A distributed, open-standard protocol for decentralized derivative trading.

Rawad Rifai, Brett Hayes
taurus0x.com

April 8, 2018

Abstract

We describe a distributed, open standard protocol that facilitates decentralized trading of digital asset derivatives. The protocol intends to standardize derivative trading and provide a distributed blockchain backend for persisting and executing contracts. The architecture adopts an off-chain/on-chain approach. Contracts are generated and signed securely offline, and may be transmitted over any medium. The protocol leverages multisig contract technology and Elliptic Curve Cryptography in the form of ECDSA [1] to generate and verify digital signatures. The protocol business logic is maintained in a system of smart contracts on the Ethereum blockchain. The protocol operates in two modes: Peer-to-peer and Exchange, allowing exchanges and/or independent dApp developers to utilize its functionality. Finally, the protocol is an open source project intended to be governed by a proof-of-stake Decentralized Autonomous Organization or DAO [2]. Decentralized governance facilitates continuous integration and development with consensus of the governing the community.

Table of Contents

1 Introduction

It is fair to say that digital currencies more or less have altered the course of money. On January 3, 2009, the digital currency Bitcoin [3] was created introducing a new way of tracking ownership of digital assets. We now call that blockchain. Innovations since then, perhaps most significantly Ethereum [4], have laid the foundation for building further innovations capable of growing borderless economies.

The global financial system in its current form is largely centralized, and that applies to cryptocurrencies. Traders are left with little choice but to rely on a broker to trade assets and asset derivatives. While this model has worked and failed many times, we now see an opportunity in decentralization for trading derivatives peer-to-peer over a blockchain network. Reducing reliance on an intermediary naturally reduces the attached risks.

Our goal with Taurus0x is a protocol that standardizes derivative trading of any digital asset, not limited to cryptocurrencies. The protocol may serve centralized and decentralized exchanges, or any independent dApp. Taurus0x protocol allows any entity to issue contracts secured by asymmetric cryptography. Contracts are persisted downstream of the issuer on the Ethereum blockchain. The blockchain ultimately provides a single source of truth for the current state at a given moment. This logic is powered by distributed, pluggable smart contracts and governed via a Decentralized Autonomous Organization (DAO).

2 Existing Work

Most of the major derivative trading platforms are centralized today, such as CME [5] and CBOE [6]. Exchanges host their own data, matching algorithms and follow their own proprietary message format. While there are vast advantages construed in centralized exchanges, most significantly performance and scalability as of today, there are critical technical and economic risks attached to centralization of data and business logic. Some of these risks are data breaches, market monopolization, and trust invested in third party entities with potential insider malicious activity. This led to fragmentation of markets and economies on a larger scale as they become more silo-ed as a result of market growth and competition.

Ethereum has provided a medium for the community to develop dApps that can handle pretty much any logic handled in mainstream programming languages, such as Java. Smart contracts live in the Ethereum Virtual Machine (EVM) [7]. The EVM is, by design disconnected from the outside world, i.e. any service that is not implemented within the Ethereum blockchain. This led to the concept of oracles. Oracles mitigate this limitation by providing a communication mechanism between the EVM and the internet. Oraclize [8] is one successful implementation of distributed oracles.

Decentralized exchanges leverage Ethereum's smart contracts to perform derivative trading and asset swaps. Full decentralization alleviated the risks of centralization but also came at an expense, most significantly performance and transaction costs. Ethereum network charges gas every time a data modifying transaction is issued [9], therefore offsetting the infrastructure cost to service consumers.

We see a preliminary solution, as of today, in building hybrid systems, i.e. apply decentralization where it makes sense as opposed to abrupt migration from its polar opposite. Off-chain/on-chain combinations have the potential of providing a great stepping stone into full decentralization. Some protocols and implementations, like Lightning [10] and 0x [11], have proven successful in adopting a similar thought process.

3 Protocol Identity

Secure: Test security rigorously.

Taurus0x shall undergo continuous security testing and compliance as part of being production ready.

Decentralized: Forgo a central authority.

Taurus0x does not have nor rely on a central authority, therefore no central point of failure.

Simple: Make it easy for developers.

We are building a solution that embraces a wide spectrum of levels of knowledge and expertise. Simplicity and ease of use is a key factor in designing Taurus0x.

Transparent: Open-source our work.

Transparency equals trust. We plan on open sourcing the entirety of the Taurus0x protocol. We welcome and encourage talents to participate in product enhancements and/or build their own clients to communicate with the core protocol.

4 Protocol Specification

Taurus0x protocol facilitates management of multi-sig contracts off-chain and on-chain. The protocol can run in two modes: Peer-to-peer and Exchange. The two modes have slightly different message formats to adapt to the environment. The protocol supports initially 3 types of derivative contracts i.e. call, put and binary. The architecture is modulated enough to allow adding virtually any type of digital contracts.

4.1 Modes of Operations

4.1.1 Peer-to-peer Mode

In a peer-to-peer mode, contract participants do not rely on an intermediary throughout the whole contract lifecycle. Contracts are multi-sig and are generated and signed locally/offline by the contract maker. Signed contracts may be transmitted to a Taker over any network (text, email, social media, etc.). Once received, the Taker provides their signature for the contract. With both signatures available in the contract, it may be published to the Ethereum blockchain by either party.

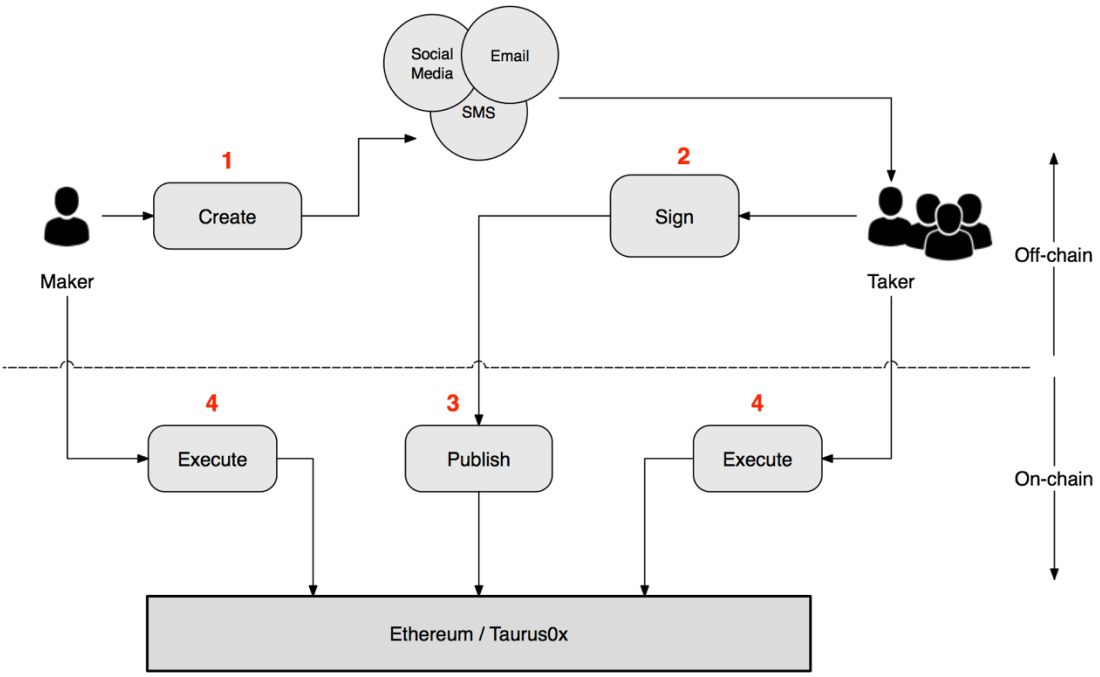

Figure 1 – Peer-to-peer Mode

1. Maker generates a contract off-chain and signs contract with a private key.
2. The Taker joins the contract by signing it with their private key.
3. The contract may be published to the Ethereum blockchain by any party with both signatures.
4. Both parties may execute the contract prior to its expiration date.

4.1.2 Exchange Mode

In Exchange Mode, contract participants communicate via a moderator for transaction completion. The moderator can be any relayer hosting order books, most likely an exchange. The moderator is responsible for the following tasks:

- Matching algorithms
- Relaying orders
- Verifying and collecting signatures
- Publishing contracts
- Providing price points for underlying assets

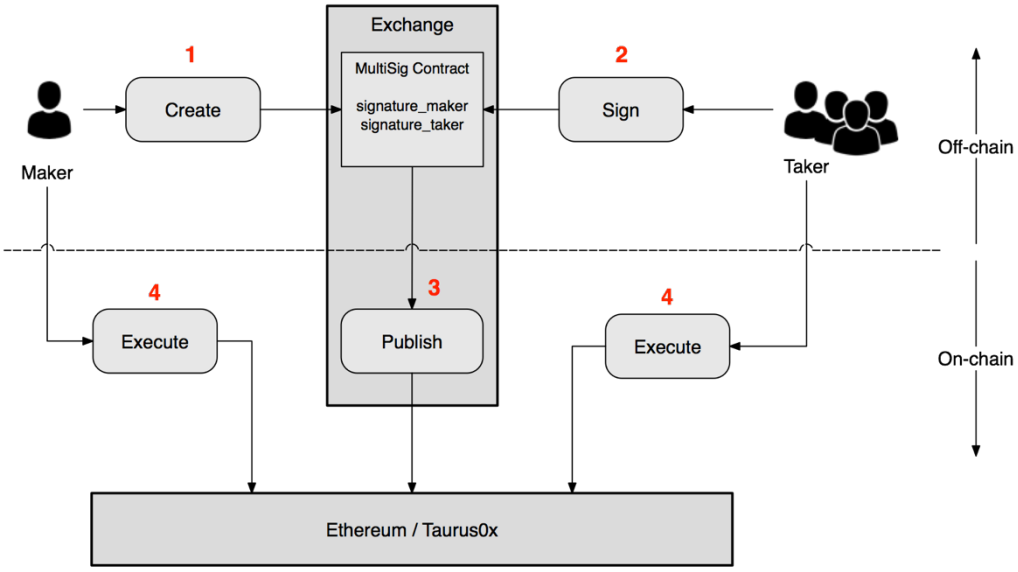

Figure 2 – Exchange Mode

1. Maker generates contract off-chain and signs contract with private key through the exchange.
2. Taker decodes, verifies and signs the same contract through the exchange.
3. With both signatures available, the exchange publishes the contract to the blockchain.
4. Both parties may execute the contract through the exchange, prior to expiration date.

4.2 Message Format

A message is a network-friendly hex string representation of a contract, its Keccak or SHA3 hash, and its ECDSA signature concatenated. The hex message may be transmitted, decoded, verified and validated at the receiving end. The intent of verification and validation is to confirm the identity of the sender and the integrity of the message sent.

Name	Data Type	Description
premium	uint256	The price of the contract in premium_token denomination.
premium_token	address	Address of the token used for premium.
max_margin *	uint256	Maximum margin limiting profit/loss in the contract.
max_marging_token *	address	Address of the token used for max_margin.
index	string	The underlying asset of the derivative contract.
index_feed	string	URL for retrieving index price.
qualifier	bool	Greater than or less than.
starting_price	uint256	Index price at contract creation time.
strike_price	uint256	Index strike price determing contract logic.
expiration_date	uint256	The contract expiration date.
Moderator **	address	The address of the moderator.
sha3	string	Hex format of the message parameters above.
v	uint8	ECDSA signature of the contract parameters.
r	bytes32	
s	bytes32	

* = Call and put only. ** = Exchange mode only.

Table 1 – Message format

4.3 Smart Contracts

4.3.1 Proxy

Taurus0x protocol implements a core smart contract, called Proxy. This contract is the entry point to other contracts. It is responsible for decoding incoming messages and issuing smart derivatives. The proxy must be given ERC20 spending allowance to draft a contract, and it cannot spend more than what is approved. Allowances would ideally match what the participants have already agreed to in the multi-sig contract, off-chain. When the proxy issues a new smart derivative, it moves the allocated allowances into the contract itself. When a contract is executed later, the proxy is not needed. The execution funds and logic live within each contract. The diagram below describes the steps involved in publishing a multi-sig contract through the Taurus0x Proxy.

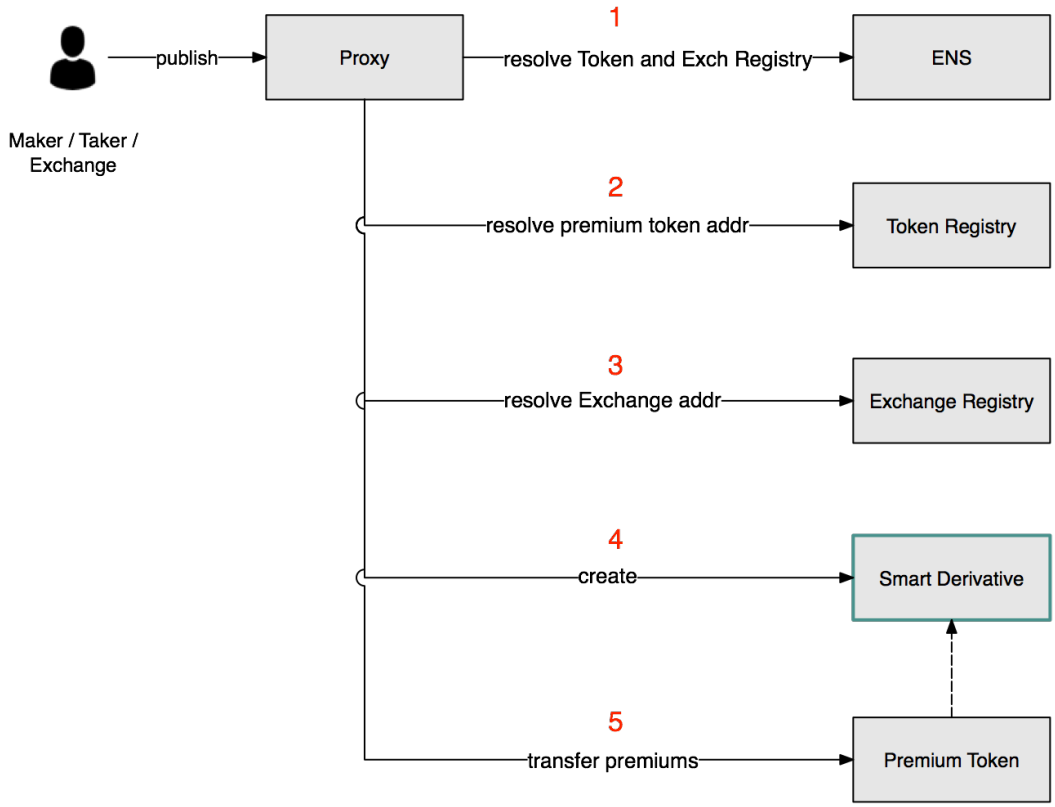

Figure 3 – Publish

1. Taurus0x proxy queries the public ENS resolver to resolve addresses for Token Registry and Exchange Registry.
2. Proxy queries Token Registry to resolve premium token address (example: TAUR => 0x...).
3. Proxy queries Exchange Registry to resolve the contract's exchange address (example: Coinbase => 0x....)
4. The proxy creates a new executable smart derivative contract.
5. Proxy issues the premium token transfer from maker and taker wallets into the newely created smart derivative.

4.3.2 Token Registry

Taurus0x protocol implements another core smart contract, called TokenRegistry. This contract is responsible for storing supported ERC20 token metadata: name, symbol, address and decimals. It is isolated from the proxy in order to separate business logic and data, which makes is smoother for rolling out updates. The proxy relies on the token registry for order completion. The Ethereum blockchain understands addresses not token names or symbols. The token registry is responsible for keeping that mapping. Issued contracts are self-contained and do not need to communicate with the token registry. The token registry requires maintenance by the governing community. Tokens may be added and removed but not edited.

4.3.3 Exchange Registry

Similar to the Token Registry, the Exchange Registry is responsible for storing a list of exchanges that to play a role in the ecosystem. The role of an exchange is to provide prices and it charges a fee in return. The exchange may also act as an order relayer and collects fees separately for that. Contract participants pay the exchange in order to relay their orders and to query exchange prices at the time of execution.

In order for an exchange to collect fees and be listed as a price provider, the exchange needs to insert themselves into the Exchange Registry. The exchange also needs to issue a new price provider contract as described in the next section "On-chain Prices". It is up to contract participants to choose which price provider to use.

4.3.4 On-chain Prices

Taurus0x team will submit a new EIP and ERC standard to the Ethereum Foundation. The goal behind the submission is to set a standard forward for maintaining on-chain asset prices.

Exchanges listed in the Exchange Registry must take the responsibility of maintaining their on-chain prices. Prices come with a TTL (time to live) and are considered invalid after the TTL has passed. Having failed executions will end in lack of trust in a given exchange prices. Price accuracy, fees and the free market will decide how many orders each price provider receives. It is worth nothing that on-chain prices may be used by any other system not just Taurus0x.

The figure below describes how a smart derivative is executed after being published. The proxy is only needed for the publish step and it stores native addresses in the smart derivatives it creates. Therefore, the smart derivative is self contained and does not need to resolve any addresses. The only external information it needs is the price of the premium token, which is maintained in the on-chain Exchange (price provider).

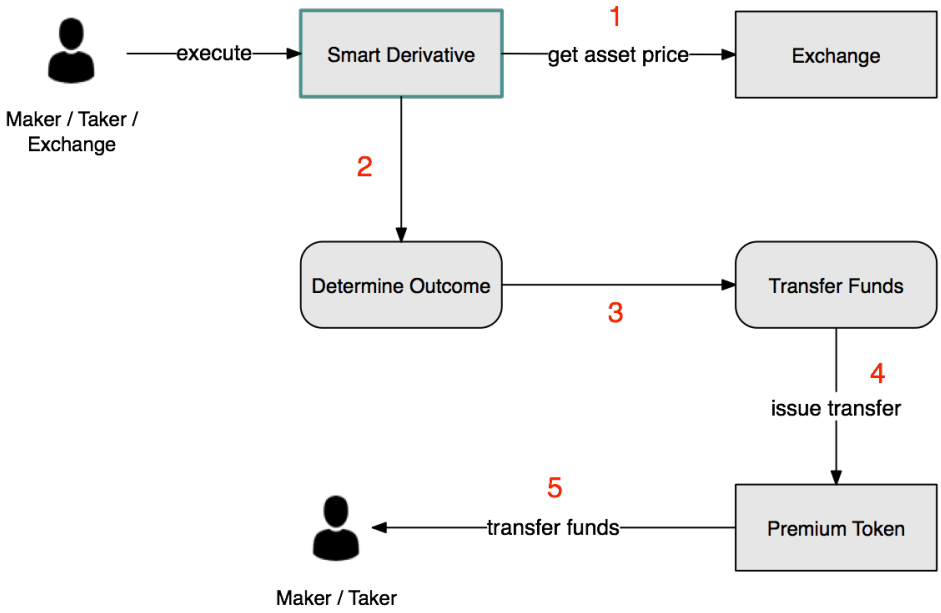

Figure 4 – Execute

1. Smart derivative queries the assigned exchange for latest asset price.
2. The execute logic determines the outcome of the execution.
3. Smart derivative issues a transfer funds request.
4. The transfer funds request is forwarded to the premium token address.
5. Funds are transferred to the Maker or Taker based on the outcome of execution.

4.4 Derivative Types

Taurus0x protocol supports three types of derivative contracts initially: call, put and binary. Binary is straight forward with known profit or loss. Call and put have more sophisticated logic.

4.4.1 Call

A call contract is an offer from the Maker to the market, i.e. potential Takers. The Maker offers a call contract on any digital asset if they speculate that the value of the asset will not appreciate over a specific value in the future, called strike price. The contract Taker speculates otherwise. The Taker pays a premium to the Maker in exchange of sharing potential profits shall the asset appreciate over the specified strike price. The profit sharing is limited by the maximum margin.

The Taker of a call contract has the right to execute the contract any time before the expiration date. It only makes sense to execute if the value of the asset appreciates over the strike price. Upon execution, the contract calculates the delta between the index price at the time of execution and the strike price in the specified ERC20 denomination, should there be any. The delta will be made available for withdrawal by the Taker, up to the maximum margin.

Consider this flowchart:

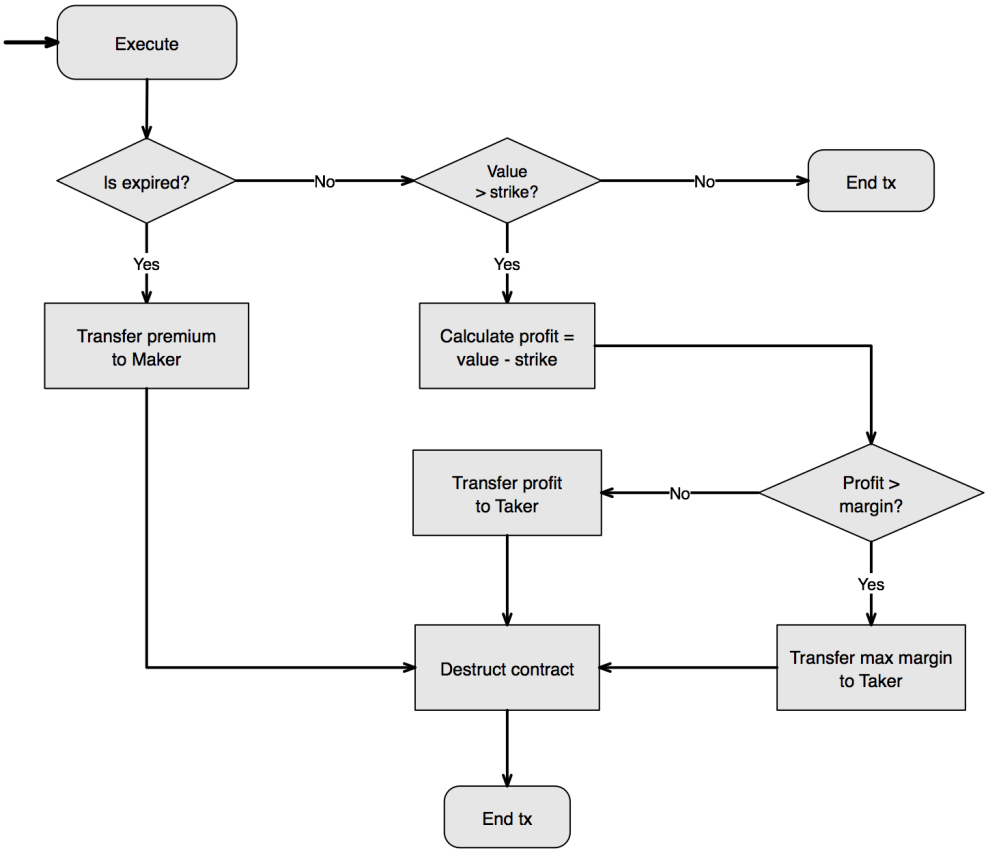

Figure 5 - Call Execute Logic

Sample Call Contract

Consider the following call contract. The contract Maker foresees the value of BTC staying lower than 8,000 USD by June 1, 2018. The contract Taker speculates otherwise. At any point before expiration date, if BTC value appreciates over the strike price, the Taker has the right to execute the contract. In that case, the contract calculates the delta between the strike price and the price of BTC at the time of execution. The contract then transfers the delta from the Maker to the Taker in TAUR denomination, up to 1000 TAUR. Either way, the contract Maker is transferred the premium of 100 TAUR.

Key	**Value**
premium	100
premium_token	TAUR
max_margin	1000
max_marging_token	TAUR
index	BTC
starting_price	7000 USD
strike_price	8000 USD
expiration_date	2018-06-01

Table 2 – Sample Call

4.4.2 Put

Put contracts are similar to buying a zero-deductible insurance policy on an asset. A put Taker pays a premium to guarantee their loss is compensated in case the asset depreciates, up to a defined maximum margin and time in the future.

A put contract is an offer from the Maker to the market, i.e. potential Takers. The Maker offers a put contract on a digital asset if they speculate the value of that asset will not depreciate below strike price, before the contract expiration date. The Taker believes the asset price might depreciate below strike. The Taker pays a premium to have their potential loss compensated, up to maximum margin.

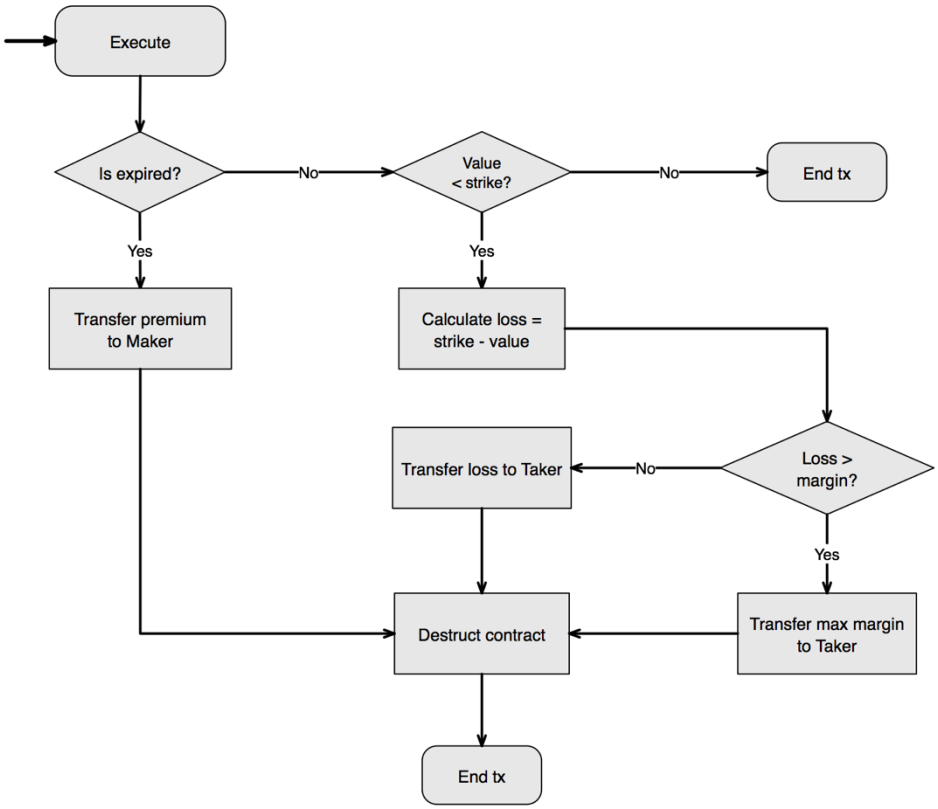

Figure 6 - Put Execute Logic

Sample Put Contract

Consider the following put contract. The contract Maker foresees the value of BTC staying higher than 8,000 USD by June 1, 2018. The contract Taker speculates otherwise. At any point before expiration date, if BTC value depreciates below the strike price, the Taker has the right to execute the contract. In that case, the contract calculates the delta between the strike price and the price of BTC at the time of execution. The contract then transfers the delta from the Maker to the Taker in TAUR denomination, up to 1000 TAUR. Either way, the contract Maker is transferred the premium of 100 TAUR.

Key	**Value**
premium	100
premium_token	TAUR
max_margin	1000
max_marging_token	TAUR
index	BTC
starting_price	7000 USD
strike_price	8000 USD
expiration_date	2018-06-01

Table 3 – Sample Put

4.4.3 Binary

Binary contracts are the easiest type of derivative based contracts to understand for beginners. They allow for direct price speculation of a given digital asset. Binary contracts accept two participants: the Maker and the Taker. The Maker is the entity creating the contract and the Taker is the entity that joins the contract against the Maker. The binary nature of this type of contracts implies that the contract may be won by either party in full, while the opposite party loses in full.

Both participants of a binary contract have the right to execute the contract before the expiration date specified at creation time. For the Maker, it only makes sense to execute if the specified condition has been met. Upon executing a binary contract, the contract checks whether the condition has been met, by comparing the current index price to strike price. If the condition has been met, the smart contract will withdraw the specified allowance from the Taker's wallet and move it to the Maker's wallet. The full premium will be made available for withdrawal by the Maker. If the Maker never successfully executes the binary contract prior to the expiration date, the Taker will be allowed to execute it post expiration date. The outcome of this execution is always the same. The smart contract will use its allowance to withdraw the premium from the Maker's wallet and make it available for withdrawal by the Taker.

Consider the following flowchart:

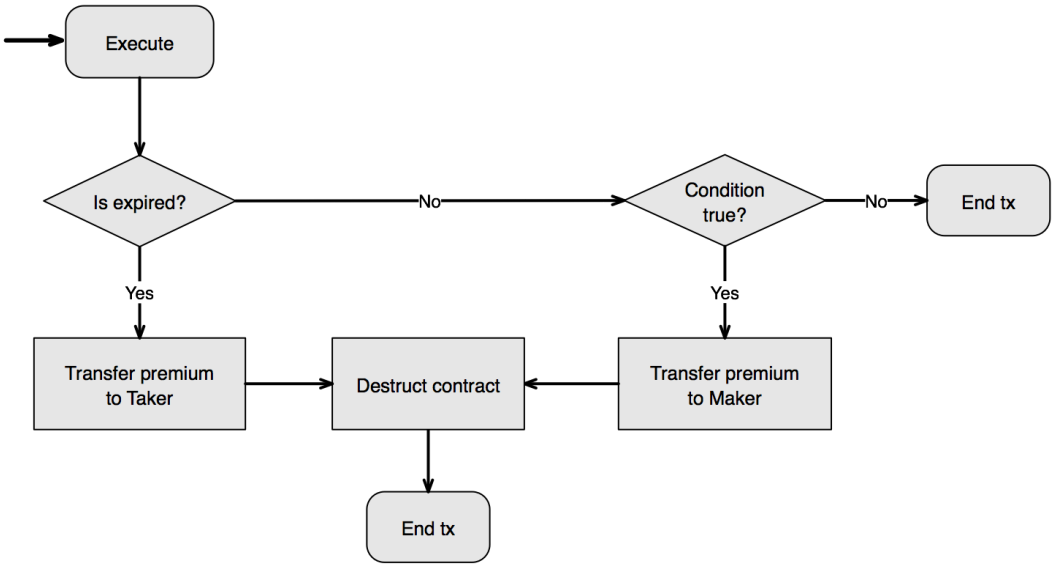

Figure 7 - Binary Execute Logic

Sample Binary Contract

Consider the following binary contract example. The contract Maker foresees the value of BTC at 20,000.00 USD or higher by April 22, 2018. The Maker is allocating 100 TAUR as premium, therefore requires the Taker to allocate an equal amount. If the price of BTC appreciates to 20,000.00 USD or higher and the Maker tries to execute the contract, the contract will transfer 200 TAUR to the Maker. Otherwise, 200 TAUR is transferred to the Taker.

Key	**Value**
premium	100
premium_token	TAUR
index	BTC
qualifier	true *(greater than)*
starting_price	7000 USD
strike_price	8000 USD
expiration_date	2018-06-01

Table 4 – Sample Binary

4.5 Cryptography

Taurus0x protocol leverages Elliptic Curve Cryptography to generate ECDSA signatures. Signatures are implemented according to known DSA standards. Transmitted messages follow the format below:

Hex = Hex(Message)
Message = Contract + Keccak256(message) + Signature(message)
Keccak256(message) = sha3 hash of message content
Signature(message) = Encpk(message)
Encpk(message) = encryption of message using the sender's private key.

When a recipient receives a message the protocol verifies two things:

- **The hash**: to guarantee message integrity and protect against man the middle attacks.
- **The signature**: to verify the sender of the message, which should match a value in the hash.

The following diagram describes the implementation:

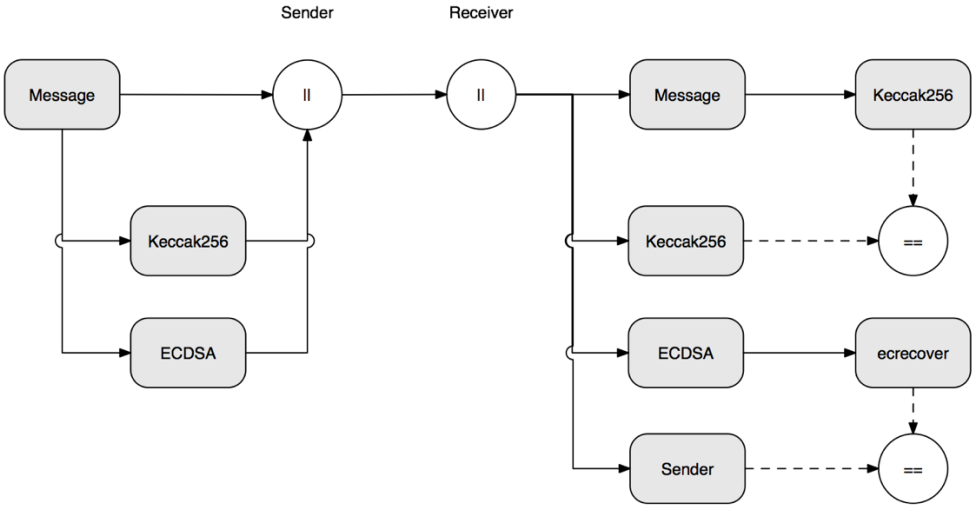

Figure 8 – Cryptography

5 Protocol Token

Governance of a distributed protocol is a business and an engineering challenge. Cryptoeconomic protocols create financial incentives to drive successful protocol governance. This model assumes rational decision making and basic human nature of inhabiting well intentions towards what one owns.

Unlike conventional software, smart contracts on Ethereum are of immutable definition. Once a contract is deployed to the Ethereum blockchain, its code may not be updated by anyone. The mechanism to updating Ethereum-based software is simply by publishing new contracts and re-routing network traffic. This is also known as *forking*, and introduces significant challenges to coordinating protocol updates. Having multiple uncoordinated forks may lead to data fragmentation and protocol inefficiencies.

We propose an ERC20 token to be an integral part of operating and governing the protocol. The token has three main use cases which we discuss below.

5.1 Protocol Evolution

We intend to implement a decentralized governance model to allow protocol token stakeholders to govern the future of the protocol. Since the protocol is open source, anyone is free to fork it and deploy a new version to the blockchain. It is up to the stakeholders to determine which version represents and serves the protocol best. It is ultimately up to the users of the protocol to choose where to route their traffic. Choosing a protocol version implies choosing which Proxy to give ERC20 token allowance to prior to publishing a contract.

5.2 Relayer Fees

Exchanges, relayers or independent dApps building on top of the protocol may choose to attach fees to contract engagements they relay. A fee schedule would be listed by the relayer and accepted by the participants. Fees are completely at the discretion of the relayer. The free market in the form of competition among relayers will eventually calibrate contract fees. Fees are per contract and denominated in the protocol token. We do not take any fees. The protocol is completely open source and free to use.

5.3 Price Provider Fees

On-chain price providers may choose to associate a fee to their service. The fee amount is also maintained on-chain. Price providers are the exchanges and the associated fee is completely at the discrection of whoever provides the data. Price provider fees are strictly denominated in the protocol ERC20 token.

6 Summary

- Standardization of protocol messaging formats facilitates inter-exchange, inter-dApp operability and mitigates data fragmentation.

- The off-chain contract agreement reduces the number of needed trips to blockchain, thus mitigates the blockchain scalability limitations.

- The protocol is network-agnostic, therefore contracts may be transmitted from Maker to tAker and vice versa over any medium allowing for hex format transmit of information.

- The protocol decouples assets and derivatives, therefore supports derivatives for any digital asset (cryptocurrency, stocks, etc.).

- Decentralized governance allows for safe and community-based protocol evolution over time.

7 Acknowledgements

We would like to express our gratitude to our mentors and advisors who helped constantly review and provide feedback on our work. We also like to thank the members of the Ethereum community whose innovations help us craft a tokenized, decentralized economy. We would also like to acknowledge the work of teams like Lightning, 0x Project and Oraclize who paved the way for an off-chain/on-chain mentality. Special thanks to Bernard Abdo, Rees Morgan, and Henry Park whose knowledge and expertise helped provide valuable insight throughout this project.

8 Appendix

8.1 ERC20 Token

ERC20 is the technical standard used for smart contracts on the Ethereum blockchain. The standard describes the functions and events that an Ethereum token contract has to implement. Tokens built on Ethereum are technically smart contracts. Developers are advised to implement the ERC20 interface in their token code for simpler integration with other smart contracts. ERC20 provides the following function signatures:

function totalSupply() – get total token supply.
function balanceOf() – get balance of owner.
Function allowance(owner, spender) – get balance spender is allowed to spend.
Function transfer(receiver, tokens) – transfer tokens to receiver.
Function approve(spender, tokens) – allocates spending allowance to spender.
Function transferFrom(sender, receiver, tokens) – sends tokens from sender to receivers.

8.2 Asymmetric Cryptography

Asymmetric cryptography is a cryptographic concept that uses public/private key pairs. Each private key corresponds to one and only one public key, and vice versa. A message encrypted with a private key can only be decrypted with the corresponding public key, and vice versa. Brute forcing attacks on systems adopting asymmetric cryptography are of exponential complexity.

8.3 Decentralized Autonomous Organization (DAO)

DAO is a concept of decentralized ownership and governance of an organization. Implementations may differ from one organization to another. Ideally, decentralized autonomous organizations allow for stake-based voting on product and fiscal directions. The higher percentage of tokens a party owns, the stronger its vote is.

8.4 Ethereum Name Service (ENS)

The Ethereum name service is similar to the familiar DNS (Domain Name Service). Using the ENS, one can reserve a specific name with a .eth extension. That name plus the .eth will resolve to an Ethereum address through the resolver. Using the ENS is critical for any protocol evolution. Contracts may be changed or upgraded, and once they are ready for rollout, the ENS record can be updated to resolve to the new address.

References

[1] **ECDSA**: https://www.maximintegrated.com/en/app-notes/index.mvp/id/5767

[2] **DAO**: https://en.wikipedia.org/wiki/The_DAO_(organization)

[3] **Bitcoin whitepaper**: https://github.com/ethereum/wiki/wiki/White-Paper

[4] **Ethereum whitepaper**: https://www.ethereum.org/

[5] **CME**: http://www.cmegroup.com/trading/bitcoin-futures.html

[6] **CBOE**: http://cfe.cboe.com/cfe-products/xbt-cboe-bitcoin-futures

[7] **Ethereum Virtual Machine**: http://solidity.readthedocs.io/en/v0.4.21

[8] **Oraclize**: http://www.oraclize.it/

[9] **Gas**: http://solidity.readthedocs.io/en/v0.4.21/introduction-to-smart-contracts.html#gas

[10] **Lightning whitepaer**: https://lightning.network/lightning-network-paper.pdf

[11] **0x Protocol**: https://0xproject.com/pdfs/0x_white_paper.pdf