Hedera Hashgraph

TOKENIZATION ON HEDERA

CONTENTS

INTRODUCTION

Motivation

This paper provides an overview of the two primary models Hedera Hashgraph supports for tokenization – natively on Hedera, using the new Hedera Token Service and in a permissioned network setting, using Hedera Consensus Service. Tokenization on Hedera introduces the technical fundamentals of each and aims to assist token issuers with determining the deployment model most appropriate for their use case.

AN OVERVIEW OF TOKENIZATION

To date, the primary use of blockchain and distributed ledger technology has been the issuance and exchange of tokens. Tokenization is being readily explored across enterprises and decentralized protocols for securities, stablecoins, digital art, and more. For each category, the act of tokenization has resulted in enhanced market efficiencies introduced through the their transparency and ability for all to frictionlessly participate resulting in increased liquidity.

Tokenization, or the process of converting real assets and ownership rights into digital assets (tokens), recorded on a distributed ledger, has the potential for underlying a mega-shift across how multiple trillion dollar markets operate. Ultimately, the success and continued rise of tokenization will rely on public networks supporting the performance, cost, and compliance required to achieve mainstream adoption. Hedera Hashgraph solves these limitations through the introduction of the Hedera Token Service to now offer two token deployment models.

While we will dive deeper into each of Hedera's tokenization models, a snapshot of them and their key differences is as follows:

	HEDERA TOKEN SERVICE Tokens natively on Hedera	**HEDERA CONSENSUS SERVICE** Tokens on a permissioned network with public trust
STORAGE	Hedera public ledger	Permissioned blockchain or database
PRIVACY	Pseudonymous	Public or encrypted
GOVERNANCE	Hedera Governing Council	Custom
CUSTOMIZATION	Limited	Customizable logic and roles

[1] https://coincentral.com/crypto-coin-vs-token-cryptocurrency/#:~:text=The%20term%20coin%20generally%20refers,top%20of%20an%20existing%20blockchain.&text=In%20contrast%2C%20tokens%20represent%20a,added%20to%20an%20existing%20infrastructure.

This section presents key concepts, use cases, and ecosystem participants necessary for the mass adoption of tokens across industries.

Key Concepts

It is first helpful to understand the key components of a token implementation. These concepts define a token and its usage, which we will map to Hedera's two tokenization models.

TOKEN CONTRACT

A Token Contract refers to the code which defines the roles and behaviors of a token. The contract is typically programmed by the token creator and is made available for review by third parties either through open sourcing or through deployment as a smart contract to a public network. This enables the token's implementation to be verifiable without the need to trust a "black box" created by a third party.

In the context of Hedera, the equivalent of a Token Contract differs based on the deployment model:

- Hedera Token Service the Token Contract is part of the token definition
- Hedera Consensus Service equivalent is defined in the application logic run by a set of permissioned nodes.

STATE

Once a token contract has been developed and deployed it creates a token which can be transferred between different accounts, often connected to some business process or other event.

The balance of each account and the history of transactions is reflected in the state of accounts holding the token.

- Hedera Token Service state is kept on the public ledger, on the Hedera mainnet nodes
- Hedera Consensus Service state is stored on the permissioned network nodes, staying in sync via HCS messages in a defined topic

ACCOUNTS

Accounts, reflected in the state as some unique identifier like a public key or user ID, hold balances of tokens which they can send and receive. Accounts can be controlled by individuals, institutions, or even computer code. Accounts can send and receive tokens for purposes relating to the token use case which could include exchanging for access to a product or service, as a means of payment, or transferring of ownership.

- Hedera Token Service uses Hedera accounts managed by the Hedera mainnet
- Hedera Consensus Service uses a custom account identifier, typically represented by a public key

Token Types

FUNGIBLE TOKENS VS. NON-FUNGIBLE TOKENS

Tokens of a common type can either be indistinguishable (fungible) or unique (non-fungible). Fungible tokens like Central Bank Digital Currencies, stablecoins, or governance tokens are interchangeable with one another, much in the same way currency is fungible (although currency is still identified with a unique ID number). Implementations of fungible tokens are frequently called account-based models where each account holds a balance of fungible tokens. Fungible token use cases are typically associated with higher throughput use cases.

Non-fungible tokens, or NFTs, like a digital collectible or real-estate asset, are distinguishable from one another. Each token can have a unique name or serial number which allows it to be distinguished from another token. Each token is most commonly connected to some digital or real world item. Non-fungible tokens are often implemented using what is referred to as a token-based model where each token is uniquely identified and is associated with a specified owner. The use cases are typically associated with lower throughput but potentially higher-value off-chain or digital assets.

USE CASES

There are a wide ranging and ever growing number of token use cases in the market today. These cover everything from regulated securities to decentralized governance tokens. This section will present a few such examples which helped to drive requirements for the Hedera Token Service.

Utility Tokens

Provide the holder access to some product or service. These can be technical services, similar to cloud credits, or real world products and services like access to a property for vacation.[2]

Security Tokens

Trading of the token represents trading of that value. Security tokens could represent shares in a company, or ownership of a property. Most notably these tokens must comply with relevant securities and other financial laws and derive their value from some underlying asset.

[2] https://blockgeeks.com/guides/utility-tokens-vs-security-tokens/

TOKEN DEPLOYMENT MODELS

Hedera Token Service

Tokenization options on most public cryptocurrency networks force issuers to cope with high and fluctuating costs, low transaction speeds, and the constant threat of network forking. This often limits token issuance to proofs of concept and prevents enterprise or consumer scale.

With Hedera Token Service (HTS), Hedera provides the ability to issue tokens on a globally distributed public network without compromising on performance. Developers can now define and issue tokens directly to the Hedera mainnet. Tokens inherit many of the characteristics of hbar itself, including asynchronous byzantine fault tolerant (ABFT) consensus, thousands of transactions per second, and finality in a matter of seconds without a risk of forking.

HTS, provided with a publicly accessible set of APIs, ensures that stablecoins, security tokens, governance, and more issued on the Hedera mainnet can be accessed by anyone through a Hedera Account. Users can frictionlessly transfer tokens between one another without the threat of a massive spike in cost disrupting their commerce nor reliance on intermediary.

Tokens issued with HTS will form the basis for peer to peer and business to business economic activity across industries and geographies on a globally distributed Hedera mainnet.

HTS was designed based on user requirements from existing token issuers and interested enterprises. Work done by groups such as the Interwork Alliance, of which Hedera is a founding member, provided the taxonomy and sample behaviors to consider for HTS. The requirements were vetted with the Hedera Governing Council, token issuers, ecosystem members, and security experts.

HTS API OVERVIEW

The Hedera API (HAPI) is implemented with protocol buffers (protobufs) and defines how users can interact with network services and functions. These APIs are publicly accessible for a range of use cases and businesses.

HTS leverages existing network primitives to allow users to define and interact with tokens as a native network entity. Every entity on the network (account, file, smart contract, or topic) is represented with a unique identifier in the format of shard.realm.num (e.g. 0.0.1234). HTS adds a new entity type called a token type entity represented with the same format.

Each Hedera account will be able to hold both hbar and multiple token types. Hedera Accounts will be required to sign and associate each token type prior to being able to hold the specific token type. Use and transfer of hbar by the Hedera account functions independently of the token type. For example, users can still send hbar from their Hedera account to any other Hedera account regardless of the other token(s) held by the account.

Each function defined in the Hedera Token Service inherits Hedera's advanced key support which allows for single keys, key lists, threshold keys, and nested key structures. An overview of these key types can be reviewed here.

HTS also includes new fields in transaction receipts, records, and eventually state proofs. Each response can be independently captured, verified, and stored by any third party. This means that tokens issued using HTS will inherit the ABFT trust model enabled by the hashgraph consensus algorithm. Consensus on the validity and order of token transfers will be determined fairly without trusting a leader or small group of leaders. Transaction order is final within a matter of seconds, thus creating an immutable and verifiable log of transfers.

Mirror nodes will also be able to receive the record stream and account balance file reflecting the balance and transfers of tokens issued with HTS. At the time of publishing, files and smart contracts will not be able to interact with tokens issued using the Hedera Token Service.

Token Definition

A user's journey with HTS begins with defining the token to be created. A token issuer will need to specify the following fields in a token definition transaction. This definition will be publicly verifiable by any user which requests info about the specific token entity. In return, the network will provide the transaction response including the unique token type entity that inherits this definition.

Some of the below fields are required whereas others are optional in order to better suit specific use cases. The token definition will include:

NAME	A required string of only ASCII characters representing the name of the token. This field is used to identify a token. The name is not required to be unique compared to other tokens in the network.
SYMBOL	A required string of only uppercase letters representing the symbol of the token. This field is used to identify a token in shorthand, and can potentially support ease of integration with exchanges or wallets. The symbol is not required to be unique compared to other tokens in the network.
DECIMALS	A required specification of the number of decimal points the token can be divisible by. Once set, this field cannot be changed with an update transaction. A token issuer can specify 0 decimals.
TREASURY ACCOUNT	A required field which specifies the Treasury Account for the token. The Treasury Account will receive the initial supply specified in the definition transaction along with any tokens created from a mint transaction. The Treasury Account can be updated by the Admin key signing an update transaction.
TOKEN RENEWAL ACCOUNT	A required field if auto renew is set to true, the Token Renewal Account pays for token storage. The account will be automatically charged to renew the token's expiration, at autoRenewPeriod interval.
ADMIN KEY	An optional field which specifies the public key or keys which must sign any admin transactions. Admin rights include the ability to update the token definition. This includes updating the admin, freeze, wipe, or KYC keys, only if they already exist at the time of token creation, as well as being able to modify the Treasury Account and auto renew properties. This key or keys can also delete a token. If the admin key is left empty then token properties cannot be changed once defined.

FREEZE KEY	An optional field which specifies the public key or keys which are able to freeze or unfreeze accounts as it relates to the defined token. A frozen account cannot interact with the specific token type, unable to send or receive the token. This key can also unfreeze accounts for them to resume sending and receive tokens. The account will still be able to send and receive hbar, but will not be able to send or receive the specific token type. Potential use cases for this field include AML enforcement or other governance requirements of the token. If this field is left empty, the specified token within these accounts cannot be frozen or unfrozen. The key cannot be updated if it was left empty at definition.
DEFAULT FROZEN	An optional field which can be set to either true or false. If true, then every account associated with the token type specified is marked as frozen. If false, then every account associated with the token type specified is marked as unfrozen.
WIPE KEY	An optional field which specifies the key or keys which can be used to wipe token balances from a specified account. A wipe burns a specified amount of the token from a specified account. An account freeze followed by a wipe is commonly referred to as a clawback. The wipe transaction does not impact the hbar balance of the account. Potential use cases for this field include AML enforcement or other governance requirements of the token. If specified, any account associated with a specific token type can be wiped by this key or keys. If this field is left empty, accounts cannot have token balances wiped. The key cannot be updated if it was left empty at definition.
KYC KEY	An optional field which specifies the key or keys which can be used to flag an account as having completed 'Know Your Customer' (KYC) with respect to the specific token type. This key or keys can switch the flag on an account from un-KYC'd to KYC'd and vice versa. If KYC is required, an un-KYC'd account will behave similar to a frozen account in that it cannot send or receive transactions for the specified token. Potential use cases include KYC/AML assurance for certain token types. If this field is left empty, accounts cannot be KYC'd or un-KYC'd with respect to the specific token type. The key cannot be updated if it was left empty at definition.
SUPPLY MANAGER KEY	An optional field which specifies the public key or keys which are able to mint or burn tokens. Minted tokens are sent to the Treasury Account. Tokens can only be burned from the Treasury Account. The Initial supply is immutable if a Supply Manager key is left empty. The key cannot be updated if it was left empty at definition.
INITIAL SUPPLY	A mandatory field that specifies the initial supply of the token to mint with the token definition. The initial supply will be transferred to the Treasury Account. The initial supply can be set to 0.

Transactions

Once a token has been created, transactions moving that token between accounts can be submitted to the network.. These transactions support the defined behaviors of the token definition. A user's ability to submit a transaction against a particular token is dependent on whether or not the involved accounts have met the defined criteria, i.e. that the accounts are associated to the token, are not frozen, etc.

In each transaction and query, a Hedera account is required to pay any hbar-denominated transaction fees in order for the transaction to succeed.

Amounts in each transaction, such as the amount of tokens to be transferred in a transfer transaction, are specified as the smallest divisible unit based on the number of decimal places specified in the definition transaction.

The following section provides an overview of the transactions and queries that are supported in the Hedera Token Service.

CREATE	The creation transaction is the initial transaction which can be submitted by any user in the network to create a new token type. The transaction will specify each of the parameters outlined in the previous section (admin key, supply manager key, etc.). The transaction results in the creation of a token type entity which inherits the roles and behaviors specified in the transaction submission. The result will provide the entity ID for the token that was created (e.g. 0.0.1234).
UPDATE	An update transaction is used to change certain fields specified in the create transaction for a specific token type entity. The update transaction can update the name, symbol, keys for each role, as well as Treasury Account and auto renew account and period. It cannot change the decimals of the token type entity. An update transaction must be signed by the admin key. Updates are not possible if an admin key was not specified in the token's initial create transaction.
MINT	The mint transaction adds tokens to the circulating supply on the network. The transaction can only be submitted by the Supply Manager key and specifies the amount to be minted. The mint transaction sends the newly minted tokens to the Treasury Account.
BURN	The burn transaction removes tokens from the circulating supply on the network. The transaction can only be submitted by the Supply Manager key and specifies the amount to be burned which cannot bring the supply below 0. Tokens can only be burned from the Treasury Account.
WIPE	A wipe transaction burns a specified amount of the token from a specified account. The circulating supply is reduced because the wipe burns the tokens. A wipe transaction can only be signed by the wipe key. Any account's token balance can be wiped if the wipe key is present in the token's initial define transaction.
FREEZE	A freeze transaction can mark an account as frozen which prevents a specified Hedera Account from interacting with the token type. This means the account cannot transfer or receive the token to or from other accounts. The freeze transfer must be signed by the freeze key. The frozen account can still interact with hbar and other token types. The freeze transaction can also unfreeze accounts to resume interacting with the token type.

KYC	A KYC transaction acts similar to a freeze transaction in that it updates the flag on a specified account to mark it as KYC'd or not KYC'd. If KYC is required then a not KYC'd account cannot send or receive the token. The KYC transaction must be signed by the KYC key.
ASSOCIATE	A token associate transaction approves the provided Hedera account can accept a token type. Hedera accounts must be associated with a token prior to be able to transfer the token to that account. The Hedera account that is being associated to a token is required to sign the transaction. Accounts can also be disassociated with token types.
TRANSFER	A transfer is the primary transaction a token holder uses to transfer tokens from one account to another. A transfer transaction must be signed with the key which controls the account sending the token. The transfer transaction will send the token from the specified account(s) to a specified receiving account(s). A transfer transaction can optionally involve multiple tokens and hbar as an atomic operation.

These basic transaction types can be used to enable tokens to support a wide range of use cases from stablecoins and securitization to decentralized finance and non-fungible tokens.

Queries

The Hedera Token Service will also add new fields to receipts and records, as well as allow applications to query information about a created token on the network. Receipts and records will include information on token transfers between accounts.

Applications will also be able to get info on a token to learn which keys and inputs are associated with the token type.

Applications will also be able to get account balances of the new token types similar to how they get the balance of hbar today.

Pricing

The pricing of the Hedera Token Service is designed to be fixed in USD, like every Hedera network service, and supportive of a cost profile similar to that of hbar transactions. This means that users can expect low fees for basic transfers and higher fees for more expensive actions like creating a token.

Fees on the Hedera mainnet are determined by the Hedera Governing Council and have not yet been finalized at the time of writing this paper. Information on the latest network fees can be found at hedera.com/fees.

HTS BENEFITS

Ease of Deployment

The Hedera Token Service is available on the Hedera mainnet for public use without restriction. A developer can deploy their token natively without needing to configure additional infrastructure such as a mirror node(s) or permissioned network node(s). Token transfers and account balances would be exposed via mirror nodes in a publicly verifiable way similar to other network services.

Tokens can be easily deployed after defining the key roles and behaviors to support its operation. Accounts can be created to access the token once the definition transaction comes to consensus on the mainnet.

Low Cost

The Hedera Token Service will have a cost profile similar to hbar transactions. This means that token transfers can occur, fixed in USD, for less than $0.01. This will support high throughput use cases such as payments, financial markets, or in app purchases. Hedera Token Service fees will be visible once finalized through the pricing calculator here.

High Performance

Hedera Token Service presents thousands of transactions per second with finality in 3-5 seconds. We anticipate the throughput of which to improve over time.

Interoperability

The Hedera Token Service allows defined tokens to inherit the account, transaction, and trust model of hbar transactions on the mainnet. This provides a benefit for interoperability with both tokens on Hedera and on other distributed ledger networks.

Hedera will also work to create demo code to showcase how transfers can also be verified in order to trigger events on other networks. Those networks could be built as permissioned networks leveraging the Hedera Consensus Service, or fully separate public networks.

Decentralized Trust

Tokens issued using the Hedera Token Service inherit the decentralized trust model enabled by the hashgraph consensus algorithm and Hedera network.

Token transactions are included in receipts, records, and state proofs. This means that any third party application can independently verify transactions that come to consensus on the mainnet. Mirror nodes will expose this history as well without relying on a centralized source of proof.

These proofs are also portable to other networks or applications to enable things like atomic swaps as discussed in the prior section.

Ease of Integration

Finally, tokens issued on the Hedera mainnet can benefit from standardized integrations with third party services including wallets, custody providers, and exchanges. Third parties which adopt the Hedera account model are able to easily extend their integrations to any token type. This can provide a path to adoption and liquidity for tokens issued on Hedera.

EXAMPLE

Hedera has built a demo application which supports token creation with the Hedera Token Service. The code, and associated user interface, allows a user to create a token, define the roles and behaviors associated with the token, and transfer between user accounts which opt in to the token.

The code is open sourced here.

Tokenization on Hedera Consensus Service (HCS)

OVERVIEW

Hedera Consensus Service enables more customized and permissioned networks to be built for a wide range of business use cases. These networks allow builders to define specific network use cases, participants, deployment models, and data privacy. The networks are made up of permissioned node operators who have a local copy of a database which is updated with custom code based on synchronized messages from the Hedera Consensus Service.

Hedera Consensus Service (HCS) allows any application or network of applications to submit messages to the Hedera mainnet in order to generate a consensus timestamp, sequence number, and running hash. These outputs of the Hedera mainnet can be used to synchronize an order of messages that can be leveraged by each instance of an application independently.

In the context of tokenization, the messages sent to HCS can include instructions for transferring tokens between a ledger of accounts. The actual ledger would be maintained by a permissioned set of node operators who expose that ledger of accounts to end users. Said another way, the state is maintained by a permissioned network of nodes while ordering is provided by the Hedera mainnet with the Hedera Consensus Service.

The following section outlines how Hedera Consensus Service along with a set of formally verified specifications and audited sample implementations support tokenization for a unique set of use cases.

TOKEN MESSAGE STANDARD

The Token Message Standard is a formally verified specification and audited implementation for building a token using Hedera Consensus Service. Hedera worked with Quantstamp to create the formal verification model and audit report on the reference implementation.

The core components of a token on HCS which are used to implement the Token Message Standard include:

- Application logic that defines the token contract to codify the roles and behaviors of the token.
- Permissioned nodes which execute the logic, expose the token to users through accounts which they control on a replicated ledger, and stores that ledger of accounts, and HCS Transaction Ordering which ensures the nodes stay in sync in a fast and secure manner.

An overview of the architecture of the Token Message Standard for tokenization on HCS is shown in Figure 1.

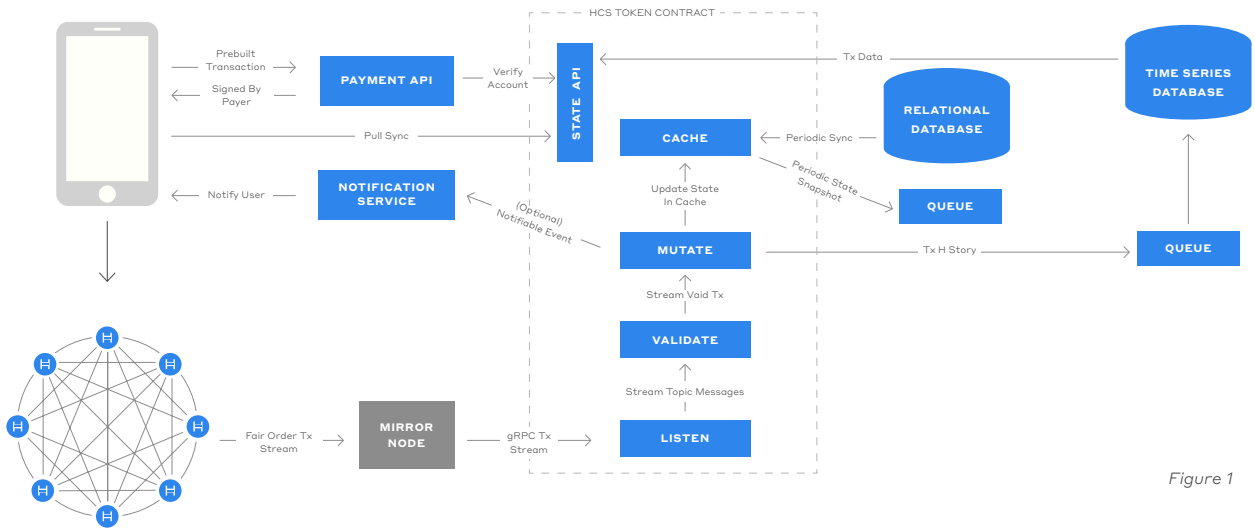


Figure 1

Token Contract

The application logic, referred to as the token contract in the middle of Figure 1, codifies the roles and behaviors of the token such that each permissioned network node updates their database in the same manner and exposes the token to the user through a common taxonomy.

The token contract receives an ordered message from the Hedera Consensus Service and is streamed through a mirror node for other parties to access. The token contract enables the node to "listen" to a specific TopicID against which all messages relevant for the token are submitted. The message submission to that topic can either be public and submitted directly from the user's device, or can be permissioned to a specific set of parties such as the node operators.

The token contract then "validates" the message received from the Hedera Consensus Service to ensure that it complies with the roles and behaviors specified in the token definition. For instance it can validate the keys signing a required transaction type to ensure that the proper user signed the transaction. It can also validate the details of the transaction to ensure that an account does not transfer more than it holds, or that the transaction is blocked if the account is frozen. This validation could accept the transaction or reject it based on the rules outlined in the token contract.

Post-validation the token contract will then change the state of the permissioned network node based on the details of the transaction. This could be as simple as updating the new balance of accounts based on a token transfer or an update of the key which controls minting and burning of the token. Token contracts can also introduce more complex logic based on the needs of the use case which could include atomic swaps, automated event triggering, or reference to external oracles.

The state mutation then can trigger a notification to the user of a completed transaction as well as caching of the transaction details and storage in relevant databases on the permissioned network

Finally, it is worth noting that the most significant advantage of tokenization on the Hedera Consensus Service is the ability to fully customize and control the token contract.

SPECIFICATION

Hedera partnered with Quantstamp to create a specification that could be formally verified to use as a secure and stable base for tokenization on HCS. Each permissioned network, or token network is able to create their own implementation of the token contract while still leveraging this formally verified specification.

The Specification started with a set of roles and behaviors for creating a stablecoin from the InterWork Alliance's Token Taxonomy Framework. Hedera is a founding member of the InterWork Alliance, a technology-agnostic standards group focused on standards development driven by business requirements for tokenization. We choose a stablecoin as the token type given its demand in the market and its modularity to support additional use cases including payments and securitization.

Third parties can review the specification and formal verification created by Quantstamp here.

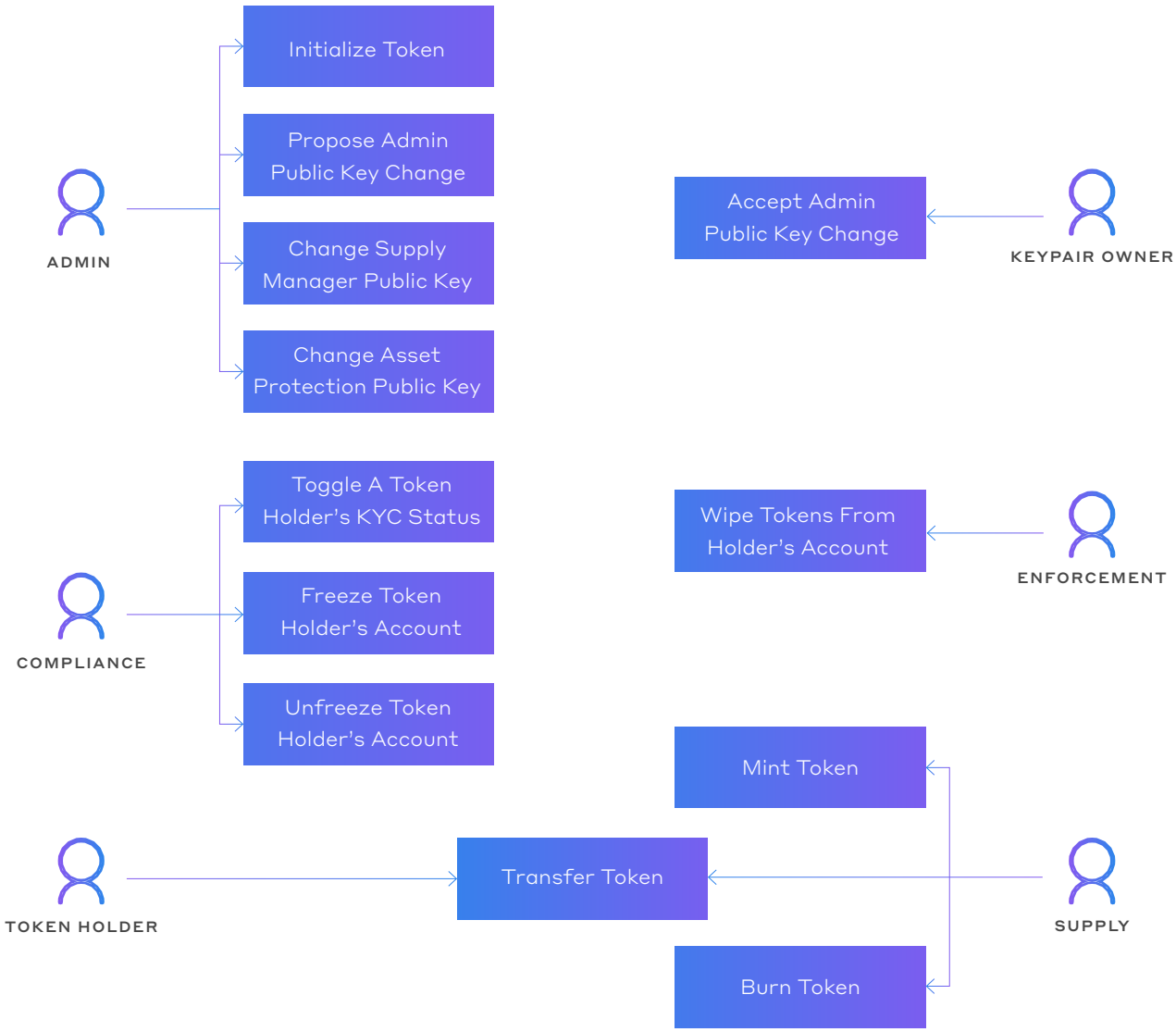
ROLES AND BEHAVIORS

In practice the roles and behaviors outlined in the Token Message Standard are very similar to those in the Hedera Token Service. They are designed to be optional mechanisms for enforcing things like administrative rights, supply management, compliance, and token transfer. Figure 1 outlines the optional roles and their associated behaviors.

The roles and their associated behaviors are outlined as follows:

ADMINISTRATOR	The administrator provides functions related to the creation and updating of the Token Contract roles and behaviors. The administrator is responsible for initializing the token contract including defining the characteristics of the token including its name, symbol, and decimals as well as which roles are included. This same role would be required to sign any update transactions to change any information about the token contract. The Administrator can also propose changes to the admin key required to sign the above transaction types. The same holds true for the other roles. An administrator can change the key for the supply manager, compliance, or enforcement roles. The new keys would then be required for carrying out any of the associated behaviors once the message had been updated to the Token Node.
KEYPAIR OWNER	The new key owner of an administrator key must also sign the transaction updating the admin key for the token contract. The Keypair Owner refers to the role which holds the private key corresponding to the public key now referenced as the admin key.
SUPPLY MANAGER	The Supply Manager role has the primary responsibility for minting and burning tokens in the token network. This is done to increase or decrease the circulating supply. This role can be used for stablecoins, securities, or other tokens which may have their supply fluctuate. Use of the Supply Manager role can be connected to the logic included in the token contract. This can connect minting of a token to some other business event. A Supply Manager can also transfer tokens between accounts.
COMPLIANCE	The Compliance role is responsible for enforcing any necessary compliance controls for the token network. The specification allows compliance managers with the ability to KYC and un-KYC as well as freeze and unfreeze accounts in the network. These capabilities can be used to enforce compliance requirements in the token network to prevent illicit activity. Typically this role could be fulfilled by a third party providing the KYC and AML monitoring services.

ENFORCEMENT	The Enforcement manager role enables the clawback of tokens from user accounts. This role is carried out through a wipe transaction which burns a specified amount of tokens from a user's account. The burned amount is removed from the circulating supply of the token network.
TOKEN HOLDER	The Token Holder role is likely the most common role in the network. The Token Holder role can hold a balance of the token in an account and transfer up to the amount held to another account. Each Token Holder controls their account with a unique key pair, enabling decentralized account management.



AUDITED REFERENCE IMPLEMENTATION

Hedera programmed a reference implementation for a Hedera Consensus Service based stablecoin using the Token Message Standard. The implementation was based on the formally verified specification and was audited by Quantstamp. The open source implementation along with audit report can be found here.

Token Node

Thus far, the token contract defined in the previous section could be deployed by a single application with an associated database to expose tokenization over HCS to an end user. However, it is only once this token contract is deployed for the same token across a series of Nodes that the true value of decentralization in its fault resistance and high availability can be delivered to the end user. We refer to the permissioned nodes which deploy the token contract logic for a common token type as a token node, with the totality of nodes making up the token network.

This section will provide an overview of the components required to be deployed to each Token Node in order to support a common token defined in the token contract.

TOKEN CONTRACT

The token contract, defined in the previous section, includes the code which defines the role and behaviors of the token for a specific use case. The token contract automates the execution of logic consistently on each node such that each ordered message from the Hedera Consensus Service results in a consistent update to each Node's state.

Token Nodes may be required to update their token contracts at some point during operation to support new functionality or desired roles and behaviors. Updates can also be synchronized with the Hedera Consensus Service providing an ordered message with the same timestamp to each Token Node instructing them to update their token contract at that time.

API

Each Token Node can independently expose the token to a user. And each user can access their token account through each Token Node.

The Token Nodes can deploy an API server which provides a standard interface for getting information about an account in the token network. The API server can get the balance of an account, confirm a transaction record, or even get information on the current keys associated with each role in the token contract.

Front end user interfaces providing user access or administrator dashboards can be customized by d ifferent providers using these APIs. Token networks should seek to provide a diverse number of these applications to end users for greater competition and diversity.

DATABASE

Tokenization on the Hedera Consensus Service allows Token Nodes to store data from operating the token network in their local databases. They can use the logic in the token contract to update their local state to ensure it is consistent based on the latest message ordered by the Hedera Consensus Service.

ON-BOARDING

Token Node on-boarding enables decentralized token network operation as well as limited downtime for individual Token Nodes should issues occur.

A Token Node on-boarding will likely depend on the requirements of the token network. These requirements could include KYC of the Token Node Operator, specification of where the Token Node is located, and potentially key sharing for purposes of data encryption.

At a minimum, the Token Node would need to deploy a copy of the latest token contract. The Token Node can either get a copy of the latest state from a trusted third party or can reconstruct the state by streaming all of the Hedera Consensus Service messages from any third party Mirror Nodes. This would update the state to be consistent with that of the token network.

The Token Node would also need to subscribe to the Topic ID of the token network created on the Hedera Consensus Service.

Transaction Ordering

Consistency of the Token Nodes and the ability to support decentralized exchange of token relies on the process of ordering transactions on the network. Transaction ordering ensures that instructions to transfer tokens or carry out any other behavior on the network is updated to each Token Node's state in the same way.

The following section will discuss how the Hedera Consensus Service serves as the foundation of decentralized ordering for distinct token networks.

HEDERA CONSENSUS SERVICE

The Hedera Consensus Service provides fair, fast, and decentralized ordering of messages for any permissioned network.

Messages received from any participant in the token network are submitted to the Hedera mainnet with a common TopicID. The Topic ID can either be public so anyone can submit a message to that topic, or permissioned such that only specific parties can submit a message to that topic.

Once a message is received by a mainnet consensus node it is gossiped between the mainnet Nodes until it comes to consensus. Hedera uses the hashgraph consensus algorithm to determine the fair timestamp of each individual message, thus establishing a consensus order of messages for the Hedera Consensus Service. More detail on the hashgraph can be found here.

A unique feature of transaction ordering on the Hedera mainnet is the ability to deliver the level of performance needed for high throughput payment use cases. The Hedera Consensus Service can enable permissioned networks to support thousands of transactions per second with latency end to end in a number of seconds.

Finally, the Hedera Consensus Service enables each permissioned network to take advantage of the Asynchronous Byzantine Fault tolerance enabled by the hashgraph consensus algorithm. Every consensus node on the mainnet participates in every round of consensus meaning that no single party or small group of parties is trusted as leader. This makes the network resilient to fault or outage, either maliciously or accidentally.

MESSAGE

Each transaction within the permissioned network (i.e. transferring 5 tokens from user A to User B) is packaged into a message that includes the signature of the party transferring the tokens. The same is true if the transaction is meant to mint tokens, burn tokens, freeze an account, update the keys on account, etc.

In general, each message requires the signing from a specific key depending on the action that is being carried out.

The message is then sent to any consensus node in the Hedera mainnet. Each permissioned network node can independently receive the ordered message post-consensus in order to update their local state. The Token Message Standard defines a common structure for these messages such that each permissioned network node can interpret them in the same way.

Messages can be customized based on the output of the token contract to provide additional instructions relating to any given use case. These instructions could reference transactions on other networks to support the relay of a token transfer from one network to another in order to support a token bridge.

By default, each message sent to the Hedera Consensus Service is publicly visible as any application or mirror node can view and save that information. This is acceptable for many token use cases which seek to provide the maximum amount of decentralization to the end users.

More privacy may be required in some cases. A use case (such as Central Bank Digital Currencies) could require that certain data be only visible by the permissioned network nodes. Numerous models for encryption can be used in these instances to encrypt and decrypt the messages sent to the Hedera Consensus Service. This model of privacy is not currently implemented in the reference implementation cited above.

MIRROR NODE

Client applications can get records of HCS messages directly from consensus nodes if they so choose. In practice, the majority of applications, and the reference implementation of tokenization on HCS, leverage mirror nodes to automatically receive messages as they come to consensus.

A Token Node can subscribe to the Topic ID configured for their token network in order to receive the HCS messages from a Mirror Node. A Token Node could ask any mirror node operator for the same information and verify that they are providing the same result as each other and the mainnet.

A Token Node could also choose to operate its own mirror node in order to get records directly from the Hedera mainnet and store them as long as is needed for their use case.

FEES

Use of the Hedera Consensus Service requires a Hedera Account which pays the transaction fees associated with using the service. Like every Hedera network service fees are fixed in US dollars, but charged in hbar.

Token issuers using the Hedera Consensus Service can expect to pay fees associated with creating and maintaining their Hedera Account, creating a Topic ID for their token network, and submitting a message for consensus. An overview of the fees associated with all actions on the network can be found here.

A token network has flexibility when it comes to who pays the relevant Hedera fees. A single account can be configured to pay for all of the transaction fees. Token Nodes can also each have their own Hedera Account so each pays fees necessary to support their end user. Both of these models enable the underlying cryptocurrency to be obscured from the end users.

Token networks can also allow end users to each have their own Hedera Account which they use to pay for relevant Hedera fees. This is best enabled by applications which enable a token user to interact directly with the Hedera mainnet to submit their own messages to update the token network.

HCS BENEFITS

Tokenization on the Hedera Consensus Service is best suited for token use cases which require a greater degree of privacy and control over the token contract logic and Token Node operation. This will come at the expense of a higher operating and governance burden which may be acceptable for some use cases.

Token Contract Customization

Token contracts can be programmed custom to the needs of the use case. Unlike the token definitions using Hedera Token Service, they are not restricted in their complexity because there is no need to worry about consumption of the shared resources on the Hedera mainnet.

This means that rules such as lock up periods, automated payment on delivery, or locking of funds autonomously could be programmed. We expect that examples of this type of logic will be created and published in the community above and beyond the Token Message Standard.

It is important to recognize that adding this complexity to a token contract can add performance overhead to the token network as whole. As mentioned previously, the basic Token Message Standard cna support thousands of transactions per second with low latency. These metrics may increase as more complexity is added to the token contract.

Node Deployment

Tokenization on HCS may also be preferred when privacy or control over who has access to the token network state is required. For instance, certain use cases like Central Bank Digital Currencies may require that the actual token nodes only be operated in a specific geography. A token network could thus require token nodes reside in that jurisdiction while still leveraging a fairly ordered stream of messages from the Hedera Consensus Service.

Tokenization on HCS may also be beneficial when more stringent privacy concerns are required as well. Token nodes could be required to comply with all privacy requirements necessary for the use case including the encryption of messages submitted to HCS.

In practice, Tokenization on HCS can also enable the Token issuer to govern the token network. Governance can include rules for who can access the token, who can operate a token node, and other requirements suited for the use case.

These privacy and governance features are not included in the basic Token Message Standard. Frameworks like Hyperledger Fabric, R3's Corda, and more provide tools for this extended functionality for a token network. More information on HCS integrations with these frameworks can be found here.

EXAMPLE

Hedera created a stablecoin demo application that has been open sourced to show how the Hedera Consensus Service could be used for tokenization. The demo application leverages the above framework for tokenization on the Hedera Consensus Service.

CONCLUSION

Hedera Hashgraph provides multiple models to support a variety of tokenization use cases. For some, Hedera Token Service presents a more trusted and transparent solution with tokens living directly on the Hedera public ledger, and thus being able to take advantage of a developing ecosystem. To support use cases that require higher degrees of customization or control, Hedera Consensus Service offers a means to establish a permissioned tokenized asset network with public trust. Through the use of the hashgraph consensus algorithm both of these models offer a high-throughput, low-latency, solution with predictable fees all atop the decentralized Hedera network.

As of Q1 2021, Hedera has released the Hedera Token Service to the Hedera mainnet. Hedera has also identified the next set of features which will be built into the Hedera Token Service to build additional transfer types that can occur automatically. We have also identified a number of tools and frameworks which will support the Hedera Token Service. Updates to the Hedera Roadmap can be found here.