Increasing transparency and innovation in the adoption of two-party contracts

matthew@roberts.pm
Draft V1 - September 2, 2017

Abstract: Decentralized ledgers have vastly complicated the adoption of two-party contracts. By formalizing the way that two-party contracts are carried out, transparency can be increased and new innovation can be encouraged.

1. Introduction

The rise of smart contracts and decentralized ledger technology has led to many new kinds of two-party contracts. Understanding how these contracts work is in customers' best interests; however, the complexity underpinning this technology forces customers to place an enormous amount of faith—and money—in the claims made by an issuer.

If every two-party contract can be enforced using a multitude of different strategies—each with their own pros and cons—and if every provider of a contract claims to offer the customer the same benefits, then how can a customer differentiate between similar but competing contracts?

More so, how does the issuer distinguish between different approaches for carrying out a two-party contract when every issuer claims to be the same? In this scenario, the issuer adversely selects against the market. The result is a lemon market for two-party contracts where none of the individual attributes for a contract's enforcement are represented.

Under such a model, a contract executed on a centralized exchange is considered both as secure and as fast as a contract carried out on a decentralized ledger. Likewise, if the market equally rewards every issuer based on the same claims, then there is no incentive to improve contract enforcement.

What is needed is a new kind of exchange that highlights the unique attributes pertaining to how a contract can be enforced. Such an exchange would create a competitive market for enforcement strategies whereby the market could select exactly the right approach for a given purpose. By introducing such a design, it is my hope that financial transparency within the space can be improved and new innovation can flourish.

2. Two-party contracts

The introduction of general-purpose ledgers has made it much easier to experiment with new types of two-party contracts. Below is a table of some of those contracts.

Contract type	Description	Example transfer strategies
Asset exchange	Exchanging assets across blockchains, consensus systems, and ledgers of all kinds	Most
Rock paper scissors	Two-party gambling.	Atomic transactions
Derivative trading	Futures, forward, information bets, etc.	Atomic transactions, centralized exchange
Options	An option to buy or sell something in the future.	Cross-chain contracts, centralized exchange
Margin Trading	Using leverage to reap more profits or losses.	Centralized exchange
Outsourceable storage	Buying and selling hard drive space contracts.	Micro-payment channels
Outsourceable computation	Buying and selling compute resources.	Micro-payment channels
Outsourceable proxies	Using another host as a proxy over time.	Micro-payment channels
Outsourceable network services	Outsourcing the running of a server to a third-party with a tree of tests to audit progress.	Micro-payment channels
Trustless content delivery/discovery markets	Using AIs on large data sets to discover taste and using that as a way to match against content.	Atomic transactions
Trustless vulnerability markets	Limit orders to seek vulnerabilities for a software component and the ability to match vulnerabilities based on an exploit DSL for contracts and a virtual machine for arbitrary code on a blockchain.	Atomic transactions
Trustless malware markets	Same as above, but based on malware test cases.	Atomic transactions
Trustless affiliate marketing	Dapps that are provably bonded and designed to offer provable affiliate marketing cuts with an integrated sales, payment, and affiliate identification scheme.	Atomic transactions
Decentralized product markets	A standard market for exchanging physical goods or service between parties.	Federated co-signers
Meme derivative markets	Speculations on the adoption or decay of memes across internet services.	Atomic transactions
Algorithmic speed improvement markets	Bulk generate tests against a branch of code and offer a bounty for its improvement.	Atomic transactions

3. Enforcement strategies

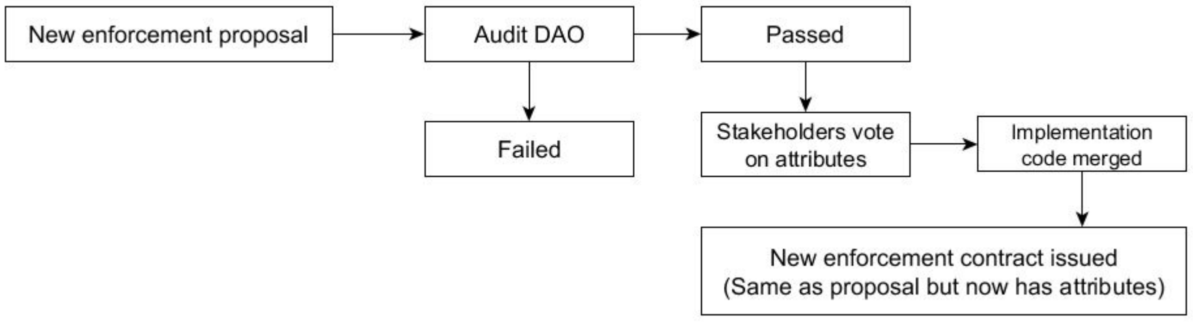
A two-party contract can be implemented using a number of different enforcement strategies. Depending on the type of enforcement strategy used, it will have different tradeoffs. For example, using a centralized exchange offers the customer faster speeds and more convenience than settling on a decentralized ledger, but it comes at the cost of having lower security overall. Selecting the right enforcement strategy better serves customers' needs.

Name	Speed	Security	Privacy	Compatibility	UX	General purpose	Complexity	Pricing	Offline matching	Wallet
Centralized exchange	10	2	2	10	9	10	2	✓	✓	Instant
Cross-chain contracts	4	9.5	5	5	8	1	3	×	×	Low-trust
Micropayment channels	1	8	5	9	8	1	6	×	×	Low-trust
Lightning channels	8	9	5	1	9	5	10	✓	×	Low-trust
SPV-pegs	4	9	5	1	3	10	7	✓	✓	Low-trust
Poor man's peg	8	3	5	9	9	10	2	✓	✓	Low-trust
Rich man's peg (weak ECDSA keys that are brute forced by a network to achieve cross-chain contracts.)	1	7	5	9	3	6	6	×	×	Low-trust
Risk deposit	4	3	5	9	5	9	4	×	×	Low-trust
Real life trading	8	1	9	10	1	10	1	×	×	Physical
Federated co-signers	4	2	1	9	8	9	4	✓	✓	Low-trust
Atomic transactions	5	10	5	2	9	10	2	✓	✓	Low-trust
ZK Proofs	3	4	8	5	4	8	8	?	?	Low-trust
Trusted computing	8	3	2	9	9	9	2	✓	✓	Low-trust
Zero-knowledge proofs, indistinguishability obfuscation, snarks / starks, black box obfuscation, etc	?	?	?	?	?	?	?	?	?	Low-trust

4. Attribute DAOs

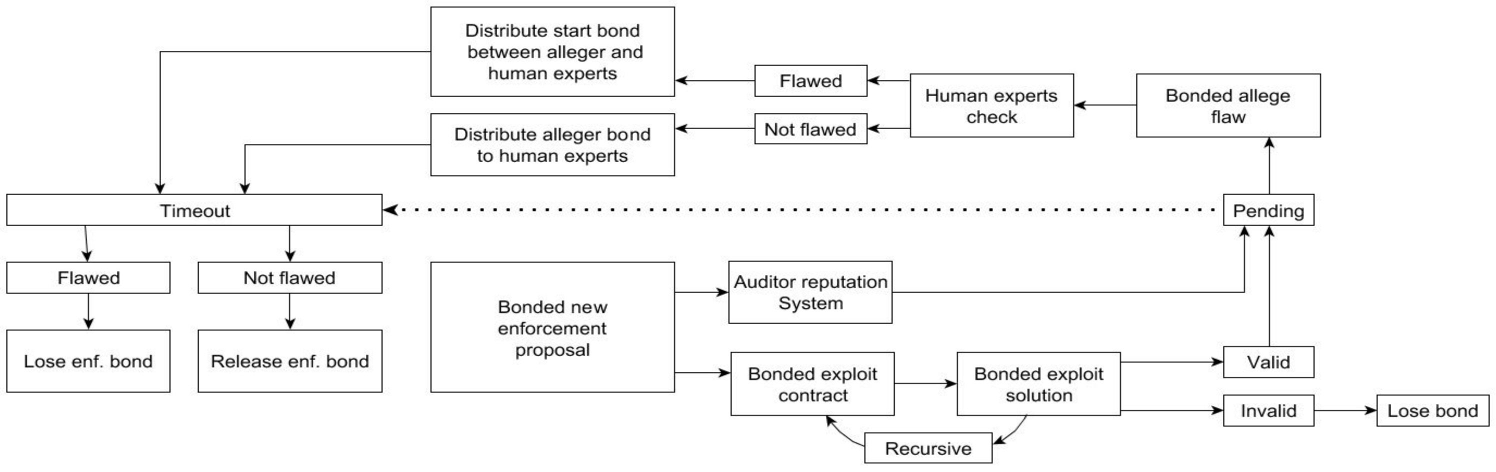
The attribute decentralized autonomous corporation (DAO) takes a new proposal to expand the exchange and allows the stakeholders to vote on its attributes. Every proposal describes an implementation for carrying out a two-party contract with a given enforcement strategy.

A stakeholder's vote is affected by the type of enforcement strategy, the reputation of the developer implementing the proposal, and the quality of the developer's code. The market takes all these factors into account when a vote is cast.



New enforcement proposal
Audit DAO
Passed
Failed
Stakeholders vote on attributes
Implementation code merged
New enforcement contract issued (Same as proposal but now has attributes)

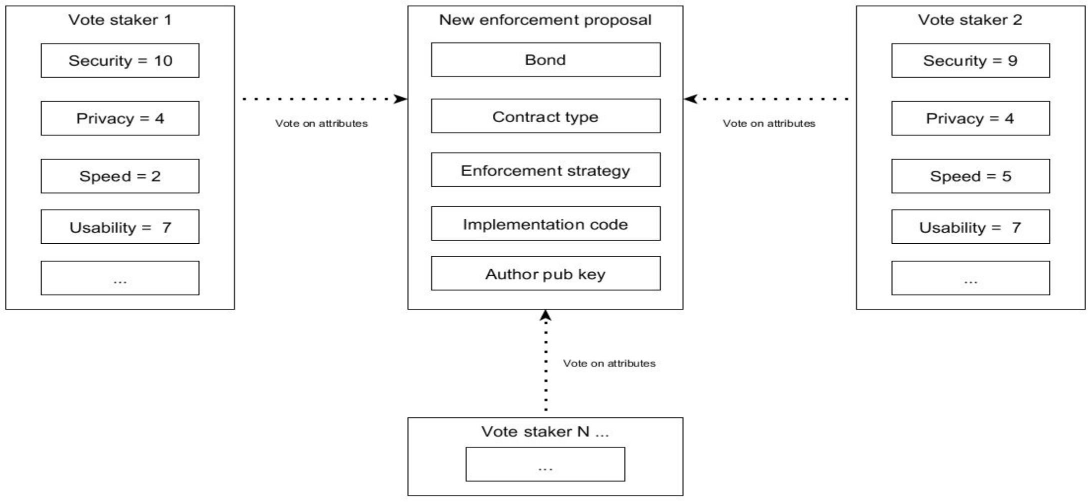
All proposals submitted to the DAO are also submitted to the audit DAO before they are voted on. This helps to ensure that every new extension to the exchange is of a high quality before it is merged into the main exchange code.



New proposals must carry with them a bond, and it is this bond that is lost should the proposal be found to be flawed. Every proposal has a standard set of test-cases that are automatically turned into a bonded exploit contract using the original proposal bond (where solutions are also bonded exploit contracts against themselves).

This offers automatic payment for breaking the implementation. A reputation system helps to refute any erroneous claims where a team of expert security auditors have the last say during a dispute.

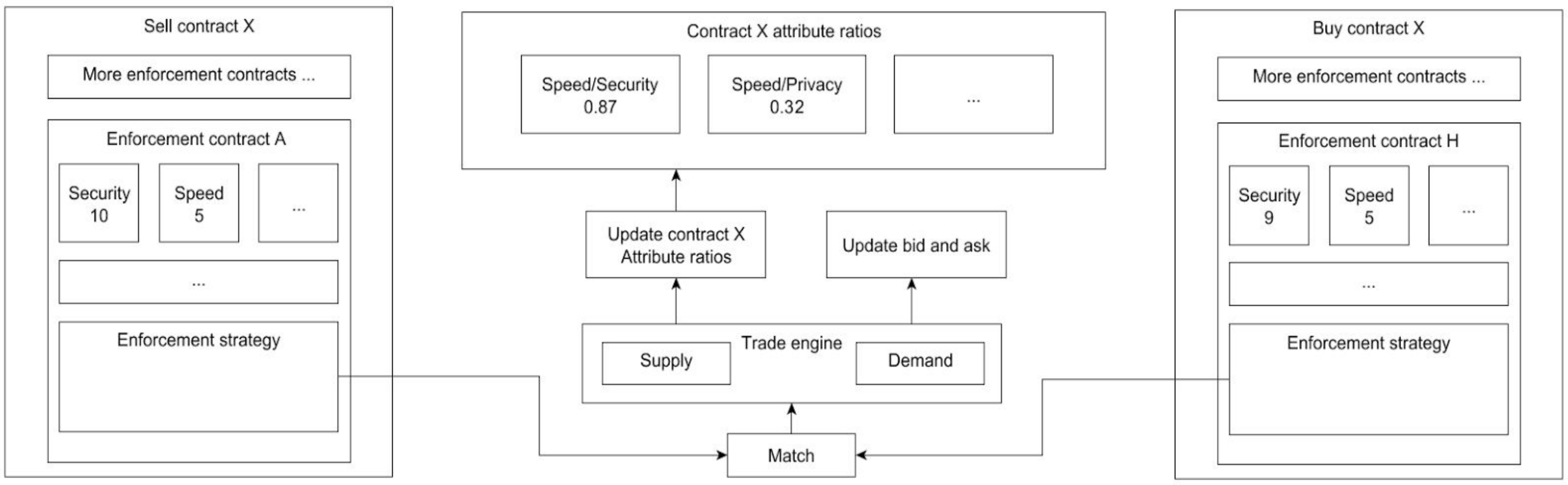
Proposals that have passed the audit DAO can be voted on by stakeholders. Multiple attributes for a proposal are taken into account and the results are summed using a two-phase commit scheme. The final process is to mint a new enforcement certificate. This certificate is a formalization of all the main attributes relating to a given implementation, which is stored on the exchange and then relayed to the user-interface.



5. Attribute markets

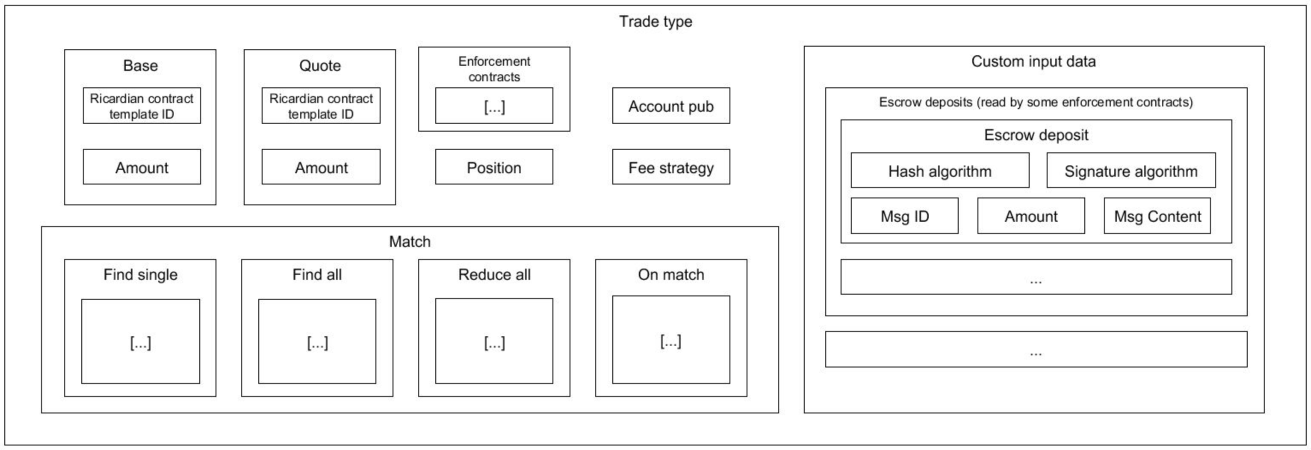
A market uses the intersection of supply and demand to arrive on a price for an asset. This price reflects the market's preference for the type of contract and asset, but it does not take into account the market's preference for its enforceability.

An attribute market updates the ratios between popular attributes based on the preference for how a contract ought to be enforced. This can facilitate new opportunities to speculate on the adoption of enforcement contracts based on factors like speed, privacy, security, and so on.



6. Trade engine

The trade engine is based on an event sourcing design. Input events are fed in and processed, producing output events. As long as the order of events is recorded, it becomes possible to recreate the entire state of the market without having to maintain a slow database.
The trade engine is built for extreme modularity and extensibility.



Every aspect of this design is a module that can be extended as needed. The matching of orders is based on a simple pattern of map-reduce and different types of triggers for new events. Various fee strategies can be described to improve flexibility. However, the trade engine itself does not define any way to carry out a contract, instead deferring this activity to a formalized Enforcement Contract that describes its implementation and trade-offs.

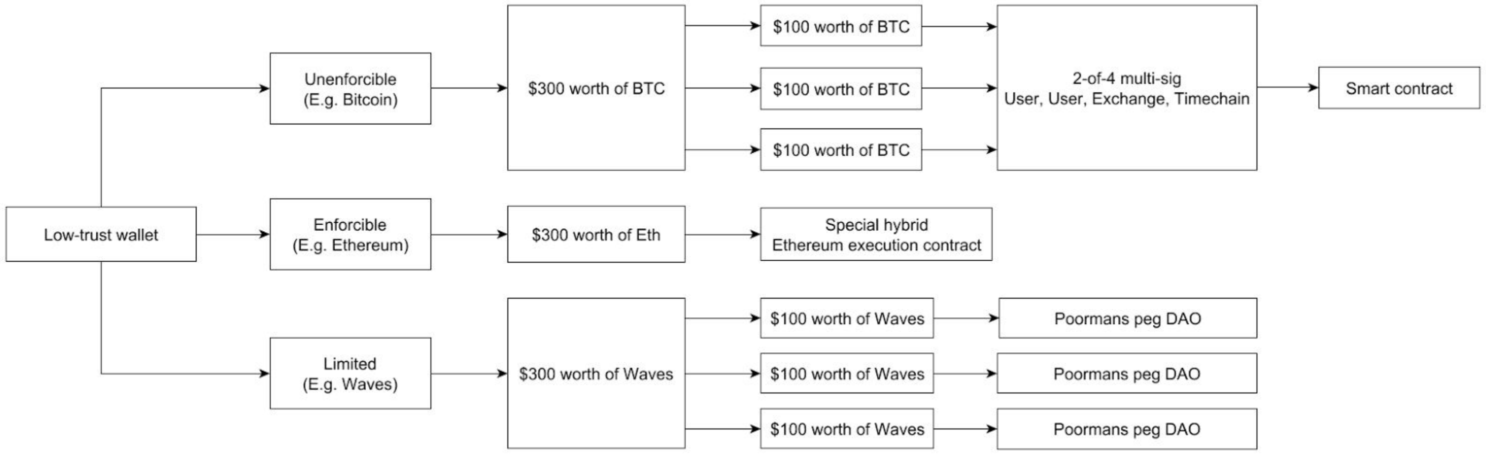
The definition of an asset in this exchange is formalized using a Ricardian Contract Template. This offers more flexibility to precisely describe different yet similar types of assets. Finally, the addition of a field for custom inputs allows every trade to be programmable and customized based on a given enforcement strategy and/or contract type.

7. Ledger-based fund allocation

Decentralized ledgers that use smart contracts as an enforcement strategy, where the outcome is not guaranteed to occur (such as in micropayment channels or cross-chain contracts), should be allocated into a temporary escrow account to reduce time-wasting attacks.

Enforcement strategies with a guaranteed outcome (such as Ethereum-based contracts) can be allocated to a special exchange contract where the potential for time-wasting attacks does not exist. More limited ledgers that don't have multi-sig will need to use a collateral-bonded "poor-man's peg" to allow a secret sharer to enforce agreements.

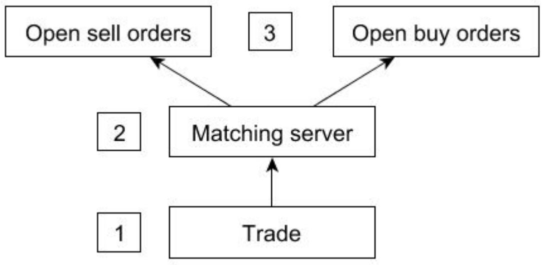
Fund allocation for more simple cases, such as in a centralized strategy, is not considered as this can be guaranteed to occur by the software.



8. Partial matching

8.1. Centralized matching

The most common type of matching is centralized matching, which gives the exchange the full ability to implement a contract. The use of centralized matching is preferable to a decentralized approach due to multiple flaws found in protocols like 0x and for the potential for new flaws to be found in more advanced designs like Omisego. The Omisego design seems to have reduced the potential for flaws like front-running to occur (a kind of insider-trading), but there may still be other places where information asymmetry could exist.

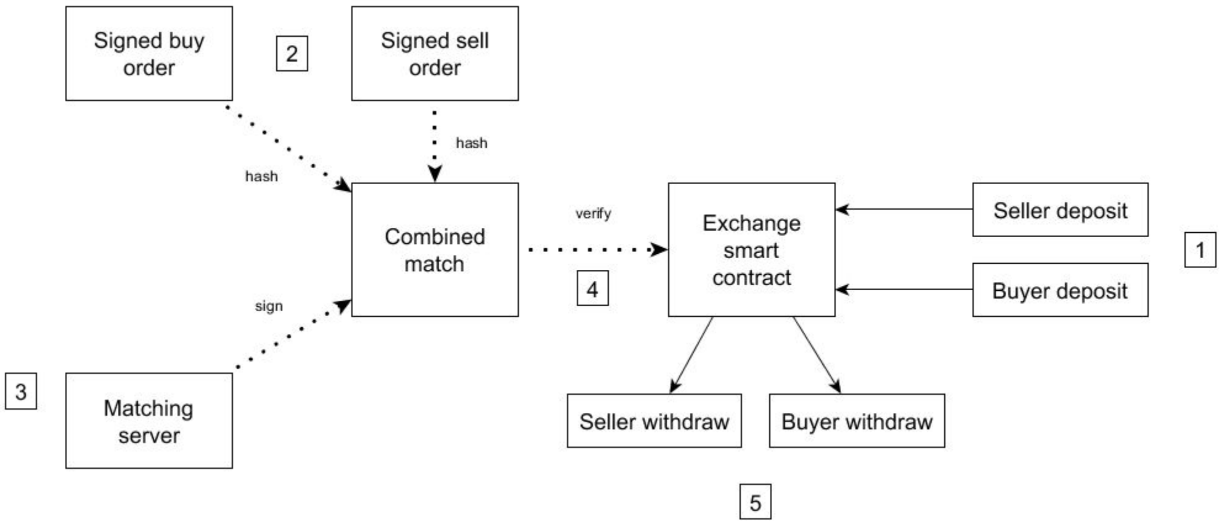


Decentralized exchanges like Etherdelta, Omisego, 0x, etc, sacrifice the ability to easily calculate the price of a two-party contract and/or offer traders the best possible price through limit-based orders, and in return, they gain the ability to prevent orders from being censored.

Since the same benefits can be gained by using a p2p gossip protocol to force an engine to match orders, there is little benefit to be had from using a decentralized design

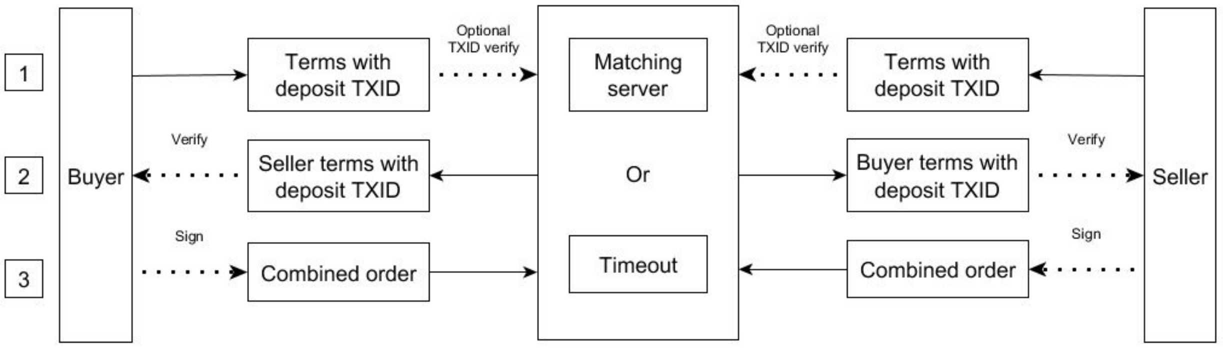
8.2. Overlapping matching

Overlapping matching is the preferred approach to matching ERC-20 tokens on-chain. In this design, both sides deposit their tokens into a special exchange contract and sign messages to authorize their intentions off-chain. These messages cannot be matched directly without a matching server participating, so it is easy to calculate the price.



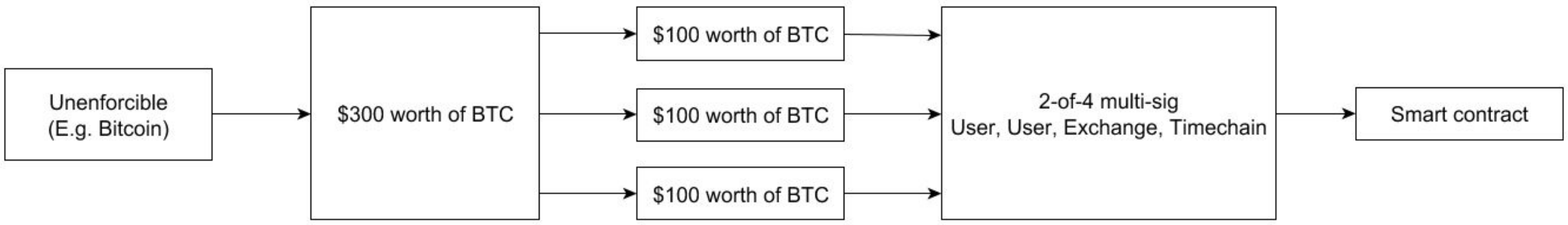
8.3. Timed matching

Timed matching refers to matching contracts where only some of the parties are required to validate whether or not funds have been allocated prior to contract enforcement. The matching server can defer validation of on-chain escrowed funds to both sides, and as long as they both agree, the server doesn't have to download any blockchain software. A slight variation is to have one or both of the sides offline. In this case, the server validates the offline side(s).



8.4. Chunked matching

For on-chain assets with limited potential for complex contracts (like Bitcoin) and which follow the UTXO (Unspent Transaction Output) model, chunked matching allows for partial matching of funds between multiple types of enforcement strategies.



With chunked matching, funds are split up into divisible quantities equivalent to an arbitrary market price of X USD dollars. Multiple smart contracts can then be assigned between peers for partial matching of contracts. This will not allow funds to be fully matched due to the rise and fall of asset pairs, with respect to each other, but it should allow for more efficient matching to occur without having a single contract tie up all funds.

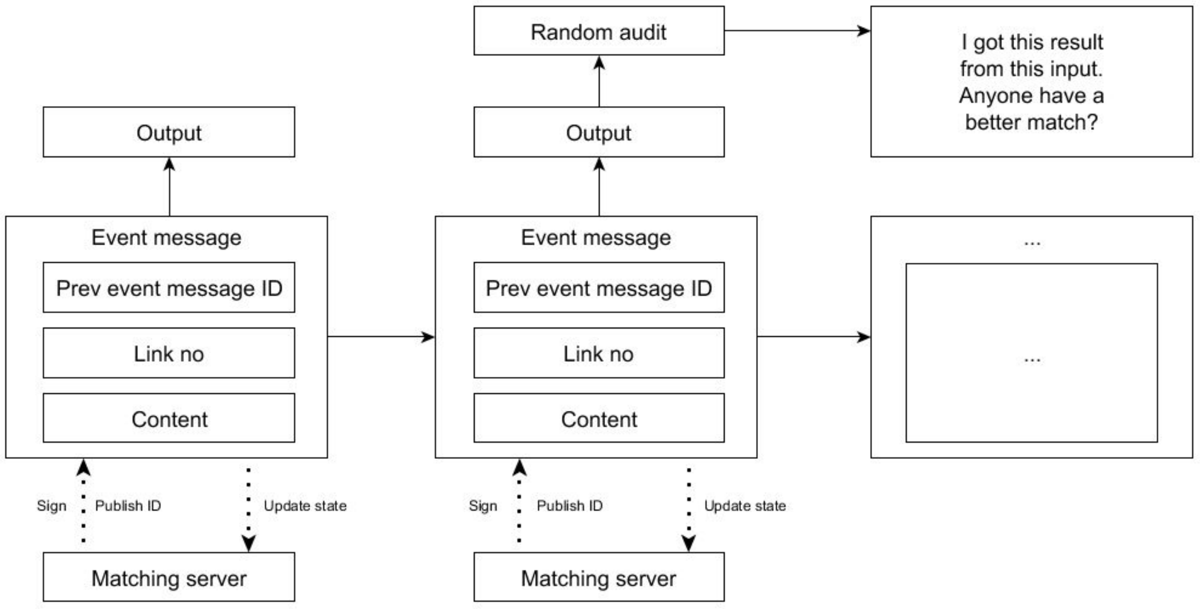
Over time, it may be necessary to “rebalance” the distribution of outputs if the price has risen or fallen by too much. This will be a trade-off between higher transaction costs, slower confirmations, and more efficient matching (which will need to be decided algorithmically).

9. Tradechains

Tradechains offer a simple way to audit whether or not a matching server has accurately carried out a match. Every event received by the server is notarized into a chain of events, and the details of the chain height, signature, and ID are published.

If someone receives a match from the server, it can ask if anyone else has a closer or more compatible match. The use-cases for this reside in provably random matching for a liquidity incentivize algorithm or for complex matching behaviour where the user may request the exchange run massive computable functions against complex orders for a higher fee.

If the user is paying for execution time on the server, the addition of bonded-state transitions make it possible to audit whether or not the server has reliably executed a function. In the future, more complex contracts that deploy AI agents based on massive data sets could reasonably interact with the market across multiple types of data.



10. Infrastructure

Decentralization of the infrastructure for matching servers can be achieved by allocating a server per contract type. To improve the DDoS resiliency, a proof-of-stake system could be developed where a server is randomly selected to match orders based on a lottery draw.

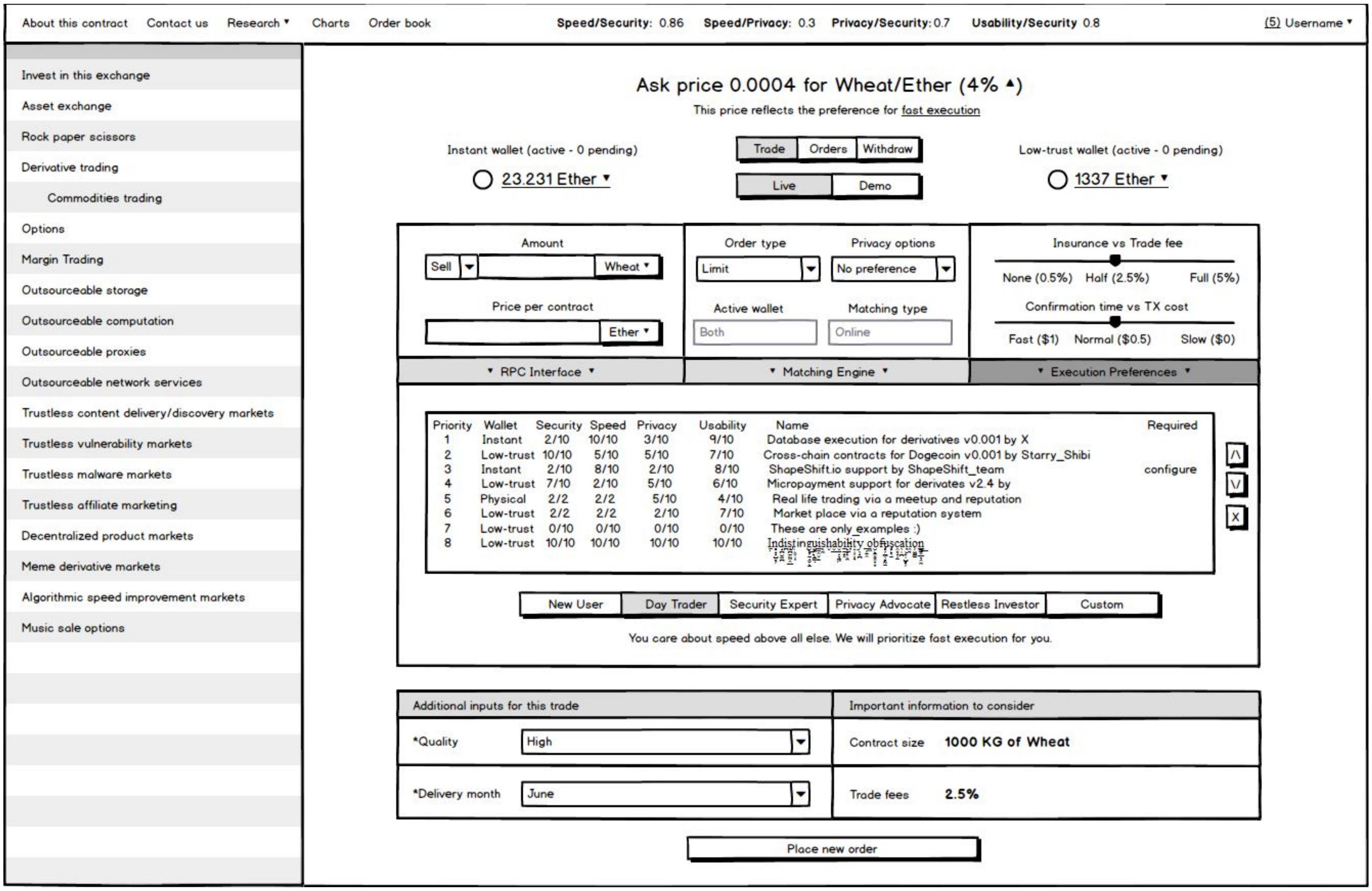
The addition of a bonded matching server would improve censorship resistance too, as it is possible to commit orders on-chain and then prove that an exchange hasn't matched them reliably. If the network detects censorship, it can initiate a new election to choose a new server with the old server losing the bond that it staked to participate.

Synchronization between servers could be done more efficiently with sorted order books. Where all the sell orders are sorted from highest to lowest, and all the buy orders are sorted from lowest to highest. It may be possible to prevent front-running with this design by considering a combination of financial cryptography and off-chain consensus systems, e.g., provably random draws or homomorphic encryption; failing that, reputation will have to be used.

11. User-interface

The user-interface allows the exchange to adapt to the needs of the user. Because the execution of a contract has been formalized as an enforcement contract, the exchange can easily adapt to the needs of the user.

The exchange has three main types of wallets. The instant wallet (centralized), the low-trust wallet (decentralized-ledger), and the physical wallet. Contracts can be matched across some or all of these wallets based on the enforcement contracts chosen. The enforcement contracts can be chosen dynamically based on a pre-set profile or customized as needed.



Speed/Security: 0.86 **Speed/Privacy:** 0.3 **Privacy/Security:** 0.7 **Usability/Security** 0.8

Ask price 0.0004 for Wheat/Ether (4% ▴)

This price reflects the preference for fast execution

Instant wallet (active - 0 pending): 23.231 Ether ▾

Low-trust wallet (active - 0 pending): 1337 Ether ▾

Amount: Sell ▾ Wheat ▾

Price per contract: Ether ▾

Order type: Limit

Privacy options: No preference

Active wallet: Both

Matching type: Online

Insurance vs Trade fee: None (0.5%) Half (2.5%) Full (5%)

Confirmation time vs TX cost: Fast ($1) Normal ($0.5) Slow ($0)

Priority	Wallet	Security	Speed	Privacy	Usability	Name	Required
1	Instant	2/10	10/10	3/10	9/10	Database execution for derivatives v0.001 by X	
2	Low-trust	10/10	5/10	5/10	7/10	Cross-chain contracts for Dogecoin v0.001 by Starry_Shibi	
3	Instant	2/10	8/10	2/10	8/10	ShapeShift.io support by ShapeShift_team	configure
4	Low-trust	7/10	2/10	5/10	6/10	Micropayment support for derivates v2.4 by	
5	Physical	2/2	2/2	5/10	4/10	Real life trading via a meetup and reputation	
6	Low-trust	2/2	2/2	2/10	7/10	Market place via a reputation system	
7	Low-trust	0/10	0/10	0/10	0/10	These are only examples :)	
8	Low-trust	10/10	10/10	10/10	10/10	Indistinguishability obfuscation	

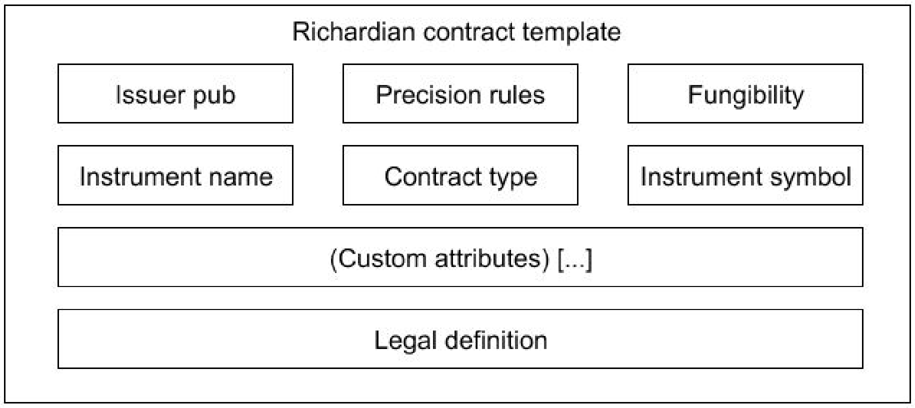
You care about speed above all else. We will prioritize fast execution for you.

Additional inputs for this trade		Important information to consider	
*Quality	High	Contract size	**1000 KG of Wheat**
*Delivery month	June	Trade fees	**2.5%**

Special attention to detail needs to be placed in the design of buying and selling non-fungible assets as these are more suited to a marketplace UI than a trading UI. This can be set to occur based on the type of contract and assets expected to operate with it.

12. Ricardian contracts

Due to the complexity of assets and the broad range of two-party contracts, it is necessary to formalize an asset definition as a contract. The Ricardian contract is a formalization of the issuance of an asset that is both machine-readable and human-readable. The contract is structured to be legally enforceable in as many jurisdictions as possible. This provides an additional layer of security as some smart contracts may not be one hundred percent trustless.



A digital signature is a legally valid way to sign a contract under international law; however, in practice, there may be problems with proving the identity of the signer. Therefore, a certificate authority (CA) can be used to issue a certificate that associates a given key pair with an identity so that every signature made by a key can be easily traced back to the person who made it.

Ricardian Contracts need to be archived client-side for auditability reasons. Since Ricardian Contract Template hashes are used as asset identifiers, being able to use the correct hash / ID implies having access to the full contract. To further improve legality of the contract, every two-party contract placed on the exchange should show the appropriate legal text where the parties know that it constitutes a legally binding agreement.

13. Adding two-party contracts

A DAO can be used to add new types of two-party contracts to the exchange. A definition should be formalized based on a Ricardian Contract. This will be referenced within every asset definition so that the full type of contract is legally and semantically defined.

14. Conclusion

I have proposed a simple design for an exchange that reduces the adverse selection of two-party contracts. I started by introducing various types of two-party contracts and then spoke about how they can be carried out. Finally, I formalized their attributes via a DAO and presented them clearly to the user so they can make informed decisions. My design will improve the liquidity of an exchange because unlike existing designs, my design can be used to exchange any type of two-party contract or asset to the satisfaction of any possible type of user.

15. Future work

1. What is the best way to prevent front-running in a design like this?
2. Does it make sense to formalize the matching engine too (on-chain vs centralized)?
3. What other attributes are important in the selection of a two-party contract?
4. What is the best way to model the attributes of a two-party contract?
5. What experiments might prove the existence of a lemon market for two-party contracts?
6. What is the best way to convey two-party contract attributes to the user?
7. What is the best way to legally formalize a two-party contract and show it to the user?
8. What role does insurance play in this process?
9. Who is to blame for a loss as a result of a software bug?
10. How can the security of a fast execution / instant wallet be improved?