability fee owed for each Vault, which depends on:

- How much Dai was borrowed
- How long the Dai has been outstanding
- The stability rate(s) while the Dai was outstanding (which can change at any time, even in the middle of a Vault's lifetime)

This needs to be done for all Vaults, and there could be millions of Vaults.



https://github.com/makerdao/dss/wiki/Drip

Thanks to Tyler Sorensen

Solution: Rate Accumulation Function

Keeping track of compounded stability fees for all Vaults only requires storing two types of abstract variables:

- 1 variable per Vault, **(urn.art)**, *adjusted by the user*
- 1 global variable for each collateral type, **(ilk.rate)**, *updated by calling Drip*

Calculating the total debt owed (drawn dai + stability fee accrued) for a given Vault only requires multiplying these two variables together, regardless of changes in stability rates.

The Total Debt is not stored anywhere in the contracts. Instead, when it is needed, the below calculation is performed with the urn's unique **art** and the **rate** for an individual collateral type. A common rate, used among urns of the same collateral type, sidesteps the problem of looping across all urns to update their Total Debt.

$$Total\ Debt\ (Dai) = urn.art\ x\ Ilk.rate$$

https://github.com/makerdao/dss/wiki/Drip

Thanks to Tyler Sorensen

Solution: Rate Accumulation Function

$$Total\ Debt\ (Dai) = urn.art\ x\ Ilk.rate$$

urn.art

What - Abstract variable for an urn. It is equivalent to the urn's total debt when rate equals 1.

How is it updated? - It is adjusted when Debt is drawn/wiped by the urn owner.

$$art = old\ art \pm \frac{additional\ Debt\ drawn/wiped}{rate}$$

Notes - The user must have unlocked/locked collateral (urn.ink) in the Vat to increment/decrement art

ilk.rate

What - Abstract variable for an Ilk. When a new collateral type is added to the Vat, it is set to 1. It is a function of stability rate and time.

How is it updated? - Every time Drip is called.

$$rate = old\ rate * (1 + Stability\ Rate)^{(Time\ since\ the\ last\ Drip\ call)}$$

Notes - As long as the stability fee is positive, Ilk.rate will *increase indefinitely.*

Concept: Rate Accumulation Function

Case: Ilk.rate is updated

An Ilk.rate vs. time graph follows a generalized rate curve, compounded on a per second basis. Every Drip call updates the actual rate and brings it back to this ideal rate curve.

Due to gas costs and the tragedy of the commons problem with the public Drip function, the update frequency could be recurring but irregular.

As the system matures, more stakeholders with more stake will ensure that Drip is called more and the ilk.rate(s) follow the ideal rate curve(s) more closely.

As a consequence, a Vault opened and closed between drip calls (i.e. rate updates) would avoid stability fees.

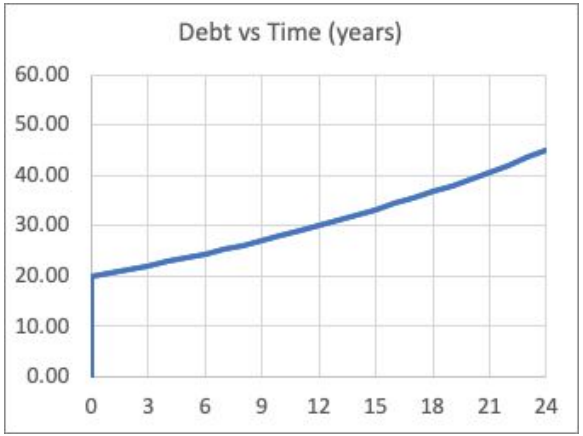
Keeping drip out of the core was largely for a couple security reasons:

1. It was difficult to resolve a Formal Verification proof of functions that would naturally include `drip()`, like `frob()`, `bite()`, etc.
2. It was a design goal to limit the amount of dependencies across the system (i.e. the `Vat` contract should not and does not have any external dependencies). Therefore, we pushed out the drip call to be made by users in `DssProxyActions`, even though calling drip is generally not a requirement

[Compound Stability Fee Example]

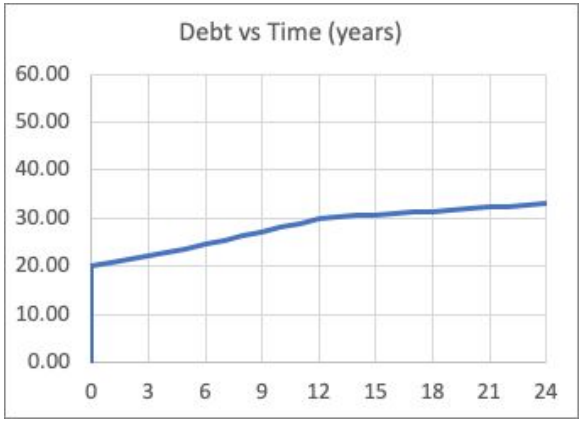
Concept: Rate Accumulation Function

Case: Stability Rate is updated



Step 1: 50% APR
Draw 20 Dai at t = 0

Step 2: 10% APR
Rate Decrease to 10% at t = 12

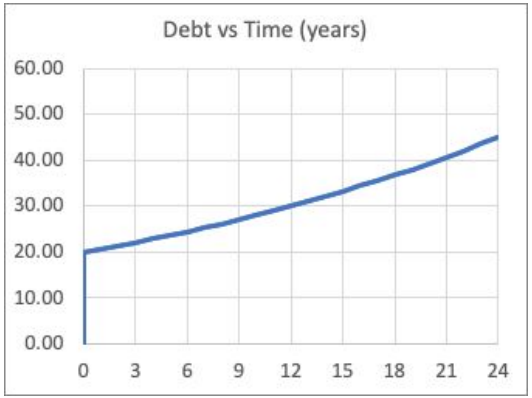


When the stability fee is updated, the next ilk.rate update (i.e. Drip call) would simply use the new 'Stability Rate'. This varies the ilk.rate curve, which has an effect on all Vault debt curves. As long as the stability rate is positive, Ilk.rate will increase indefinitely. Again, in favor of simplicity and storage, the system does not record the stability rate history for each Ilk.

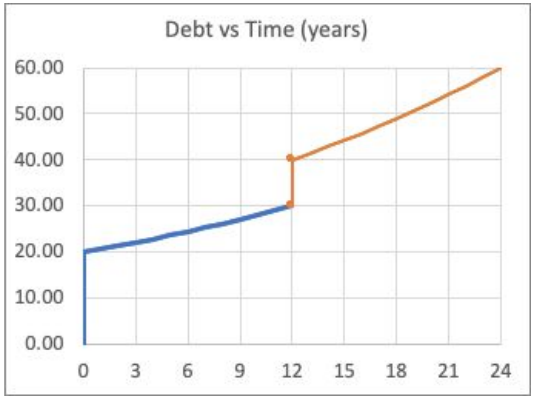
$$rate = old\ rate * (1 + Stability\ Rate)^{(Time\ since\ the\ last\ Drip\ call)}$$

Concept: Rate Accumulation Function

Case: Additional Dai is drawn before debt is wiped



Step 1: 50% APR
Draw 20 Dai at t = 0
Note: at t =0, rate is 1



Step 2: 50% APR
Draw 20 Dai at t = 0
Draw 10 Dai at t = 12

The Vat only records total debt when rate is 1, which is simply urn.art

When additional Dai (debt) is drawn against a Vault, the Debt curve shifts vertically by the amount of additional dai drawn. However, the Vat can only record a change in **urn.art**. Thus, the amount of additional **urn.art** is equivalent to the amount of additional Dai drawn, valued back to when **Ilk.rate = 1**. (Similar to the Time Value of Money concept). The **total debt** (at t=0 and rate=1) that corresponds to the adjusted debt profile in Step 2 is equivalent to 26.667 Dai (green line). Thus, in favor of simplicity and storage, the system does not record the Vault's draw/wipe history.

Emergency Shutdown - Design Characteristics

- **Dai no-race condition** - every Dai holder will be able to redeem the same quantity of collateral, regardless of when they interact with the contract.
- **Vault Parity** - Vault Owners are prioritized and are allowed to redeem their excess collateral before Dai holders.
  - At the time of Emergency Shutdown (ES), individual Vaults, entire collateral types, or the Maker protocol can be undercollateralized, which is when the value of debt exceeds the value of collateral ("negative equity")
  - Maker's current implementation favors Vaults owners in all cases by allowing them to free their entire amount of excess collateral. Thus, in the low likelihood event that any Vaults become undercollateralized, the Dai holders receive a "haircut" to their claim on collateral. In other words, Dai holders' claim may be less than a dollar's worth of collateral.
- **Immediate Vault redemption** - After ES is initiated, Vault owners are allowed to free their collateral immediately, provided that they execute all contract calls atomically.
- **No off-chain calculations** - The system does not require the cage authority to supply any off-chain calculated values (e.g. it can rely entirely on the last OSM feed prices).
- **Vow Buffer Assistance** - After ES is initiated, any surplus (and bad debt) in the buffer acts as a reward (and penalty) distributed pro-rata to all Dai Holders. e.g. if 10% of total system debt is in the form of net surplus in the Vow, then Dai holders receive 10% more collateral.

Thanks to Vamsi

Emergency Shutdown - User Stories

Regardless of Vault delegation (`wish(owner, manager)`), Vault redemption will be performed by the owners. Most Dai holders are assumed to sell to secondary markets. Dai redemption will likely be completed by Redemption (or Super) Keepers that are better equipped to:

- Find liquidity for all collateral types
- Handle permissioned collateral that require KYC, such as tokenized securities

How will each agent type interact with the End Contract after Emergency Shutdown has been initiated?

- **Vault Owners of Single Ilk type**
- **Dai Holders**
- **Maker Protocol Stakeholders**

Preface to the following stories:

- All stories start with Step #0 `cage( )`, which is called by the Emergency Shutdown Module (ESM)
- Each contract call assumes that it has not been called already
- Undercollateralized Vaults = debt > collateral or `urn.art * ilk.rate > urn.ink * (ilk.spot * ilk.mat)`

This slide requires familiarity of Emergency Shutdown (ES). Due to its complexity, the design and operation of ES is [elaborated here]

Emergency Shutdown - User Stories cont'd

Vault Owners of single ilk type

I own a Vault and wish to free my excess collateral from the Maker Protocol.

Overcollateralized Vaults

1. Value the collateral in their Vault → `cage(ilk)`
2. Settle their outstanding debt with their locked collateral → `skim(ilk, urn)`
3. Unlock excess collateral in Vault → `free(ilk)`

Overcollateralized Vaults that were bitten and are in the "Tend" phase of the respective Collateral auction

1. Value the collateral in their Vault → `cage(ilk)`
2. Either wait until their tend auction phase finishes OR speed up collateral processing → `skip(ilk, id)`
3. Settle their outstanding debt with their locked collateral → `skim(ilk, urn)`
4. Unlock excess collateral in Vault → `free(ilk)`

Undercollateralized Vaults → not incentivized to do anything

This slide requires familiarity of Emergency Shutdown (ES). Due to its complexity, the design and operation of ES is [elaborated here]

Emergency Shutdown - User Stories cont'd

Dai holders

I want to convert my Dai into its claim on the underlying collateral basket.

1. Transition system into the Dai withdrawal phase → `thaw()`
2. Set the Dai-to-collateral exchange rate (`fix[ilk]`) for each collateral type → `flow(ilk)`
3. Send all their Dai to the Maker Protocol → `pack(wad)`
4. Retrieve a portion of each collateral type → `cash(ilk, wad)`

Maker Protocol Stakeholders (Larger Dai holders/custodians, MKR holders, Redemption keepers, etc.)

I want to ensure that emergency shutdown is completed and accounts for all Vaults. Cage-Keeper
Note that their story may overlap with the above stories.

1. Cancel all flop and flap auctions → `Flop.yank(id)` and `Flap.yank(id)` for multiple flop and flap auctions
2. Value the collateral in all Vault types → `cage(ilk)` for multiple collateral types
3. Either wait until their tend auction phase OR speed up collateral processing → `skip(ilk, id)` for multiple flip auctions over all collateral types
4. Settle all over/under collateralized Vaults → `skim(ilk, urn)` for multiple Vaults
5. Transition system into the Dai withdrawal phase → `thaw()`
6. Set the Dai-to-collateral exchange rate (`fix[ilk]`) for each collateral type → `flow(ilk)` for multiple collateral types

This slide requires familiarity of Emergency Shutdown (ES). Due to its complexity, the design and operation of ES is elaborated here

References

Maker Foundation Team

Code, readme, and wiki - https://github.com/makerdao/dss

K specification of smart contracts - https://github.com/makerdao/k-dss

Whitepaper - https://makerdao.com/whitepaper

The End

Thank you for taking the time to go through this presentation!

Please reach out on **Reddit**, **RocketChat**, or the **forum.makerdao.com**, if any questions arise

Submit errors to kenton@makerdao.com or @Kenton on Rocket Chat

EXHIBIT G

Transaction manager

The `transactionManager` service is used to track a transaction's status as it propagates through the blockchain.

Methods in Dai.js that start transactions are asynchronous, so they return promises. These promises can be passed as arguments to the transaction manager to set up callbacks when transactions change their status to `pending`, `mined`, `confirmed` or `error`.

```
const txMgr = maker.service('transactionManager');
// instance of transactionManager
const open = maker.service('cdp').openCdp();
// open is a promise--note the absence of `await`
```

Pass the promise to `transactionManager.listen` with callbacks, as shown below.

```
txMgr.listen(open, {
  pending: tx => {
    // do something when tx is pending
  },
  mined: tx => {
    // do something when tx is mined
  },
  confirmed: tx => {
    // do something when tx is confirmed
  },
  error: tx => {
    // do someting when tx fails
  }
});

await txMgr.confirm(open);
// 'confirmed' callback will fire after 5 blocks
```

Note that the `confirmed` event will not fire unless `transactionManager.confirm` is called. This async function waits a number of blocks (default 5) after the transaction has been mined to resolve. To change this globally, set the `confirmedBlockCount` attribute in Maker options. To change it for just one call, pass the number of blocks to wait as the second argument:

```
await txMgr.confirm(open, 3);
```

Transaction Metadata

There are functions such as `lockEth()` which are composed of several internal transactions. These can be more accurately tracked by accessing `tx.metadata` in the callback which contains both the `contract` and the `method` the internal transactions were created from.

Transaction Object Methods

A `TransactionObject` also has a few methods to provide further details on the transaction:

- `hash` : transaction hash
- `fees()` : amount of ether spent on gas
- `timeStamp()` : timestamp of when transaction was mined
- `timeStampSubmitted()` : timestamp of when transaction was submitted to the network

```
const lock = cdp.lockEth(1);
txMgr.listen(lock, {
  pending: tx => {
    const {contract, method} = tx.metadata;
    if(contract === 'WETH' && method === 'deposit') {
      console.log(tx.hash); // print hash for WETH.deposit
    }
  }
})
```

Maker Docs

Vault manager

The vault manager works with vaults that are owned by the CdpManager contract, which is also used by Oasis Borrow. This intermediary contract allows the use of incrementing integer IDs for vaults, familiar to users of Single-Collateral Sai, as well as other conveniences.

In the code, this is called CdpManager.

```
1 const mgr = maker.service('mcd:cdpManager');
```

Instance methods

The methods below are all asynchronous.

getCdpIds()

Return an array describing the vaults owned by the specified address. Note that if the vaults were created in Oasis Borrow, the address of the proxy contract should be used.

```
1 const proxyAddress = await maker.service('proxy').currentProxy();
2 const data = await mgr.getCdpIds(proxyAddress);
3 const { id, ilk } = data[0];
4 // e.g. id = 5, ilk = 'ETH-A'
```

getCdp()

Get an existing vault by its numerical ID. Returns a Vault instance.

```
1 const vault = await mgr.getCdp(111);
```

open()

Open a new vault with the specified collateral type. Will create a proxy if one does not already exist. Returns a Vault instance. Works with the transaction manager.

```
1 const txMgr = maker.service('transactionManager');
2 const open = await mgr.open('ETH-A');
3 txMgr.listen(open, {
4
```

```
5 });pending: tx => console.log('tx pending: ' + tx.hash)
6 const vault = await open;
```

openLockAndDraw()

Open a new vault, then lock and/or draw in a single transaction. Will create a proxy if one does not already exist. Returns a Vault instance.

```
1 const vault = await mgr.openLockAndDraw(
2   'BAT-A',
3   BAT(1000),
4   DAI(100)
5 );
```

Vault instances

In the code, these are called ManagedCdp.

Properties

A note on caching: When a vault instance is created, its data is pre-fetched from the blockchain, allowing the properties below to be read synchronously. This data is cached in the instance. To refresh this data, do the following:

```
1 vault.reset();
2 await vault.prefetch();
```

collateralAmount

The amount of collateral tokens locked, as a currency unit.

collateralValue

The USD value of collateral locked, given the current price according to the price feed, as a currency unit.

debtValue

The amount of Dai drawn, as a currency unit.

liquidationPrice

The USD price of collateral at which the Vault becomes unsafe.

isSafe

Whether the Vault is currently safe or not.

Instance methods

All of the methods below are asynchronous and work with the transaction manager. Amount arguments should be currency units, e.g.:

```
import { ETH, DAI } from '@makerdao/dai-plugin-mcd';

await vault.lockAndDraw(ETH(2), DAI(20));
```

lockCollateral(amount)

Deposit the specified amount of collateral.

drawDai(amount)

Generate the specified amount of Dai.

lockAndDraw(lockAmount, drawAmount)

Deposit some collateral and generate some Dai in a single transaction.

wipeDai(amount)

Pay back the specified amount of Dai.

wipeAll()

Pay back all debt. This method ensures that dust amounts do not remain.

freeCollateral(amount)

Withdraw the specified amount of collateral.

wipeAndFree(wipeAmount, freeAmount)

Pay back some debt and withdraw some collateral in a single transaction.

wipeAllAndFree(freeAmount)

Pay back all debt, ensuring dust amounts do not remain, and withdraw a specified amount of collateral in a single transaction.

give(address)

Transfer ownership of this vault to `address`. Note that if the new owner plans to use this vault with Oasis Borrow, it should be transferred to their proxy with giveToProxy instead.

giveToProxy(address)

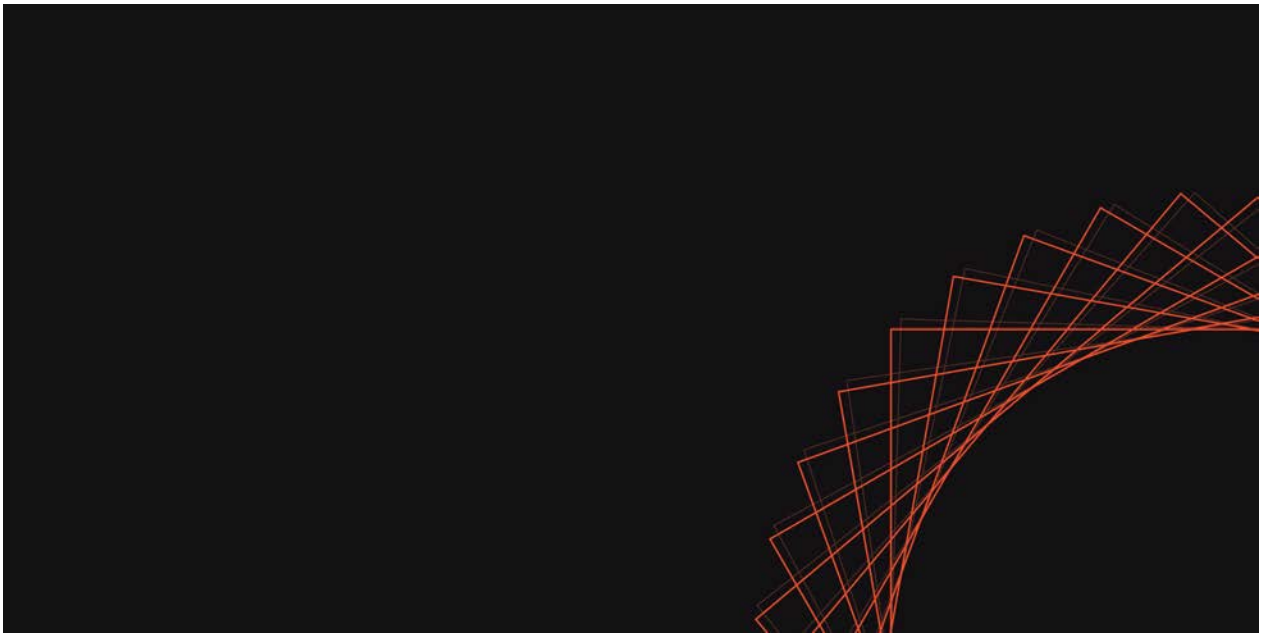
Look up the proxy contract owned by `address` and transfer ownership of this vault to that proxy.

EXHIBIT H

Back to Blog

Introducing Oracles V2 and DeFi Feeds

September 3, 2019



Last month, the Maker Foundation introduced **The Road to Multi-Collateral Dai (MCD) Roadmap**, which lists the critical milestones that must be achieved before MCD goes live. One milestone is the ratification of a set of governance proposals related to Oracles.

Oracles, collectively, are a mechanism to broadcast data from outside of the blockchain onto the blockchain. The Maker Protocol uses Oracles to obtain the real-time price of assets. This price is used to determine whether a CDP has enough collateral locked up.

This blog post introduces exciting changes to the Oracles infrastructure and Oracles governance. The aim of this post is to provide a stepping stone for MKR Governors that lays a foundation for discussing the Oracle proposals on the MakerDAO Governance Calls. The role of Oracles will be recapitulated and several new proposals will be put forward:

1. A proposal will be submitted to add a set of DeFi partners as Feeds
2. The proposal of an Oracle Team Mandate to create an Oracle Team role. The mandate will empower MKR Governance to appoint Oracle Team(s) to perform certain tasks on their behalf.
3. In the spirit of **gradual decentralization**, it will be proposed that MKR Governors' control of the Oracles infrastructure be formalized via an Oracle Governance Framework.
4. The Maker Foundation will propose a new incentive structure for Oracles.

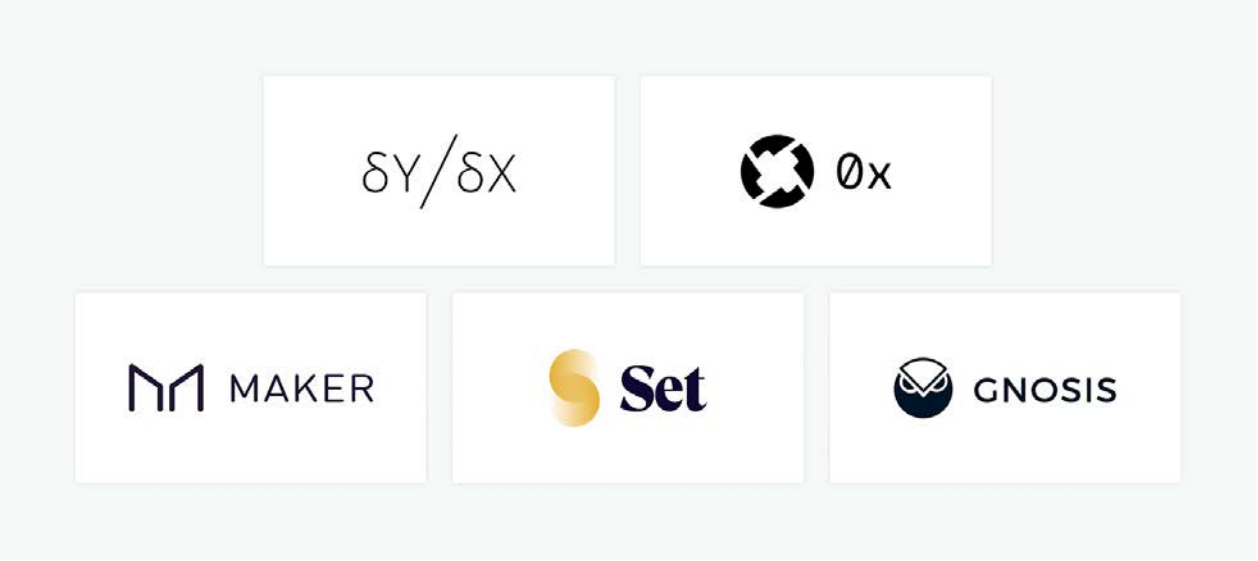
The Evolution of the Maker Oracles

In June 2017, with the introduction of Sai, the Maker Foundation released the first decentralized Oracles on the Ethereum mainnet. Since then, there has been widespread organic adoption of the Maker Oracles, fueling the growth of the emerging DeFi ecosystem. Today, the Maker Foundation is proud to announce the next generation of decentralized Oracles, Oracles V2. Utilizing the lessons learned over the past two years, Oracles V2 was built from the ground up to optimize for scalability, decentralization, resiliency, latency, and cost.

DeFi Partner Feeds

Feeds are critical components of the Oracle system. Feeds are bots, currently run by individuals, that publish the prices of assets in real time. The published prices are pooled together into a canonical price in a smart contract that can then be used by a decentralized application (dapp). The **MakerDAO Feed Dashboard** illustrates the prices from the 14 Feeds that currently make up the ETHUSD Oracle price used by Single-Collateral Dai.

From their onset, the individuals running the Feeds have been pseudonymous out of necessity, to protect the individuals from risk of extortion and blackmail. Organizations running Feeds, however, are different. Organizations are much more resilient against coercion, have the resources to combat malicious actors, and have their reputations at stake. This makes them much better equipped to be Feeds with public identities. It has become abundantly clear that a hybrid model is optimal—one which preserves the hardness properties of pseudonymous Feeds but benefits from the reputation of stakeholders in the ecosystem.



The first set of proposed organizations ready to run new Feeds in Oracles V2.

To this end, the Maker Foundation is happy to announce the first wave of organizations that are ready to run new Feeds in Oracles V2, pending voter approval: 0x, dYdX, Set Protocol, and Gnosis. They were chosen based on their reputation as leaders and stakeholders within the DeFi community.

The Maker Foundation wants to thank these organizations for coming together to create a truly decentralized Oracle with broad community support.

Governance

In order to decentralize the MakerDAO Oracles further, the release of Oracles V2 introduces significant changes to MKR Governance. Governance will become much

more active in end-to-end ownership and management of the Oracle infrastructure. To ease this transition, the Foundation plans to submit several new proposals to MKR holders (voters in the Maker community) on the public **Governance Call** on September 5. The proposals will then be published on the **MakerDAO Forum** on September 9, with a request for community feedback. Submission of the proposals to the **Governance Portal** for ratification is planned for September 16.

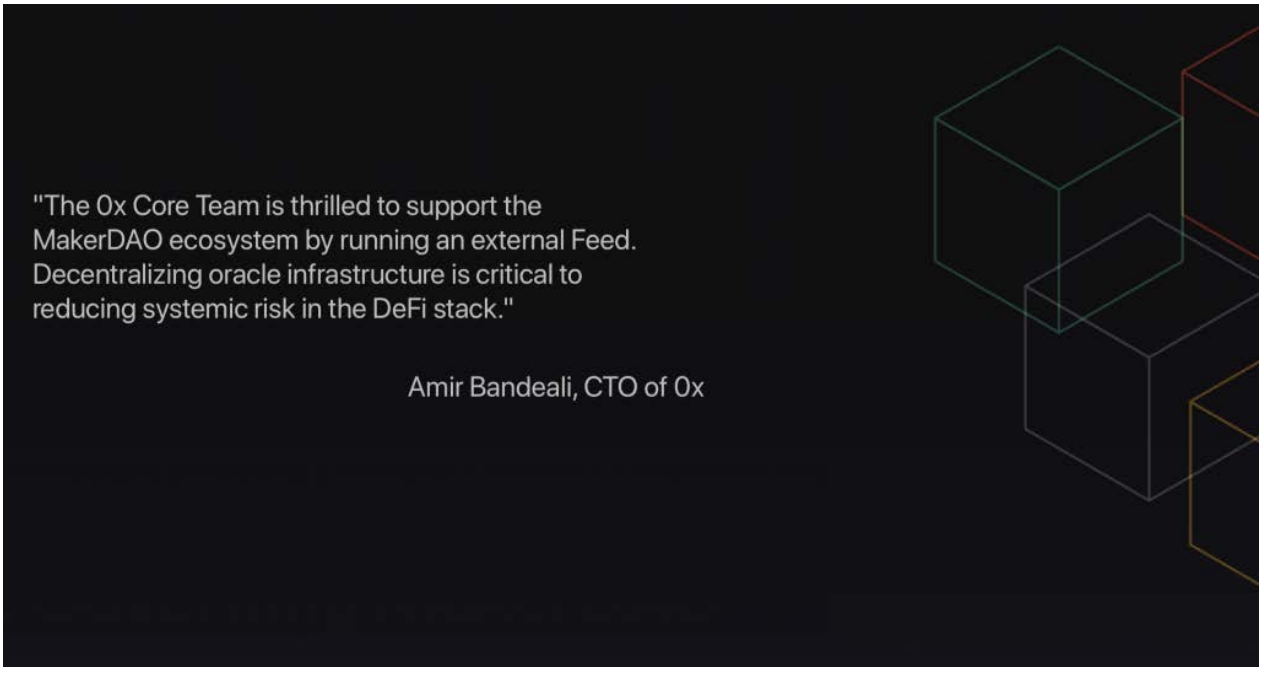
The titles for these proposals are:

- DeFi Partner Feeds (0x, dYdX, Set, Gnosis)
- Oracle Team Mandate
- Oracle Governance Framework
- Oracles Incentive Restructuring

Each proposal will shortly be described in the following.

DeFi Partner Feeds

This proposal will recommend the addition of 0x, dYdX, Set Protocol, and Gnosis as Feeds in Oracles V2.



Oracle Team Mandate (OTM)

The Oracle Team Mandate is a proposal defining the responsibilities of a delegated party chosen by Maker voters to facilitate administration and technical development of the Oracles. The Maker Foundation will propose to governance that its internal Oracle team will occupy this position in an interim capacity until a suitable external Oracle team is elected. It is important to note that the Oracle Team does not have any special privileges to enact changes without voter approval; it is merely a facilitation mechanism for helping to craft proposals and guide the governance process.

Oracle Governance Framework (OGF)

The Oracle Governance Framework is a proposal that defines the rights and responsibilities of Maker governance with respect to Oracles. A blueprint that specifies the governance processes to administer the Oracles infrastructure, it is similar to how the **Governance Risk Framework** defines the rules for how Maker Governance functions.

Some of the responsibilities of Maker Governors listed in the OGF are:

- Defining criteria for selecting new Feeds
- Defining criteria for selecting new Oracles
- Adding and removing Feeds
- Adding and removing Oracles
- Identifying performance metrics for Feeds and Oracles
- Selecting the Oracle price sensitivity parameters
- Selecting the Oracle Security Module (OSM) delay parameter

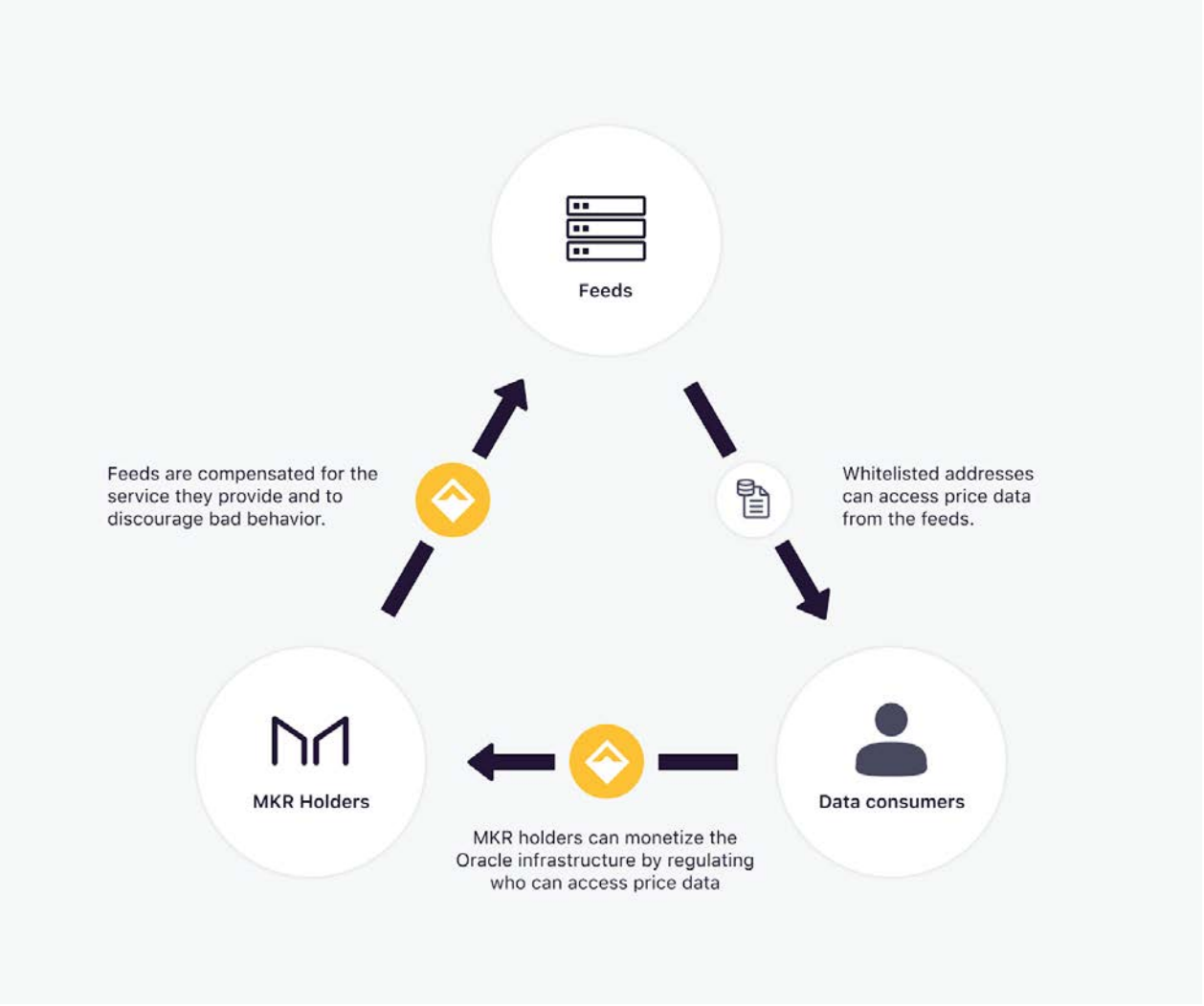
The full list of responsibilities and governance processes around Oracles will be included in the proposal and published on the **MakerDAO Forum** on September 9.

Oracles Incentives Restructuring

It is vital that the core components of the Maker Protocol are fundamentally decentralized. This ultimately means they are under the authority of Maker Governance and architected with a balanced incentive system to make them resilient

and self-sustainable. Oracles are no exception to this. Empowering the DAO to control the Oracles infrastructure through the Oracle Governance Framework is just the first step.

The Oracles Incentives Restructuring proposal recommends changes to the Oracles incentive system. The changes affect the tokenomics of the MKR token and render additional responsibilities on Maker Governance. Please take a look at the figure below.



Relationships between the different actors in the Oracle ecosystem.

Feeds are compensated for the valuable service they provide and, therefore, discourage malicious behavior. Currently, compensation is funded by the **Maker Development Fund**. In the future, payment is expected to be sourced from the

stability fees generated by the Maker Protocol, representing a transfer of value from MKR token holders to Feeds as shown in the figure above.

Note that Oracles V2 introduces a whitelist, which contains the addresses of smart contracts permitted to read Oracle prices. This enables MKR token holders to monetize the Oracle infrastructure in order to recoup their costs for running it.

This mechanism closes the loop between data producers (i.e., Feeds) and data consumers (i.e., dapps). In doing so, it provides the necessary incentive for Feeds to support prices for all kinds of assets rather than just the collateral types in the Maker Protocol.

The proposed Oracle Governance Framework details the governance process for an external entity to apply to be on an Oracle's whitelist. As MakerDAO should continue to take a leadership role in helping grow the Ethereum ecosystem, the Maker Foundation will submit a proposal permitting anyone to be added to the whitelist for any Oracle for free and for one year.

Join the Discussion

The Maker Foundation is excited to see the evolution of Oracles and their impact on fostering innovation. Oracles V2 brings us one step closer to Multi-Collateral Dai. To learn more about the Oracle proposals, join the public **Governance Call** on Thursday, September 5, at 16:00 UTC.

September 3, 2019

Read next

How New Blockchain Apps Make It Easier To Use DeFi

June 25, 2021

The Best Books on Crypto: 7 Recommendations

June 22, 2021

How and Why to Replace Your Ethereum Address with a User-Friendly Name

May 20, 2021

RESOURCES

Whitepaper

FAQs

Privacy Policy

Brand Assets

Feeds

PRODUCTS

Oasis
Migrate

Ecosystem

Governance

DEVELOPER

Documentation
Dai.js

Developer Guides

FOUNDATION

Contact

EXHIBIT I

Maker Docs

Oracle Security Module (OSM) - Detailed Documentation

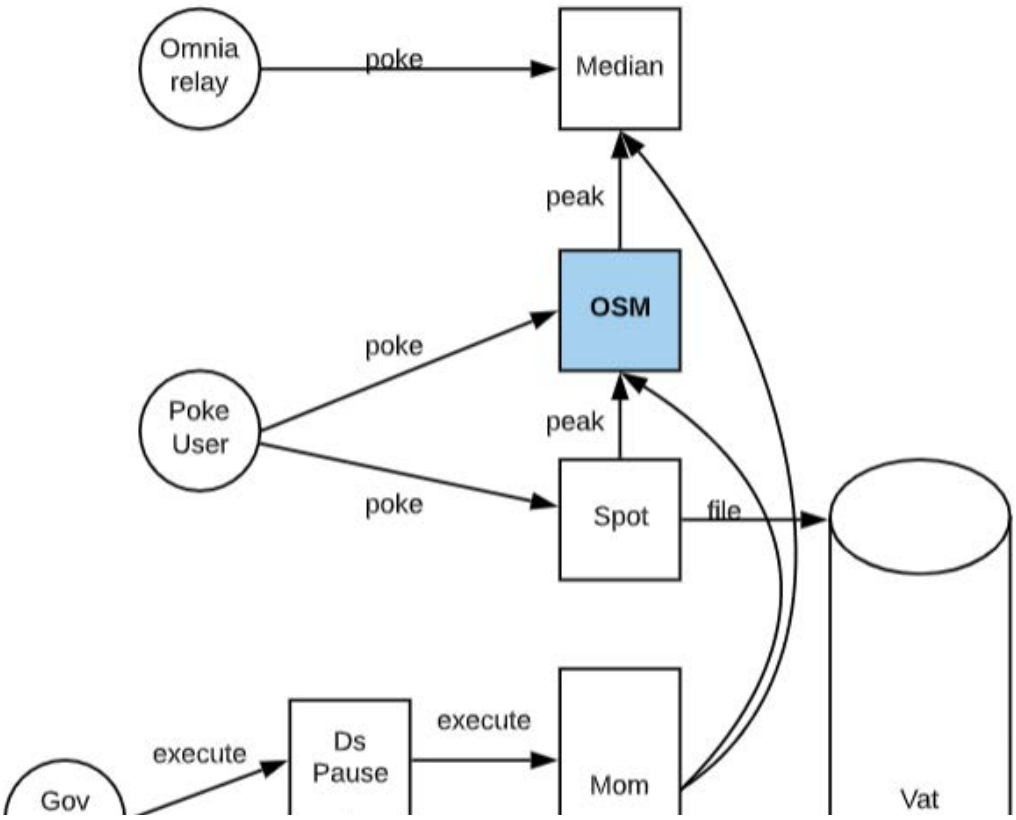
- **Contract Name:** OSM
- **Type/Category:** Oracles - Price Feed Module
- **Associated MCD System Diagram**
- **Contract Source**

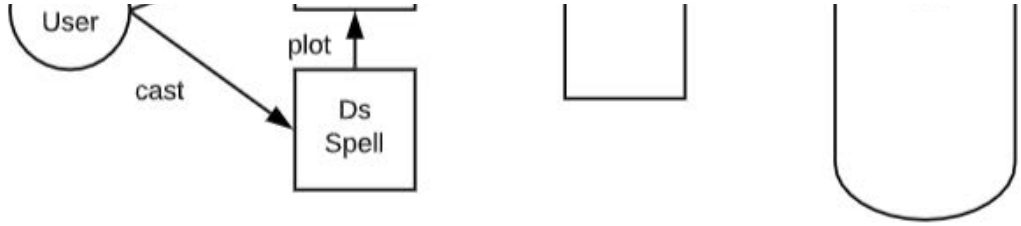
---

1. Introduction

Summary

The OSM (named via acronym from "Oracle Security Module") ensures that new price values propagated from the Oracles are not taken up by the system until a specified delay has passed. Values are read from a designated DSValue contract (or any contract that has the `read()` and `peek()` interfaces) via the `poke()` method; the `read()` and `peek()` methods will give the current value of the price feed, and other contracts must be whitelisted in order to call these. An OSM contract can only read from a single price feed, so in practice one OSM contract must be deployed per collateral type.





2. Contract Details - Glossary (OSM)

Storage Layout

- `stopped` : flag (`uint256`) that disables price feed updates if non-zero
- `src` : `address` of DSValue that the OSM will read from
- `ONE_HOUR` : 3600 seconds (`uint16(3600)`)
- `hop` : time delay between `poke` calls (`uint16`); defaults to `ONE_HOUR`
- `zzz` : time of last update (rounded down to nearest multiple of `hop`)
- `cur` : `Feed` struct that holds the current price value
- `nxt` : `Feed` struct that holds the next price value
- `bud` : mapping from `address` to `uint256`; whitelists feed readers

Public Methods

Administrative Methods

These functions can only be called by authorized addresses (i.e. addresses `usr` such that `wards[usr] == 1`).

- `rely` / `deny` : add or remove authorized users (via modifications to the `wards` mapping)
- `stop()` / `start()` : toggle whether price feed can be updated (by changing the value of `stopped`)
- `change(address)` : change data source for prices (by setting `src`)
- `step(uint16)` : change interval between price updates (by setting `hop`)
- `void()` : similar to `stop`, except it also sets `cur` and `nxt` to a `Feed` struct with zero values
- `kiss(address)` / `diss(address)` : add/remove authorized feed consumers (via modifications to the `buds` mapping)

Feed Reading Methods

These can only be called by whitelisted addresses (i.e. addresses `usr` such that `buds[usr] == 1`):

- `peek()` : returns the current feed value and a boolean indicating whether it is valid
- `peep()` : returns the next feed value (i.e. the one that will become the current value upon the next `poke()` call), and a boolean indicating whether it is valid
- `read()` : returns the current feed value; reverts if it was not set by some valid mechanism

Feed Updating Methods

- `poke()` : updates the current feed value and reads the next one

`Feed` struct: a struct with two `uint128` members, `val` and `has`. Used to store price feed data.

---

3. Key Mechanisms & Concepts

The central mechanism of the OSM is to periodically feed a delayed price into the MCD system for a particular collateral type. For this to work properly, an external actor must regularly call the `poke()` method to update the current price and read the next price. The contract tracks the time of the last call to `poke()` in the `zzz` variable (rounded down to the nearest multiple of `hop`; see Failure Modes for more discussion of this), and will not allow `poke()` to be called again until `block.timestamp` is at least `zzz+hop`. Values are read from a designated DSValue contract (its address is stored in `src`). The purpose of this delayed updating mechanism is to ensure that there is time to detect and react to an Oracle attack (e.g. setting a collateral's price to zero). Responses to this include calling `stop()` or `void()`, or triggering Emergency Shutdown.

Other contracts, if whitelisted, may inspect the `cur` value via the `peek()` and `read()` methods (`peek()` returns an additional boolean indicating whether the value has actually been set; `read()` reverts if the value has not been set). The `nxt` value may be inspected via `peep()`.

The contract uses a dual-tier authorization scheme: addresses mapped to 1 in `wards` may start and stop, set the `src`, call `void()`, and add new readers; addresses mapped to 1 in `buds` may call `peek()`, `peep()`, and `read()`.

---

4. Gotchas (Potential Sources of User Error)

Confusing `peek()` for `peep()` (or vice-versa)

The names of these methods differ by only a single character and in current linguistic usage, both "peek" and "peep" have essentially the same meaning. This makes it easy for a developer to confuse the two and call the wrong one. The effects of such an error are naturally context-dependent, but could e.g. completely invalidate the purpose of the OSM if the `peep()` is called where instead `peek()` should be used. A mnemonic to help distinguish them: "since 'k' comes before 'p' in the English alphabet, the value returned by `peek()` comes before the value returned by `peep()` in chronological order". Or: "`peek()` returns the **k**urrent value".

5. Failure Modes (Bounds on Operating Conditions & External Risk Factors)

`poke()` is not called promptly, allowing malicious prices to be swiftly uptaken

For several reasons, `poke()` is always callable as soon as `block.timestamp / hop` increments, regardless of when the last `poke()` call occurred (because `zzz` is rounded down to the nearest multiple of `hop`). This means the contract does not actually guarantee that a time interval of at least `hop` seconds has passed since the last `poke()` call before the next one; rather this is only (approximately) guaranteed if the last `poke()` call occurred shortly after the previous increase of `block.timestamp / hop`. Thus, a malicious price value can be acknowledged by the system in a time potentially much less than `hop`.

This was a deliberate design decision. The arguments that favoured it, roughly speaking, are:

- Providing a predictable time at which MKR holders should check for evidence of oracle attacks (in practice, `hop` is 1 hour, so checks must be performed at the top of the hour)
- Allowing all OSMs to be reliably poked at the same time in a single transaction

The fact that `poke` is public, and thus callable by anyone, helps mitigate concerns, though it does not eliminate them. For example, network congestion could prevent anyone from successfully calling `poke()` for a period of time. If an MKR holder observes that `poke` has not been promptly called, **the actions they can take include:**

1. Call `poke()` themselves and decide if the next value is malicious or not
2. Call `stop()` or `void()` (the former if only `nxt` is malicious; the latter if the malicious value is already in `cur`)
3. Trigger emergency shutdown (if the integrity of the overall system has already been compromised or if it is believed the rogue oracle(s) cannot be fixed in a reasonable length of time)

In the future, the contract's logic may be tweaked to further mitigate this (e.g. by **only** allowing `poke()` calls in a short time window each `hop` period).

Authorization Attacks and Misconfigurations

Various damaging actions can be taken by authorized individuals or contracts, either maliciously or accidentally:

- Revoking access of core contracts to the methods that read values, causing mayhem as prices fail to update
- Completely revoking all access to the contract
- Changing `src` to either a malicious contract or to something that lacks a `peek()` interface, causing transactions that `poke()` the affected OSM to revert

- Calling disruptive functions like `stop` and `void` inappropriately

The only solution to these issues is diligence and care regarding the `wards` of the OSM.

Maker Docs

Oracle Module

The Maker Protocol's Oracles

- **Module Name:** Oracle Module
- **Type/Category:** Oracles —> OSM.sol & Median.sol
- **Associated MCD System Diagram**
- **Contract Sources:**
  - **Median**
  - **OSM**

---

1. Introduction (Summary)

An oracle module is deployed for each collateral type, feeding it the price data for a corresponding collateral type to the `Vat`. The Oracle Module introduces the whitelisting of addresses, which allows them to broadcast price updates off-chain, which are then fed into a `median` before being pulled into the `OSM`. The `Spot` 'ter will then proceed to read from the `OSM` and will act as the liaison between the `oracles` and `dss`.

---

2. Module Details

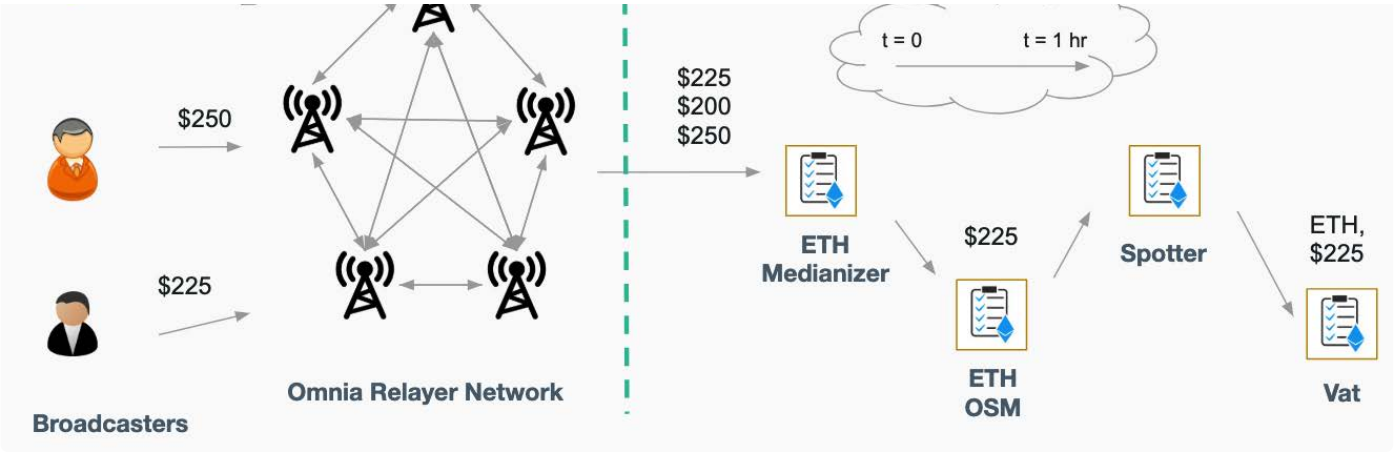
The Oracle Module has 2 core components consisting of the `Median` and `OSM` contracts.

Oracle Module Components Documentation

- **Median Documentation**
- **OSM Documentation**

---

3. Key Mechanism and Concepts



Interaction Diagram (Credit: MCD-101 Presentation, by Kenton Prescott)

Summary of the Oracle **Module Components**

- The `Median` provides Maker's trusted reference price. In short, it works by maintaining a whitelist of price feed contracts which are authorized to post price updates. Every time a new list of prices is received, the median of these is computed and used to update the stored value. The median has permissioning logic which is what enables the addition and removal of whitelisted price feed addresses that are controlled via governance. The permissioning logic allows governance to set other parameters that control the Median's behavior—for example, the `bar` parameter is the minimum number of prices necessary to accept a new median value.
- The `OSM` (named via acronym from "Oracle Security Module") ensures that new price values propagated from the Oracles are not taken up by the system until a specified delay has passed. Values are read from a designated DSValue contract (or any contract that implements the `read()` and `peek()` interface) via the `poke()` method; the `read()` and `peek()` methods will give the current value of the price feed, and other contracts must be whitelisted in order to call these. An OSM contract can only read from a single price feed, so in practice one OSM contract must be deployed per collateral type.

---

4. Gotchas (Potential sources of user error)

Relationship between the OSM and the Median:

- You can read straight from the median and in return, you would get a more real-time price. However, this depends on the cadence of updates (calls to poke).
- The OSM is similar but has a 1-hour price delay. It has the same process for reading (whitelist, auth, read and peek) as a median. The way the OSM works, is you cannot update it directly but you can `poke` it to go and read from something that also has the same structure (the `peek` method - in this case, its the median but you can set it to read from anything that conforms to the same interface).
- Whenever the OSM reads from a source, it queues the value that it reads for the following hour or following `hop` property, which is set to 1 hour (but can be anything). When it is `poke`'d, it reads the value of the median and it will save the value. Then the previous value becomes that, so it is always off by an hour. After an hour passes, when `poke`d, the value that it saved becomes the current value and

whatever value is in the median becomes the future value for the next hour.

- `spot` - if you poke it with an ilk (ex: ETH) it will read form the OSM and if the price is valid, it updates.

Relationship to the `Spot` 'ter:

- In relation to the `Spot` the oracle module handles how market prices are recorded on the blockchain. The `Spot` ter operates as the interface contract, which external actors can use to retrieve the current market price from the Oracle module for the specified collateral type. The `Vat` in turn reads the market price from the `spot` ter.

---

5. Failure Modes (Bounds on Operating Conditions & External Risk Factors)

- `Median` - there is currently no way to turn off the oracle (failure or returns false) if all the oracles come together and sign a price of zero. This would result in the price being invalid and would return false on `peek`, telling us to not trust the value.
- `OSM`
  - `poke()` is not called promptly, allowing malicious prices to be swiftly uptaken.
  - Authorization Attacks and Misconfigurations.
  - Read more here.

Oracle Security Module (OSM) - Detailed Documentation

- **Contract Name:** OSM
- **Type/Category:** Oracles - Price Feed Module
- **Associated MCD System Diagram**
- **Contract Source**

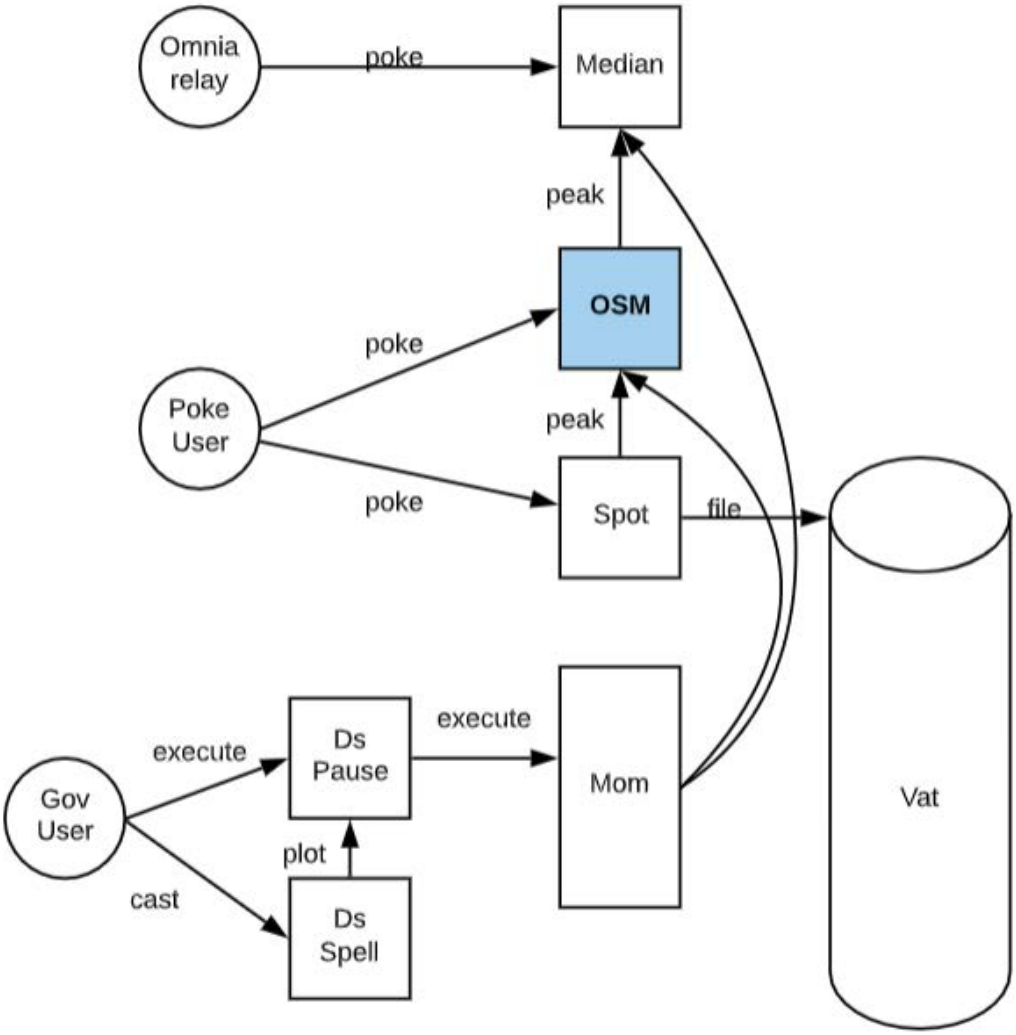
---

1. Introduction

Summary

The OSM (named via acronym from "Oracle Security Module") ensures that new price values propagated from the Oracles are not taken up by the system until a specified delay has passed. Values are read from a designated DSValue contract (or any contract that has the `read()` and `peek()` interfaces) via the

`poke()` method; the `read()` and `peek()` methods will give the current value of the price feed, and other contracts must be whitelisted in order to call these. An OSM contract can only read from a single price feed, so in practice one OSM contract must be deployed per collateral type.



2. Contract Details - Glossary (OSM)

Storage Layout

- `stopped` : flag (`uint256`) that disables price feed updates if non-zero
- `src` : `address` of DSValue that the OSM will read from
- `ONE_HOUR` : 3600 seconds (`uint16(3600)`)
- `hop` : time delay between `poke` calls (`uint16`); defaults to `ONE_HOUR`
- `zzz` : time of last update (rounded down to nearest multiple of `hop`)

- `cur` : `Feed` struct that holds the current price value
- `nxt` : `Feed` struct that holds the next price value
- `bud` : mapping from `address` to `uint256`; whitelists feed readers

Public Methods

Administrative Methods

These functions can only be called by authorized addresses (i.e. addresses `usr` such that `wards[usr] == 1`).

- `rely` / `deny` : add or remove authorized users (via modifications to the `wards` mapping)
- `stop()` / `start()` : toggle whether price feed can be updated (by changing the value of `stopped`)
- `change(address)` : change data source for prices (by setting `src`)
- `step(uint16)` : change interval between price updates (by setting `hop`)
- `void()` : similar to `stop`, except it also sets `cur` and `nxt` to a `Feed` struct with zero values
- `kiss(address)` / `diss(address)` : add/remove authorized feed consumers (via modifications to the `buds` mapping)

Feed Reading Methods

These can only be called by whitelisted addresses (i.e. addresses `usr` such that `buds[usr] == 1`):

- `peek()` : returns the current feed value and a boolean indicating whether it is valid
- `peep()` : returns the next feed value (i.e. the one that will become the current value upon the next `poke()` call), and a boolean indicating whether it is valid
- `read()` : returns the current feed value; reverts if it was not set by some valid mechanism

Feed Updating Methods

- `poke()` : updates the current feed value and reads the next one

`Feed` struct: a struct with two `uint128` members, `val` and `has`. Used to store price feed data.

---

3. Key Mechanisms & Concepts

The central mechanism of the OSM is to periodically feed a delayed price into the MCD system for a particular collateral type. For this to work properly, an external actor must regularly call the `poke()` method to update the current price and read the next price. The contract tracks the time of the last call to `poke()` in the `zzz` variable (rounded down to the nearest multiple of `hop`; see Failure Modes for more discussion of this), and will not allow `poke()` to be called again until `block.timestamp` is at least `zzz+hop`.

Values are read from a designated DSValue contract (its address is stored in `src`). The purpose of this delayed updating mechanism is to ensure that there is time to detect and react to an Oracle attack (e.g. setting a collateral's price to zero). Responses to this include calling `stop()` or `void()`, or triggering Emergency Shutdown.

Other contracts, if whitelisted, may inspect the `cur` value via the `peek()` and `read()` methods (`peek()` returns an additional boolean indicating whether the value has actually been set; `read()` reverts if the value has not been set). The `nxt` value may be inspected via `peep()`.

The contract uses a dual-tier authorization scheme: addresses mapped to 1 in `wards` may start and stop, set the `src`, call `void()`, and add new readers; addresses mapped to 1 in `buds` may call `peek()`, `peep()`, and `read()`.

---

4. Gotchas (Potential Sources of User Error)

Confusing `peek()` for `peep()` (or vice-versa)

The names of these methods differ by only a single character and in current linguistic usage, both "peek" and "peep" have essentially the same meaning. This makes it easy for a developer to confuse the two and call the wrong one. The effects of such an error are naturally context-dependent, but could e.g. completely invalidate the purpose of the OSM if the `peep()` is called where instead `peek()` should be used. A mnemonic to help distinguish them: "since 'k' comes before 'p' in the English alphabet, the value returned by `peek()` comes before the value returned by `peep()` in chronological order". Or: "`peek()` returns the **k**urrent value".

---

5. Failure Modes (Bounds on Operating Conditions & External Risk Factors)

`poke()` is not called promptly, allowing malicious prices to be swiftly uptaken

For several reasons, `poke()` is always callable as soon as `block.timestamp / hop` increments, regardless of when the last `poke()` call occurred (because `zzz` is rounded down to the nearest multiple of `hop`). This means the contract does not actually guarantee that a time interval of at least `hop` seconds has passed since the last `poke()` call before the next one; rather this is only (approximately) guaranteed if the last `poke()` call occurred shortly after the previous increase of `block.timestamp / hop`. Thus, a malicious price value can be acknowledged by the system in a time potentially much less than `hop`.

This was a deliberate design decision. The arguments that favoured it, roughly speaking, are:

- Providing a predictable time at which MKR holders should check for evidence of oracle attacks (in practice, `hop` is 1 hour, so checks must be performed at the top of the hour)
- Allowing all OSMs to be reliably poked at the same time in a single transaction

The fact that `poke` is public, and thus callable by anyone, helps mitigate concerns, though it does not eliminate them. For example, network congestion could prevent anyone from successfully calling `poke()` for a period of time. If an MKR holder observes that `poke` has not been promptly called, **the actions they can take include:**

1. Call `poke()` themselves and decide if the next value is malicious or not
2. Call `stop()` or `void()` (the former if only `nxt` is malicious; the latter if the malicious value is already in `cur`)
3. Trigger emergency shutdown (if the integrity of the overall system has already been compromised or if it is believed the rogue oracle(s) cannot be fixed in a reasonable length of time)

In the future, the contract's logic may be tweaked to further mitigate this (e.g. by **only** allowing `poke()` calls in a short time window each `hop` period).

Authorization Attacks and Misconfigurations

Various damaging actions can be taken by authorized individuals or contracts, either maliciously or accidentally:

- Revoking access of core contracts to the methods that read values, causing mayhem as prices fail to update
- Completely revoking all access to the contract
- Changing `src` to either a malicious contract or to something that lacks a `peek()` interface, causing transactions that `poke()` the affected OSM to revert
- Calling disruptive functions like `stop` and `void` inappropriately

The only solution to these issues is diligence and care regarding the `wards` of the OSM.

Median - Detailed Documentation

The Maker Protocol's trusted reference price

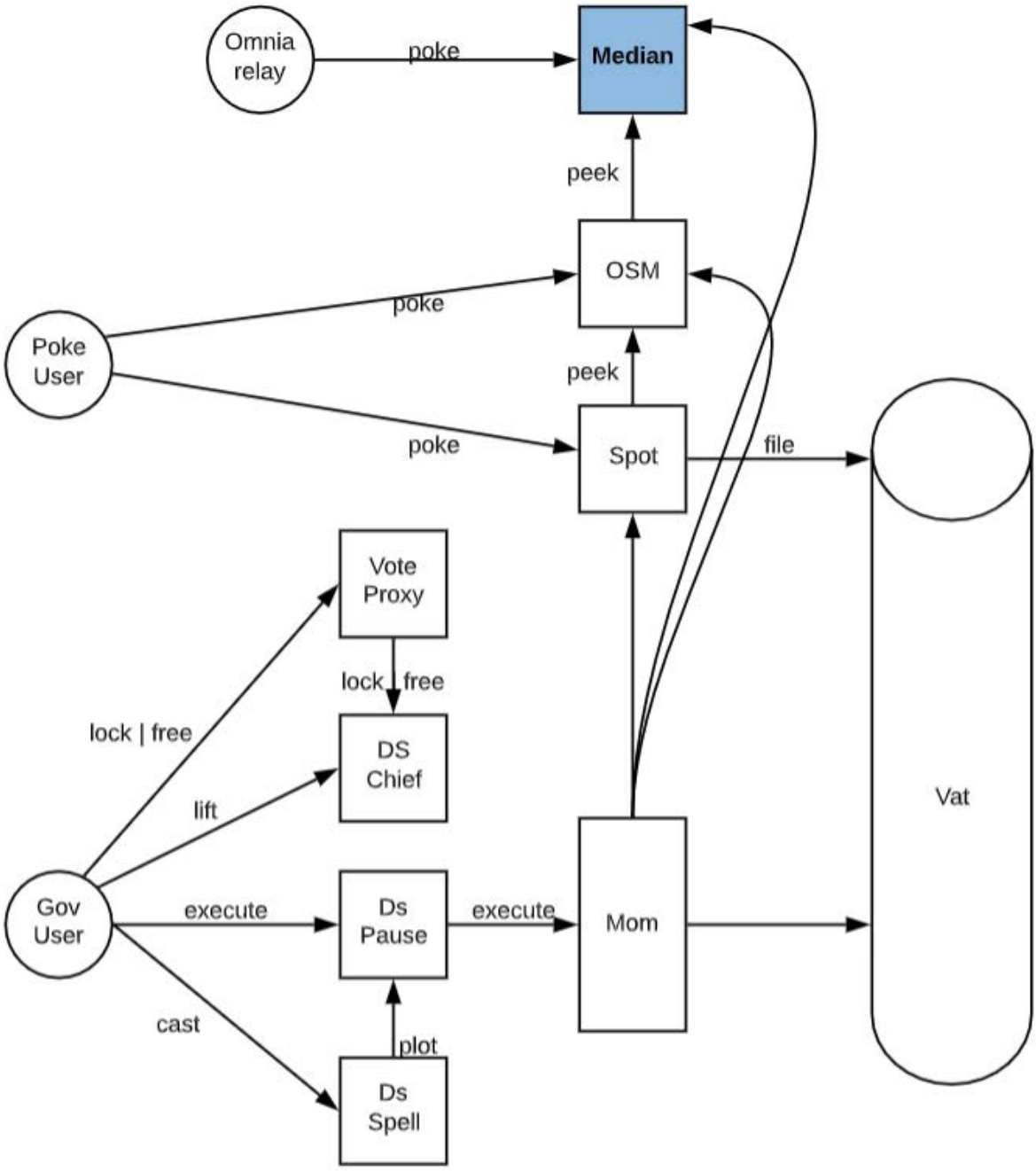
- **Contract Name:** median.sol
- **Type/Category:** Oracles Module
- **Associated MCD System Diagram**
- **Contract Source**

---

1. Introduction (Summary)

The median provides Maker's trusted reference price. In short, it works by maintaining a whitelist of price feed contracts which are authorized to post price updates. Every time a new list of prices is received, the median of these is computed and used to update the stored value. The median has permissioning logic which is what enables the addition and removal of whitelisted price feed addresses that are controlled via governance. The permissioning logic allows governance to set other parameters that control the Median's behavior—for example, the `bar` parameter is the minimum number of prices necessary to accept a new median value.

A High-level overview diagram of the components that involve and interact with the median:



Note: All arrows without labels are governance calls.

2. Contract Details

Median (Glossary)

Key Functionalities (as defined in the smart contract)

- `read` - Gets a non-zero price or fails.
- `peek` - Gets the price and validity.
- `poke` - Updates price from whitelisted providers.
- `lift` - Adds an address to the writers whitelist.
- `drop` - Removes an address from the writers whitelist.
- `setBar` - Sets the `bar`.
- `kiss` - Adds an address to the reader's whitelist.
- `diss` - Removes an address from the readers whitelist.

Note: `read` returns the `value` or fails if it's invalid & `peek` gives back the `value` and if the `value` is valid or not.

Other

- `wards(usr: address)` - Auth mechanisms.
- `orcl(usr: address)` - `val` writers whitelist / signers of the prices (whitelisted via governance / the authorized parties).
- `bud(usr: address)` - `val` readers whitelist.
- `val` - the price (private) must be read with `read()` or `peek()`
- `age` - the Block timestamp of last price `val` update.
- `wat` - the price oracles type (ex: ETHUSD) / tells us what the type of asset is.
- `bar` - the Minimum writers quorum for `poke` / min number of valid messages you need to have to update the price.

---

3. Key Mechanisms & Concepts

As mentioned above, the `median` is the smart contract that provides Maker's trusted reference price. Authorization **(auth)** is a key component included in the mechanism of this contract and its interactions. For example, the price (`val`) is intentionally kept not public because the intention is to only read it from the two functions `read` and `peek`, which are whitelisted. This means that you need to be authorized, which is completed through the `bud`. The `bud` is modified to get whitelisted authorities to read it on-chain (permissioned), whereas, everything of off-chain is public.

The `poke` method is not under any kind of `auth`. This means that anybody can call it. This was designed for the purpose of getting Keepers to call this function and interact with Auctions. The only way to modify its

state is if you call it and send it valid data. For example, let's say this oracle needs 15 different sources. This means that we would need it to send 15 different signatures. It will then proceed to go through each of them and validate that whoever sent the the data has been `auth`'d to do so. In the case of it being an authorized oracle, it will check if it signed the message with a timestamp that is greater than the last one. This is done for the purpose of ensuring that it is not a stale message. The next step is to check for order values, this requires that you send everything in an array that is formatted in ascending order. If not sent in the correct order (ascending), the median is not calculated correctly. This is because if you assume the prices are ordered, it would just grab the middle value which may not be sufficient or work. In order to check for uniqueness, we have implemented the use of a `bloom` filter. In short, a bloom filter is a data structure designed to tell us, rapidly and memory-efficiently, whether an element is present in a set. This use of the bloom filter helps with optimization. In order to whitelist signers, the first two characters of their addresses (the first `byte`) have to be unique. For example, let's say that you have 15 different price signers, none of the first two characters of their addresses can be the same. This helps to filter that all 15 signers are different.

Next, there are `lift` functions. These functions tell us who can sign messages. Multiple messages can be sent or it can just be one but they are put into the authorized oracle). However, there is currently nothing preventing someone from `lift`'ing two prices signers that start with the same address. This is something for example, that governance needs to be aware of (see an example of what a governance proposal would look like in this case in the **Gotchas** section).

Due to the mechanism design of how the oracles work, the **quorum** has to be an odd number. If it is an even number, it will not work. This was designed as an optimization (`val = uint128(val_[val_.length >> 1]);`); this code snippet outlines how it works, which is by taking the array of values (all the prices that each of the prices signers reported, ordered from 200-215) and then grabbing the one in the middle. This is done by taking the length of the array (15) and shifting it to the right by 1 (which is the same as dividing by 2). This ends up being 7.5 and then the EVM floors it to 7. If we were to accept even numbers this would be less efficient. This presents the issue that you should have a defined balance between how many you require and how many signers you actually have. For example, let's say the oracle needs 15 signatures, you need at least 17-18 signers because if you require 15 and you only have 15 and one of them goes down, you have no way of modifying the price, so you should always have a bit more. However, you should not have too many, as it could compromise the operation.

---

4. Gotchas

Emergency Oracles

- They can shutdown the price feed but cannot bring it back up. Bringing the price feed back up requires governance to step in.

Price Freeze

- If you void the oracles Ethereum module, the idea is that you cannot interact with any Vault that depends on that ilk.
    - **Example:** ETHUSD shutdown (can still add collateral and pay back debt - increases safety) but you

cannot do anything that increases risk (decreases safety - remove collateral, generate dai, etc.) because the system would not know if you would be undercollateralized.

Oracles Require a lot of Upkeep

- They need to keep all relayers functioning.
- The community would need to self-police (by looking at each price signer, etc.) if any of them needs to be replaced. They would need to make sure they are constantly being called every hour (for every hour, a transaction gets sent to the OSM, which means that a few transactions have already been sent to the median to update it as well. In addition, there would need to be a transaction sent to the `spotter`, as DSS operates in a pool-type method (doesn't update the system/write to it, you tell it to read it from the OSM).

There is nothing preventing from `lift`'ing two prices signers that start with the same address

- The only thing that this prevents is that you cannot have more than 256 oracles but we don't expect to ever have that many, so it is a hard limit. However, Governance needs to be sure that whoever they are voting in anyone that they have already voting in before with the same two first characters.
- An example of what a governance proposal would look like in this case:
  - We are adding a new oracle and are proposing (the Foundation) a list of signers (that have been used in the past) and we already have an oracle but want to add someone new (e.g. Dharma or dydx). We would say that they want to be price signers, so these are their addresses and we want to lift those two addresses. They would vote for that, and we would need to keep a list of the already existing addresses and they would need to create an address that doesn't conflict with the existing ones.

---

5. Failure Modes (Bounds on Operating Conditions & External Risk Factors)

- By design, there is currently no way right now to turn off the oracle (failure or returns false) if all the oracles come together and sign a price of zero. This would result in the price being invalid and would return false on `peek`, telling us to not trust the value.
  - We are currently researching (Oracles ETH module) that would invalidate the price but there is no way to do this in the median today. This is due to the separation of concerns that DSS does not read directly from median, it reads from the OSM, but this may end up changing.

EXHIBIT J

uniswap-price-feed

This repository contains a standalone service for publishing Uniswap prices over Websockets.

Rationale

This service facilitates distribution of real-time Uniswap prices to keeper bots from the `market-maker-keeper` https://github.com/makerdao/market-maker-keeper repository. Prices are retrieved from chain every second. The average of last 60 prices are reported to subscribed clients.

Installation

This project uses *Python 3.6.6* and requires *virtualenv* to be installed.

In order to clone the project and install required third-party packages please execute:

```
git clone https://github.com/makerdao/uniswap-price-feed.git
cd uniswap-price-feed
git submodule update --init --recursive
./install.sh
```

Running

```
usage: uniswap-price-feed [-h] [--rpc-host RPC_HOST] [--rpc-port RPC_PORT]
                          [--rpc-timeout RPC_TIMEOUT]
                          [--http-address HTTP_ADDRESS]
                          [--http-port HTTP_PORT]
                          [--base-exchange-address BASE_EXCHANGE_ADDRESS]
                          [--base-token-symbol BASE_TOKEN_SYMBOL]
                          [--base-token-address BASE_TOKEN_ADDRESS]
                          --quote-exchange-address QUOTE_EXCHANGE_ADDRESS
                          --quote-token-symbol QUOTE_TOKEN_SYMBOL
                          --quote-token-address QUOTE_TOKEN_ADDRESS
                          [--report-time REPORT_TIME]
                          [--ro-account RO_ACCOUNT]

optional arguments:
  -h, --help            show this help message and exit
  --rpc-host RPC_HOST   JSON-RPC host (default: `localhost')
  --rpc-port RPC_PORT   JSON-RPC port (default: `8545')
  --rpc-timeout RPC_TIMEOUT
                        JSON-RPC timeout (in seconds, default: 10)
  --http-address HTTP_ADDRESS
                        Address of the Uniswap Price Feed
  --http-port HTTP_PORT
                        Port of the Uniswap Price Feed
  --base-exchange-address BASE_EXCHANGE_ADDRESS
                        Address of the Uniswap Exchange
  --base-token-symbol BASE_TOKEN_SYMBOL
                        Token symbol
  --base-token-address BASE_TOKEN_ADDRESS
                        Token address
  --quote-exchange-address QUOTE_EXCHANGE_ADDRESS
                        Address of the Quote Uniswap Exchange
  --quote-token-symbol QUOTE_TOKEN_SYMBOL
                        Quote Token symbol
  --quote-token-address QUOTE_TOKEN_ADDRESS
                        Quote Token address
  --report-time REPORT_TIME
                        Time interval to report price
  --ro-account RO_ACCOUNT
                        Credentials of the read-only user (format:
                        username:password)
```

Sample scripts

- ETH-DAI price feed (e.g. pass `--price-feed` command line argument to keeper bot as `ws://user:readonly@localhost:7777/price/ETH-DAI/socket`)

```
bin/uniswap-price-feed \
    --quote-exchange-address 0x09cabEC1eAd1c0Ba254B09efb3EE13841712bE14 \
    --quote-token-symbol DAI \
    --quote-token-address 0x89d24A6b4CcB1B6fAA2625fE562bDD9a23260359 \
    --ro-account user:readonly
```

- MKR-DAI price feed (e.g. pass `--price-feed` command line argument to keeper bot as `ws://user:readonly@localhost:7778/price/MKR-DAI/socket`):

```
bin/uniswap-price-feed \
    --base-exchange-address 0x2C4Bd064b998838076fa341A83d007FC2FA50957 \
    --base-token-symbol MKR \
    --base-token-address 0x9f8F72aA9304c8B593d555F12eF6589cC3A579A2 \
    --quote-exchange-address 0x09cabEC1eAd1c0Ba254B09efb3EE13841712bE14 \
    --quote-token-symbol DAI \
    --quote-token-address 0x89d24A6b4CcB1B6fAA2625fE562bDD9a23260359 \
    --ro-account user:readonly \
    --http-port 7778 \
    --report-time 2
```

API

The primary and only entity this service operates on is *feed*. Each feed is effectively a stream of timestamped records. Timestamps never go back and it is always guaranteed that new records will be added 'after' the existing ones. This simplification makes feed streams consumption much easier for clients.

Each record is represented throughout the service as a JSON structure with two fields: `timestamp` and `data`. The first one is a UNIX epoch timestamp represented as a number (either integer or floating-point). The latter can be basically anything. Sample record may look as follows:

```
{
    "data": {
        "price": 173.03457395327663
    },
    "timestamp": 1571747588
}
```

All endpoints require and support only HTTP Basic authentication. Only one type of credentials is supported at the moment: (`--ro-account`) gives read-only access to the feeds.

`ws://<service-location>/price/<feed-name>/socket`

Opens a new socket subscription to a feed. Each new subscriber will immediately receive the last record from the feed, and will be promptly sent any new records posted by producer(s). Subscribers can assume that timestamps of records received over the WebSocket will always increase.

This is a receive-only WebSocket. Any messages sent by consumers to the service will be ignored.