Kyber: An On-Chain Liquidity Protocol

22 April 2019
v0.1

Abstract

This paper presents Kyber, a fully on-chain liquidity protocol for implementing instant cryptocurrency token swaps in a decentralized manner on any smart contract enabled blockchain.

Kyber design allows for any party to contribute to an aggregated pool of liquidity within each blockchain while providing a single endpoint for takers to execute trades using the best rates available. We envision a connected liquidity network that facilitates seamless, decentralized cross-chain token swaps across Kyber based networks on different chains.

This paper provides the technical specifications and key design principles of the protocol to be followed when being implemented on smart contract enabled blockchains. It also includes the specifications for liquidity makers or takers to participate in any given implementation.

I. Introduction

Cryptocurrency trading on centralized exchanges has been shown to be vulnerable to cybersecurity hacking and internal frauds over the years, with the most infamous hacks being Mt. Gox and Coincheck. In addition, trading on centralized exchanges is not compatible with DeFi applications since it is technically infeasible to bridge between decentralized applications and centralized servers without compromising the trust model.

Regardless, DeFi applications all need access to good liquidity sources which is a critical component to provide good services. Currently, decentralized liquidity is comprised of various sources including DEXes (Uniswap, OasisDEX, Bancor), decentralized funds and other financial apps. The more scattered the sources, the harder it becomes for anyone to either find the best rate for their trade or to even find enough liquidity for their need.

Kyber is a fully on-chain liquidity protocol that can be implemented on any smart contract enabled blockchain. Kyber’s solution allows liquidity to be aggregated from diverse sources into a single network, which in turn provides a single endpoint for takers to seamlessly perform multiple token trades in a single blockchain transaction. End users, DApps or any other party only need to query this single endpoint to get the best available rate for their trade.

The protocol allows for a wide range of implementation possibilities for liquidity providers, allowing a wide range of entities to contribute liquidity, including end users, decentralized exchanges and other decentralized protocols. On the taker side, end users, cryptocurrency wallets, and smart contracts are able to perform instant and trustless token trades at the best rates available amongst the sources.

The on-chain instant exchange property is critical for enabling a wide range of decentralized use cases, including financial protocols and cryptocurrency payments. One would expect different implementations of Kyber protocol on other public blockchains to make on-chain instant exchanges to be available for various use cases and applications on these blockchains.

There are a number of other blockchains with distinctively different feature sets, solving different use cases, resulting in a plethora of diverse emerging digital ecosystems. In order to support these emerging ecosystems, the Kyber protocol will play a key role in these blockchains by facilitating liquidity between the different stakeholders in the respective ecosystems. In addition, by standardizing the design of the reserve based liquidity model and actively working on cross-chain solutions such as PeaceRelay and the Waterloo Relay Bridge, we envision a connected liquidity network across the implementations on different chains.

Over time, we envision the protocol contributors, network maintainers and community of KNC token holders working together to govern Kyber. As Kyber scales, an appropriately decentralized governance framework will be critical in order to decide on important upgrades, treasury allocation, and additional features of the protocol, and to standardize the key aspects of the implementations. In particular, Kyber will support one main implementation per blockchain that follows the exact protocol specification, KNC governance, and utility.

II. Protocol Architecture

Design Properties

At the heart of the design, the protocol smart contracts offer a single interface for the best available token exchange rates to be taken from an aggregated liquidity pool across diverse sources.

- **Aggregated liquidity pool.** The protocol aggregates various liquidity sources into one liquidity pool, making it easy for takers to find the best rates offered with one function call.
- **Diverse sources of liquidity**. The protocol allows different types of liquidity sources to be plugged into. Liquidity providers may employ different strategies and different implementations to contribute liquidity to the protocol.
- **Permissionless.** The protocol is designed to be permissionless where any developer can set up various types of reserves, and any end user can contribute liquidity. Implementations need to take into consideration various security vectors, such as reserve spamming, but can be mitigated through a staking mechanism. We can expect implementations to be permissioned initially until the maintainers are confident about these considerations.

The core feature that the Kyber protocol facilitates is the token swap between taker and liquidity sources. The protocol aims to provide the following properties for token trades:

- **Instant Settlement.** Takers do not have to wait for their orders to be fulfilled, since trade matching and settlement occurs in a single blockchain transaction. This enables trades to be part of a series of actions happening in a single smart contract function.
- **Atomicity.** When takers make a trade request, their trade either gets fully executed, or is reverted. This “all or nothing” aspect means that takers are not exposed to the risk of partial trade execution.
- **Public rate verification.** Anyone can verify the rates that are being offered by reserves and have their trades instantly settled just by querying from the smart contracts.
- **Ease of integration.** Trustless and atomic token trades can be directly and easily integrated into other smart contracts, thereby enabling multiple trades to be performed in a smart contract function.

Overview

The protocol is implemented as a set of smart contracts on any blockchain that fits the requirements. The following diagram shows an overview of the various actors involved in an implementation of the protocol.

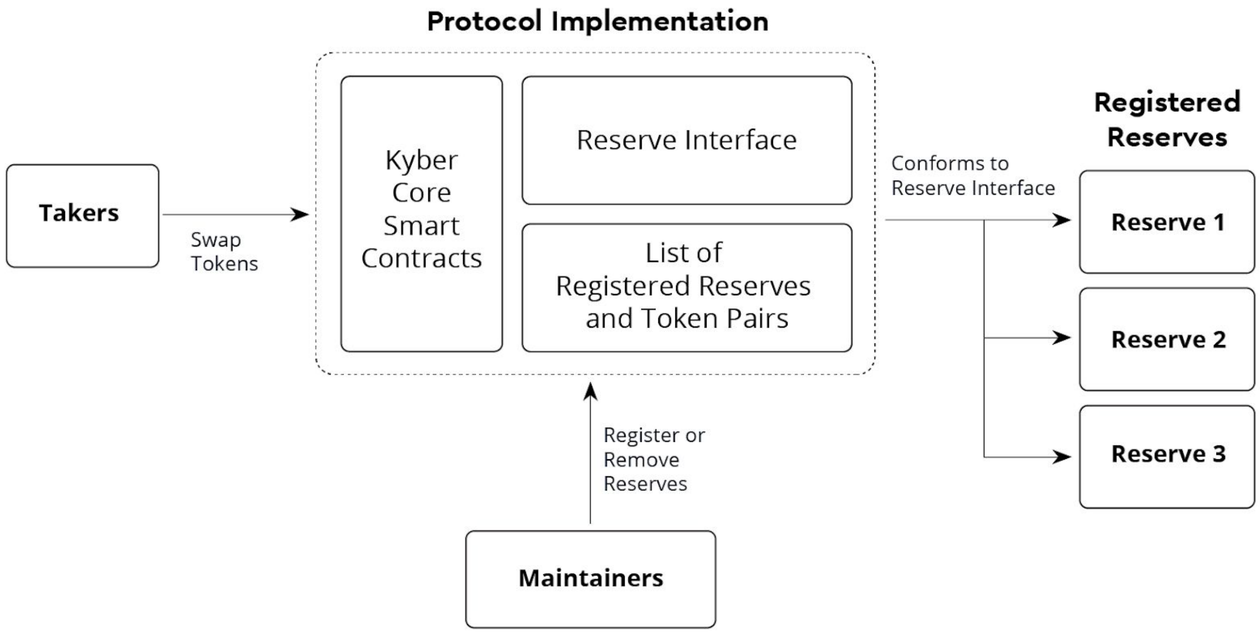

Protocol Implementation
Takers
Swap Tokens
Kyber Core Smart Contracts
Reserve Interface
List of Registered Reserves and Token Pairs
Conforms to Reserve Interface
Registered Reserves
Reserve 1
Reserve 2
Reserve 3
Register or Remove Reserves
Maintainers

Figure 1: Overview Of Actors In A Protocol Implementation

We briefly define the actors as below. The details of how each actor works is specified in Section Network Actors.

1. Takers refer to anyone who can directly call the smart contract functions to trade tokens, such as end-users, DApps, and wallets.
2. Reserves refer to anyone who wishes to provide liquidity. They have to implement the smart contract functions defined in the reserve interface in order to be registered and have their token pairs listed.
3. Registered reserves refer to those that will be cycled through for matching taker requests.
4. Maintainers refer to anyone who has permission to access the functions for the adding/removing of reserves and token pairs, such as a DAO or the team behind the protocol implementation.

5. In all, they comprise of the *network*, which refers to all the actors involved in any given implementation of the protocol.

The protocol implementation needs to have the following:

1. Functions for takers to check rates and execute the trades
2. Functions for the maintainers to register/remove reserves and token pairs
3. Reserve interface that defines the functions reserves needs to implement

Network Actors

Kyber Core Smart Contracts

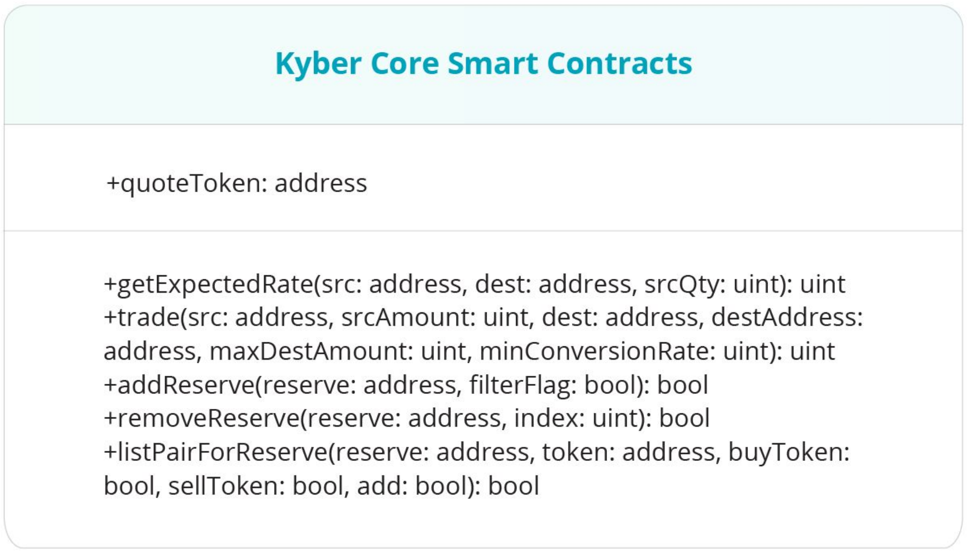
Kyber Core smart contracts is an implementation of the protocol that has major protocol functions to allow actors to join and interact with the network.

For example, the Kyber Core smart contracts provide functions for the listing and delisting of reserves and trading pairs by having clear interfaces for the reserves to comply to be able to register to the network and adding support for new trading pairs. In addition, the Kyber Core smart contracts also provide a function for takers to query the best rate among all the registered reserves, and perform the trades with the corresponding rate and reserve.

A trading pair consists of a quote token and any other token that the reserve wishes to support. The quote token is the token that is either traded from or to for all trades. For example, the Ethereum implementation of the Kyber protocol uses Ether as the quote token.

In order to search for the best rate, all reserves supporting the requested token pair will be iterated through. Hence, the Kyber Core smart contracts need to have this search algorithm implemented.

The key functions implemented in the Kyber Core Smart Contracts are listed in Figure 2 below. We will visit and explain the implementation details and security considerations of each function in the Specification Section.



Kyber Core Smart Contracts

```
+quoteToken: address
```

```
+getExpectedRate(src: address, dest: address, srcQty: uint): uint
+trade(src: address, srcAmount: uint, dest: address, destAddress:
address, maxDestAmount: uint, minConversionRate: uint): uint
+addReserve(reserve: address, filterFlag: bool): bool
+removeReserve(reserve: address, index: uint): bool
+listPairForReserve(reserve: address, token: address, buyToken:
bool, sellToken: bool, add: bool): bool
```

Figure 2: Kyber Core Smart Contract OOP Diagram

Takers

A taker is an entity that takes the liquidity provided by the registered reserves by calling the trade() function in the Protocol Smart Contracts to trade from one token to another token. A taker can be any blockchain entity including end user address, decentralized exchanges or any smart contracts.

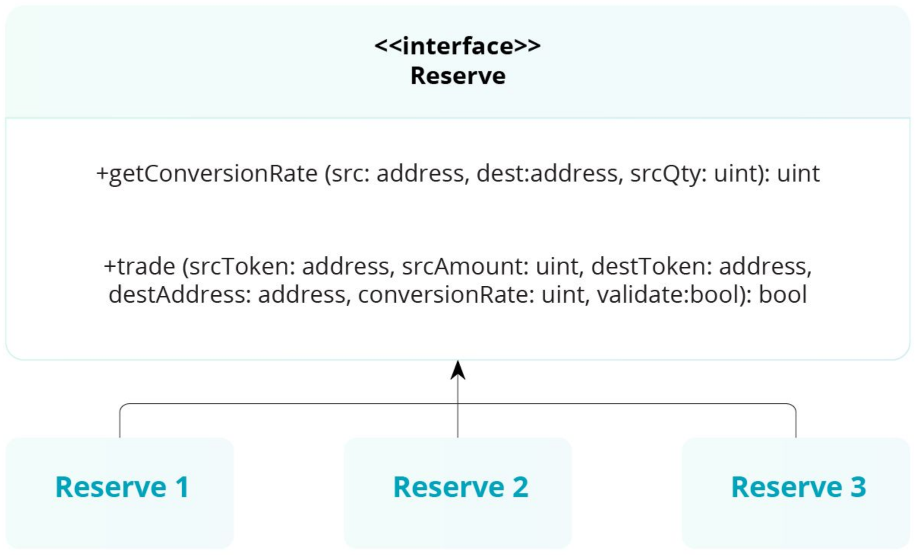
When calling the trade() function, the taker needs to specify the source and destination tokens, the minimum acceptable rate for the taker, and send the corresponding amount of source token along. The taker will receive the destination token immediately if the transaction is successfully executed, otherwise the source token will be refunded entirely.

Reserves

Reserves are liquidity sources in the network that provide liquidity in terms of tokens inventory and prices on their smart contracts. By implementing the Reserve interface, as depicted in Figure 3 below, the reserve can register to the Protocol Smart Contracts and offer its liquidity for takers to take.

The details for how reserves determine the prices and manage their inventory is not dictated in the protocol and is entirely up to the reserves, as long as the implementation complies with the Reserve Interface. That opens up a wide range of reserve types that can be part of the network including manually managed reserves (i.e. having market makers to actively manage the prices and inventory), automated market making reserves as well as many other on-chain liquidity sources.

Once the taker sent the request, the Protocol Smart Contracts will query the corresponding function in the reserve smart contract (e.g. getConversionRate() to query the price, and trade() to execute the trade). It is worth noting that in the permissionless implementation of the protocol, one may have to consider how to prevent potential attacks that makes it costly to execute trades, or even block trades from being successfully executed, such as a reserve spamming attack (i.e. attacker sets up and register a lot of reserves) or malicious implementation of reserves (e.g. reserves do not honor the quoted rates).



<<interface>>
Reserve
+getConversionRate (src: address, dest:address, srcQty: uint): uint
+trade (srcToken: address, srcAmount: uint, destToken: address, destAddress: address, conversionRate: uint, validate:bool): bool
Reserve 1
Reserve 2
Reserve 3

Figure 3: Reserve Interface OOP Diagram

III. Execution Sequences

When a trade is executed, Kyber's main contract invokes functions to iterate through all listed reserves to find and select the one offering the best exchange rate. The source tokens are transferred from the takers to that reserve, and the reserve's destination tokens are transferred to the specified destination address.

Basic Token Trade

A basic token trade is one that has the quote token as either the source or destination token of the trade request. The execution flow of a basic token trade is depicted in the diagram below, where a taker would like to exchange BAT tokens for ETH as an example. The trade happens in a single blockchain transaction.

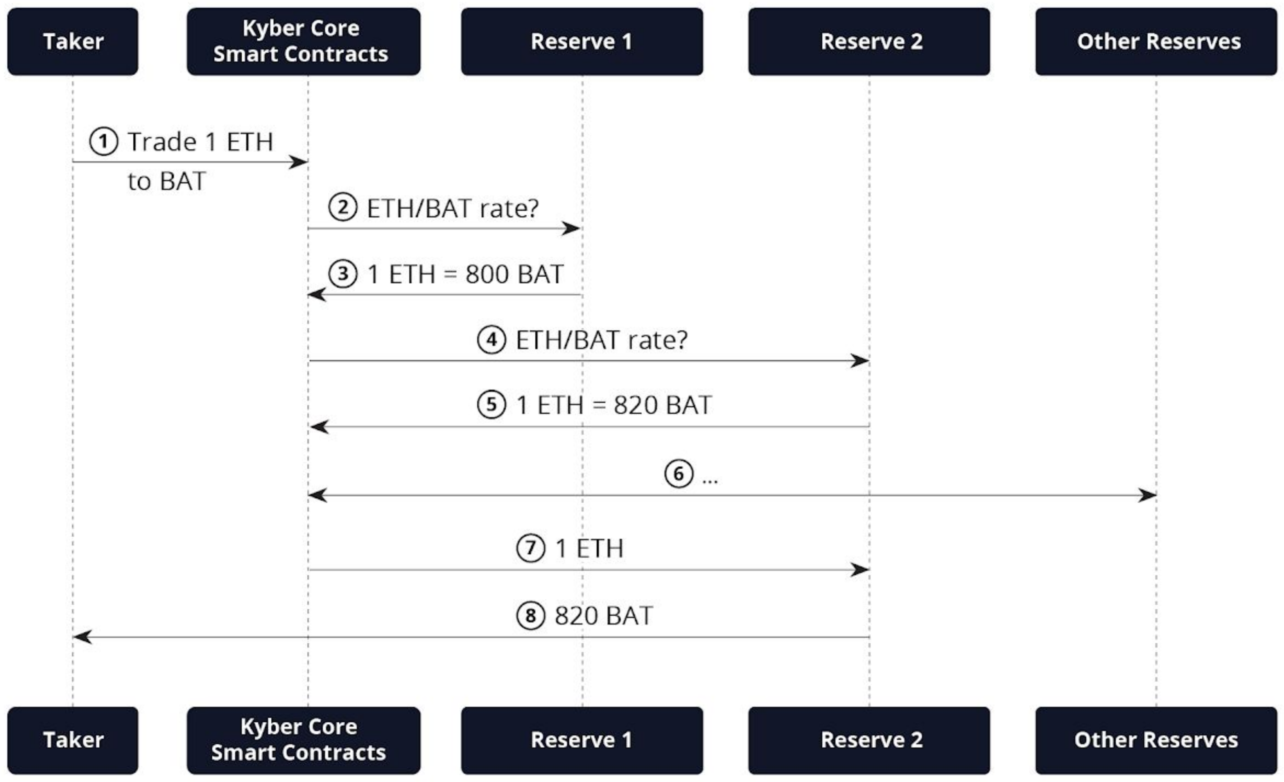


Figure 4: Basic Token Trade Execution Flow

1. Taker sends 1 ETH to the protocol contract, and would like to receive BAT in return.
2. Protocol contract queries the first reserve for its ETH to BAT exchange rate.
3. Reserve 1 offers an exchange rate of 1 ETH for 800 BAT.
4. Protocol contract queries the second reserve for its ETH to BAT exchange rate.
5. Reserve 2 offers an exchange rate of 1 ETH for 820 BAT.
6. This process goes on for the other reserves. After the iteration, reserve 2 is discovered to have offered the best ETH to BAT exchange rate.
7. Protocol contract sends 1 ETH to reserve 2.
8. The reserve sends 820 BAT to the taker.

Token to Token Trade

A token to token trade is one where the quote token is neither the source nor the destination token of the trade request. The exchange flow of a token to token trade is depicted in the diagram below, where a taker would like to exchange BAT tokens for DAI as an example. The trade happens in a single blockchain transaction.

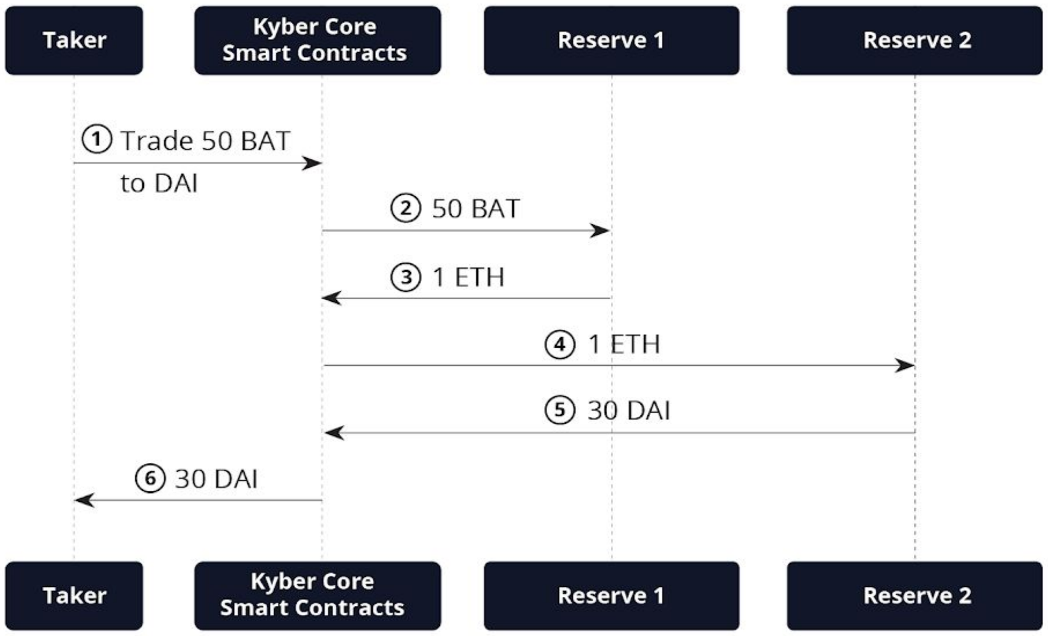


Figure 5: Token to Token Trade Execution Flow

1. Taker sends 50 BAT to the protocol contract, and would like to receive DAI in return.
2. Protocol contract sends 50 BAT to the reserve offering the best BAT to ETH rate.
3. Protocol contract receives 1 ETH in return.
4. Protocol contract sends 1 ETH to the reserve offering the best ETH to DAI rate.
5. Protocol contract receives 30 DAI in return.
6. Protocol contract sends 30 DAI to the user.

While the diagram depicts 2 reserves being selected for the token trades, it is possible that the same reserve is selected.

Reserves Registration

The diagram below shows how the protocol contract interacts with the reserve smart contract for its addition to or removal from the network of reserves, listing its token(s) for trading, and reading its token exchange rates. There are many possible implementations of reserves.

Anyone can create a new reserve, as long as the reserve smart contract contains the methods specified in the “Feeding Rates and Providing Liquidity” section. The reserve needs to be added to the network of reserves in order to be selected for taker trade requests.

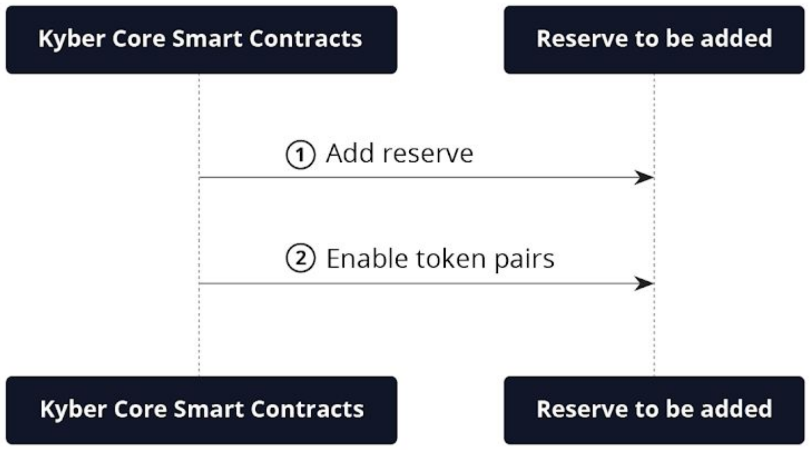
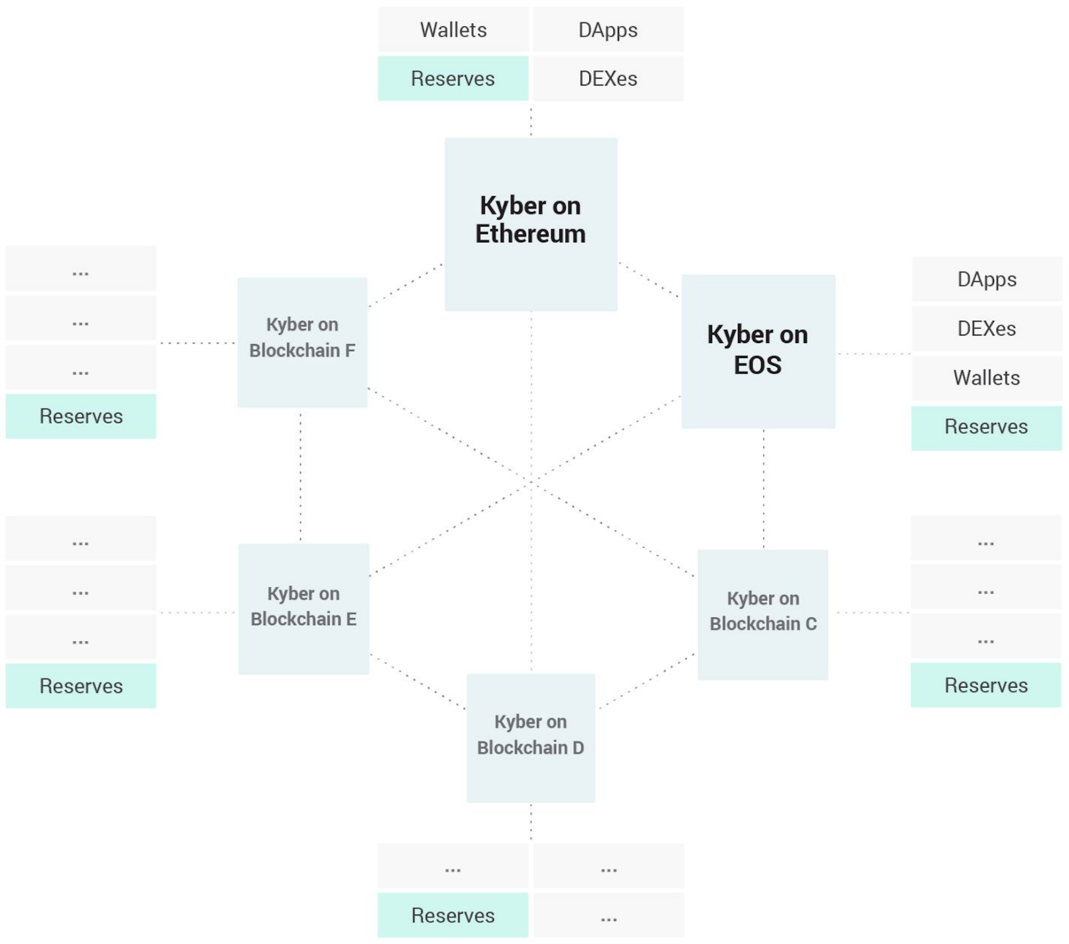


Figure 6: Reserve Interaction With Protocol

1. Protocol contract adds the reserve into the network.
2. Protocol contract enables the token pairs of the reserve for trading.

IV. Connecting Liquidity between Kyber Based Networks

Currently, Kyber Network is fully developed with a robust user base and DApp ecosystem on a single blockchain — Ethereum. In this paper, we outline how liquidity can be connected between various smart contract enabled blockchains by several practical approaches that Kyber has explored thus far. By allowing liquidity to be moved freely and (somewhat) transparently between different chains, Kyber enables a much more connected liquidity network between various blockchain ecosystems.

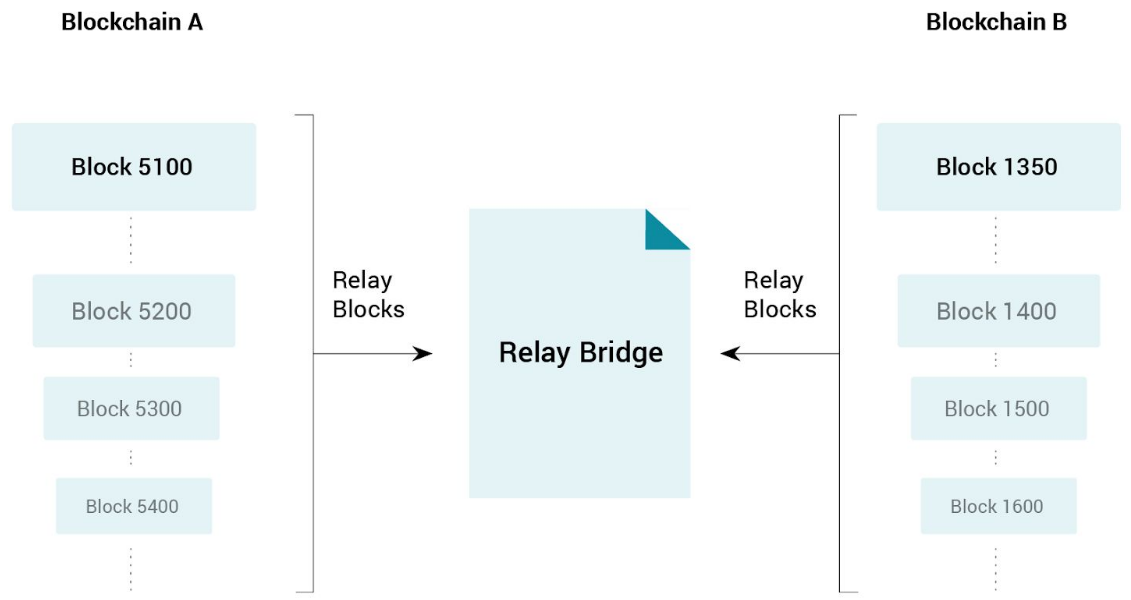


A connected liquidity network for decentralized cross-chain swaps across Kyber implementations

Practical Relay Between Chains

For smart contract-enabled chains, the most practical interoperability implementation is to have a light client as a smart contract which can be implemented on both blockchains and an efficient algorithm to verify the hash functions from both blockchains with minimal computation costs can be used. In our Waterloo project, we presented a proof of concept to prove the feasibility of a practical relay approach between Ethereum and EOS. We also demonstrated a similar approach in the PeaceRelay project to connect between Ethereum and Ethereum Classic.

We expect a generalized approach of the above to be applicable on any smart contract enabled blockchain where a light client can be implemented as a smart contract on that chain.



A relay implements a bi-directional relay of block headers between two blockchains. In such a bridge, block headers of blockchain A are constantly being submitted to a smart contract in blockchain B, which implements a light client logic to verify the validity of the headers. Analogously, headers from blockchain B are submitted to a smart contract in blockchain A. In all the blockchain protocols we are aware of, the block header contains a mathematically verifiable witness for every transaction in the block. Typically, this is in the form of a merkle root of all the included transactions. Hence, a smart contract that validates headers can be used also to validate the existence of a transaction in that header.

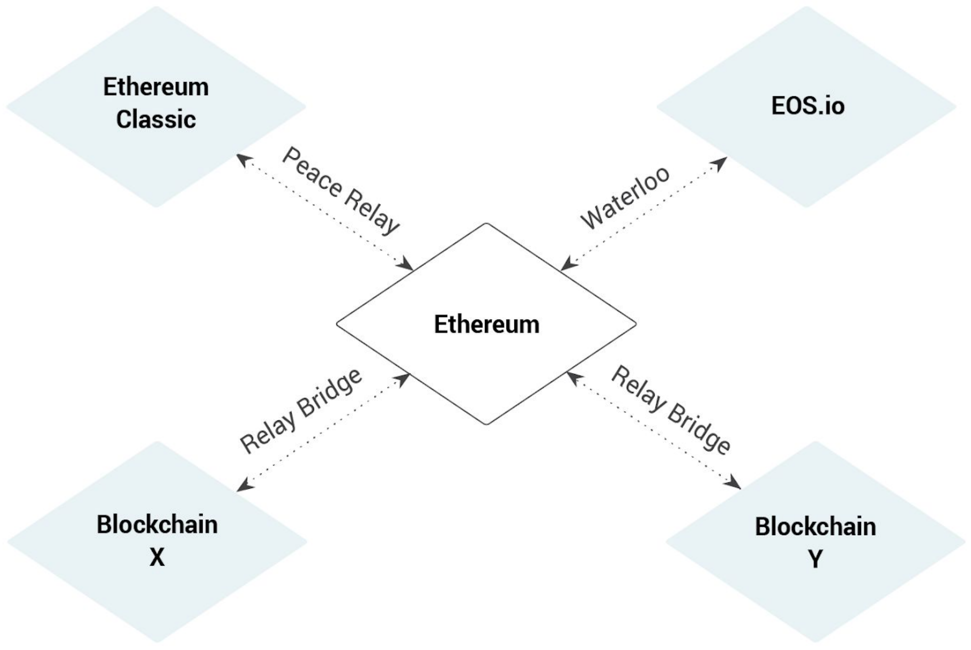
Once a smart contract can verify transactions, it can also verify an operation, e.g., token transfer, that was done by this action. The smart contracts on each blockchain serves as a light client, and as such does not verify the validity of the block content, but rather only the block header.

The relays will be used to transfer assets between the different blockchains to enable trading activity across blockchains while using each one's distinct advantages. For example, the EOS chain has an advantage of enabling higher frequency trading due to its higher transaction-per-second rate, while the Ethereum chain is considered more secure by the crypto community at large and currently offers more popular assets with advanced financial primitives, such as with the DeFi projects.

As cross-chain trades occur, anyone can view the corresponding transaction hashes of a cross-chain conversion. Every token that moves through the protocol is verifiable on all chains where the token trades have taken place.

Relay Network

The first key in implementing decentralized token swaps across two networks in different blockchains is creating a relay for every unidirectional cross-chain link, and having entities to relay and validate cross-chain communications. Each individual Kyber Network deployment is linked to a relayer and validator network on a smart contract enabled blockchain. The relays can be designed to be open and trustless such that anyone can join and relay transactions, assets between chains, removing potential central points of failure.



The above diagram shows one potential chain topology for the relay network, where Ethereum is in the center, connecting the other chains. This relay network will be an active area of research, of which we expect to share results in future updates.

Assets on Non Smart Contract Enabled Blockchains

For cryptocurrencies with blockchains that do not natively support smart contracts, an approach similar to Wrapped BTC (WBTC) can be adopted. Tokenized cryptocurrencies or wrapped coins utilize a custodian model to implement the tokenization of a cryptocurrency such as Bitcoin or Monero on a smart contract enabled blockchain. For each wrapped coin minted into existence on the smart contract enabled blockchain, there is an equivalent amount of the same cryptocurrency held in custody in a 1:1 ratio. This method has the benefit of the ease of transparency with all custodian funds being fully verifiable on-chain. The wrapped coin approach is a pragmatic one and represents an immediate benefit to DEXes, DApps, and other smart contracts without any restriction.

V. Kyber Network Crystal (KNC) Usage

Kyber Network Crystal (KNC) is an integral part of the Kyber Protocol, and will serve crucial economic, governance and treasury functionalities across all implementations of the protocol. Networks that do not have KNC as a key part of the usage in utility and governance will not be considered and will not be supported by either the project or the community.

The exact usage of KNC in different chains is dependent on the specific implementation. Key details include specific features of the different chains, and the maturity of both the ecosystem and the protocol implementation. The network maintainers are expected to make these decisions initially, and after the implementation reaches a certain stage of maturity, we expect the decisions to be made by a community of KNC holders.

As such, we can expect to see a distinction between protocol maintainers, who focus on protocol level governance like unified KNC supply, and network maintainers, who focus on the economics and technical features of the implementation on their chain.

Roles	**Likely Responsibilities involving KNC**
Protocol Maintainer *(For the whole protocol)*	- Governing the unified KNC supply - Protocol specs and treasury - Cross-chain compatibility for KNC
Network Maintainers *(For each blockchain implementation, ie. the network)*	- Governance of implementation, ensuring it conforms to the protocol specs - Determining tokenomic parameters

Protocol Level And Network Level Maintainers

Main Areas Of KNC Usage

Here are the 3 main areas we expect KNC to be used:

Economic Facilitation

KNC is to be used as part of the transaction to facilitate ongoing system operations. For example, getting reserves to pay a fee in KNC for every trade acts as a barrier for them to perform wash trading. This fee can be incorporated into their spread. In addition, users who need instant liquidity can pay the fees in order to get the trade facilitated.

As the scope of Kyber expands, there are several possible mechanisms to leverage KNC as a staking mechanism to acts as a barrier for malicious actors and safeguards the integrity of the Kyber ecosystem. For example, as a requirement for reserves to stake KNC to be considered part of the network, or as a requirement for relayers and validators to facilitate cross-chain transactions between Kyber based networks on different chains.

Governance Token

KNC will be utilized for governance at the protocol level and for the networks on different chains. For protocol level specs and initiatives, we expect there to be an overall KNC based DAO which will help to make decisions for the overall spec and funding of proposals.

For individual implementations on the various chains, there will be a host of tokenomic parameters, upgrade decisions, token listings, which KNC stakeholders will in turn be able to vote and decide on.

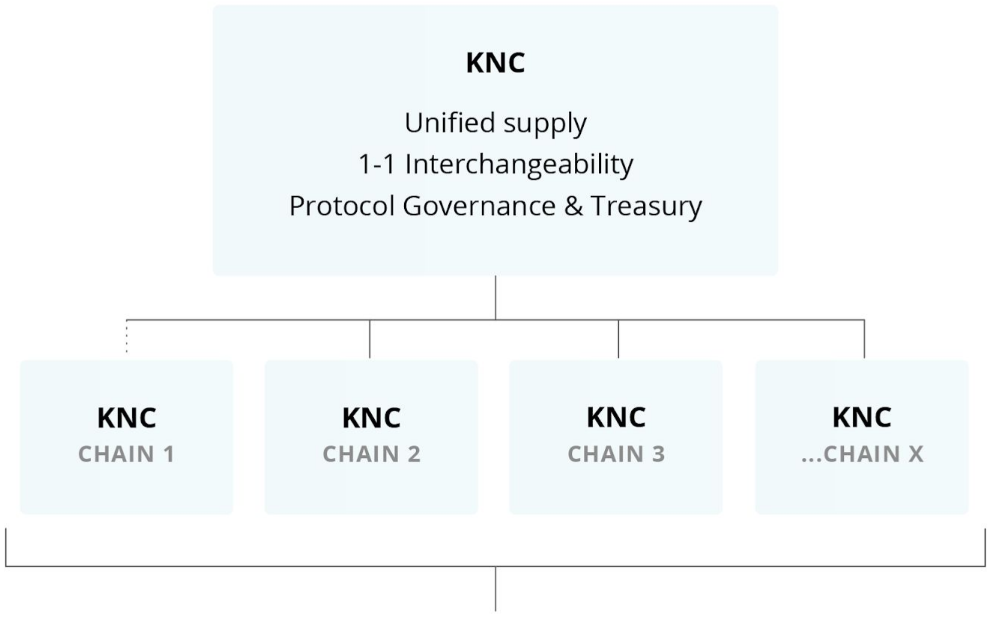
Treasury Funds

On top of the economic and governance of KNC, we expect there to be a KNC based treasury that network maintainers and the DAO members can leverage as a source of funds to fund development, marketing and growth of the Kyber protocol via open grant programs and proposals.

This will allow a network of the most important contributors, those who drive adoption and those who build the core platform to earn part of their income with KNC. By getting KNC into the hands of contributors, and giving them an economic interest in the success of the platform, we move towards a sustainable decentralized community and system.

KNC As One Single Token

While KNC will get represented on different chains (via relays), it will be managed as one single token in terms of a unified supply, interchangeability and in terms of being used for protocol governance and treasury.



KNC representations on different chains

Unified Supply Across Chains

KNC supply will be managed as one economic concept. Any event which might require transfer of KNC from the Ethereum chain to another chain, or changes to the unified supply will have to be approved by the DAO / protocol maintainers, and are only likely to occur when the protocol gets implemented on a chain.

Interchangeability Between Chains

We are striving for interchangeability between chains so that KNC tokens can be easily moved from one chain to another. As such, we are exploring various cross-chain technologies (e.g. PeaceRelay and Waterloo) that can enable this functionality in a way that is technically feasible.

Protocol Level Governance And Treasury

Apart from governing the unified KNC token supply and making decisions on the protocol specifications, protocol maintainers also govern a treasury to fund research and development efforts for adoption of the protocol on different blockchains, or for KNC token interoperability and other causes.

Moving Forward

As the Kyber system and community grows, KNC will play an ever more crucial role in ensuring that Kyber continues to evolve as a cohesive economic and purposeful community.

We look forward to including more details about KNC, particularly the protocol level governance, managing KNC ledger and supply across-chains, and the technical workings of the DAO and treasury in future papers.

VI. Specifications

Protocol Functions

The methods below are all implemented in smart contracts, and are used to interact with the protocol smart contracts.

Token Trades

The contract functions below are executed by the taker to view the expected token conversion rate, and to make a trade request.

getExpectedRate

The `getExpectedRate()` function is called to query for the expected and slippage token exchange rate of the `src` token to the `dest` token.

Parameter	Type	Description
`src`	`address`	`Source token address`
`dest`	`address`	`Destination token address`
`srcQty`	`uint`	`Amount of source token`

Figure 7: Input Parameters Of getExpectedRate() Function

Parameter	Type	Description
`expectedRate`	`uint`	`Expected token exchange rate`

Figure 8: Return Parameter Of getExpectedRate() Function

A rate of zero signifies that a trade from the requested source token amount to the destination token is not possible at the moment.

The rates returned are in precision values regardless of the source or destination's token decimals. We give an example below.

Example

Token	Token Name	Token Symbol	Token Decimals
`src`	`Gifto`	`GTO`	`5`
`dest`	`Zilliqa`	`ZIL`	`12`

Figure 9: Token Information For Conversion Rate Example

Suppose a taker queries for the GTO to ZIL conversion rate, and the number 1219076623416000000 is returned. We assume that the precision value is 10**18.
1219076623416000000 / (10**18) = 1.219
This means 1 GTO token can be exchanged for 1.219 ZIL tokens. Hence, we see that even though the source and token decimals are different, the returned rate is in precision values.

trade

The `trade()` function is called to trade from one token to another. The converted tokens will be sent to `destAddress`.
The implementation of this function should consider all security checks to prevent rate manipulation from the reserves and/ or incorrect implementation of the reserves to make sure the trade happens as expected for taker.

Parameter	Type	Description
`src`	`address`	`Source token address`
`srcAmount`	`uint`	`Amount of source token`
`dest`	`address`	`Destination token address`
`destAddress`	`address`	`Recipient address for destination tokens`
`maxDestAmount`	`uint`	`Maximum amount of destination tokens to convert to. Any excess source`

		tokens after conversion are returned to the sender.
minConversionRate	uint	Minimum conversion rate; trade is cancelled if rate is lower. This parameter protects the user from rate slippage while the transaction is being mined or being included in the blockchain.

Figure 10: Input Parameters Of trade() Function

Parameter	Type	Description
	uint	Actual amount of destination tokens traded to

Figure 11: Return Parameter Of tradeWithHint() Function

Adding and Removing Reserves and Token Pairs

The functions below are executed to add or remove new reserves, and enable or disable token trades.

addReserve

The `addReserve()` function is called to add a reserve as a liquidity provider.

Parameter	Type	Description
reserve	address	Reserve contract address
filterFlag	boolean	Dependent on protocol implementation to decide how to use this flag

Figure 12: Input Parameters Of addReserve() Function

Parameter	Type	Description

	bool	Returns true if operation is successful, false otherwise

Figure 13: Return Parameter Of addReserve() Function

removeReserve

The `removeReserve()` function is called to remove a reserve from the protocol.

Parameter	Type	Description
reserve	address	Reserve contract address
index	uint	Index of reserve in the reserve array

Figure 14: Input Parameters For removeReserve() Function

Parameter	Type	Description
	bool	Returns true if operation is successful, false otherwise

Figure 15: Return Parameter Of removeReserve() Function

listPairForReserve

The `listPairForReserve()` function is called to denote which token pairs are supported for each listed reserve.

Parameter	Type	Description
reserve	address	Reserve contract address
token	address	Token address
buyToken	boolean	True: Buying tokens enabled False: Buying tokens disabled

sellToken	boolean	True: Selling tokens enabled False: Selling tokens disabled
add	boolean	True: Enable token pair trade False: Disable token pair trade

Figure 16: Input Parameters Of listPairForReserve() Function

Parameter	Type	Description
	bool	Returns true if operation is successful, false otherwise

FIgure 17: Return Parameter For listPairForReserve() Function

Reserves Interface

Any reserve that wants to provide liquidity for protocol users will need to implement 2 methods for connecting to the main protocol contract.

getConversionRate

The `getConversionRate()` function is implemented by the reserve for the main protocol contract to obtain token exchange rates.

Parameter	Type	Description
src	address	Source token address
dest	address	Destination token address
srcQty	uint	Amount of source tokens

Figure 18: Input Parameters Of getConversionRate() Function

Parameter	Type	Description
	uint	Token conversion rate

Figure 19: Return Parameter Of getConversionRate() Function

The conversion rate returned is in precision values regardless of the source's or destination's token decimals, as explained above in the `getExpectedRate()` function.

In addition, if the reserve is unable to fulfill the trade request (Eg. does not hold enough destination tokens for the taker to exchange to), it is expected to return a rate of zero.

trade

The `trade()` function is called by the main protocol contract when the reserve is selected to fill the user trade order. Notice that this function differs from that of the taking of orders.

As it is an interface, the network contract expects reserves to implement this trade() function in order to actually use the `srcAmount` of `srcToken` the network sent to, calculate the amount of `destToken` regarding `conversionRate` and send them to `destAddress`, `validate` param is up to the reserve to use.

Parameter	Type	Description
`srcToken`	`address`	`Source token address`
`srcAmount`	`uint`	`Amount of source token`
`destToken`	`address`	`Destination token address`
`destAddress`	`address`	`Recipient address for destination tokens`
`conversionRate`	`uint`	`Actual token conversion rate`

`validate`	`boolean`	`True: Additional validations are performed`
		`False: Additional validations are not performed`

Figure 20: Input Parameters Of trade() Function

Parameter	Type	Description
	`bool`	`Returns true if operation is successful, false otherwise`

Figure 21: Return Parameter Of trade() Function

The reserve is part of the network of reserves once it has been added and its token pairs listed for trading.