Vishakh Follow
May 10 · 12 min read

A deeper look into a financial derivative on the Ethereum blockchain

Have you seen our overview post?

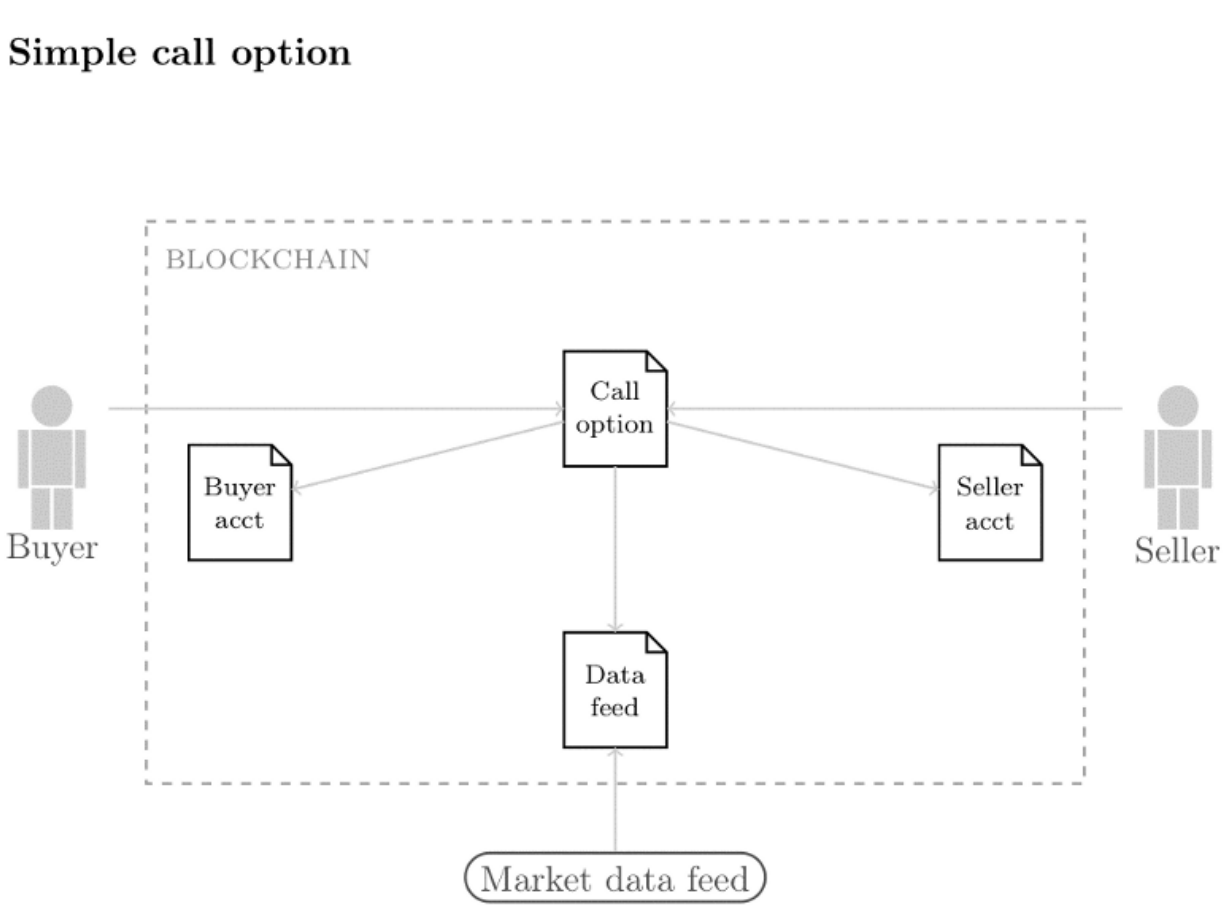
In this post, we describe the way we laid out several smart contracts in order to investigate the implementation of a financial instrument on the Ethereum blockchain. We will also talk about the associated challenges and tradeoffs that would apply to any similar effort. If you haven't done so already, please see our overview post for a general idea of our work. We assume here that the reader is either a developer or generally familiar with Ethereum and its workings.

Our code is on GitHub but we must add a disclaimer that it is the product of casual weekend hacking. All smart contracts are written in Solidity with corresponding web pages serving as front ends. The Web3 library binds the front ends to the smart contracts. We successfully tested the deployment of the contracts via IPFS to consider the optimally decentralized scenario.

Components

We now describe the individual components of our financial contract ecosystem. We will cover two kinds of smart contracts:

1. A simple option over any underlier with cash settlement
2. A derivative which is decomposed into options of type (1) which manages margin requirements.



Price Feeds

Financial smart contracts need some mechanism to refer to external market and reference data in order to take positions on them. In an ideal world, market data publishers like Bloomberg would reliably publish their data on to the blockchain. Alternatively, other market data providers would be incentivized by a reputation system to provide accurate and frequent market data updates. There are no market data publishers reliably pushing data on to the blockchain yet.

When we started out, Roman Mandeleil had just announced the creation of the Ethereum Price Feed so we built our contracts around it. However, as this feed only updates occasionally and we need to run demos which complete in mere minutes, we decided to create a mock randomized price feed. Our feed behaves identically to Roman's price feed, i.e. it's a mapping between some popular financial symbols and their values, but also randomly mutates its values by 10% every time someone asks for a quote. This lets us perturb the value of an underlier a few times during a demo to demonstrate margin calculation and option exercises. Our financial contracts use these feeds to get the prices for their underliers.

Oracles have a centralizing effect on public blockchains. As most assets are traded off the blockchain, there is no direct way to refer to them. We have to rely on prices being pushed in by reputed central authorities and are therefore

subject to the risks that these authorities aren't resilient or well-intentioned. These risks can be ameliorated by using a weighted committee of oracles or allowing oracles to be swapped out but the loss of a reliable provider may still be fatal. Even with an orderly reputation system in place, trading long-tenured instruments would still be risky. Perhaps in the future enough assets will intrinsically be traded on the blockchain that this reliance on external providers is made unnecessary.

Price feeds could work in a much more straightforward manner on private blockchains. As with any trading system, standard market and reference data interfaces could be defined and then plugged into industry-standard feeds on the back end. This would allow providers to be swapped out as desired and also make standardization easier. Participants could also submit their individual quotes and use smart contracts to infer consensus market data.

Trading Accounts

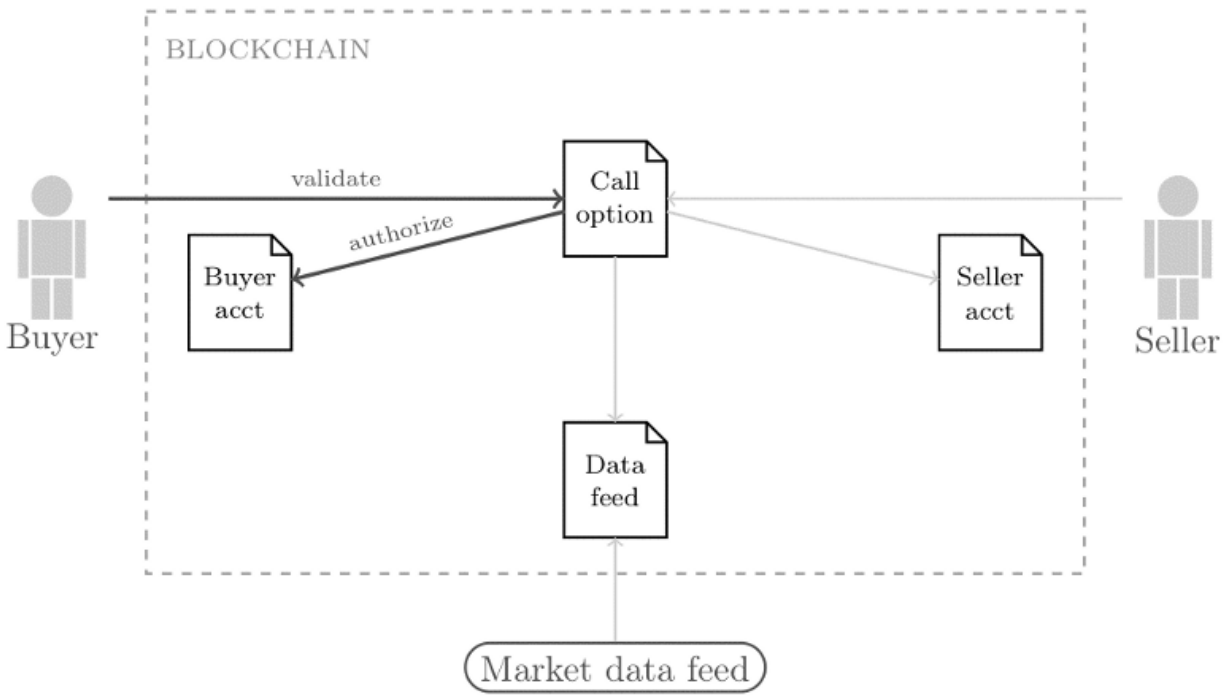
Participants in financial contracts need some proxy to stand in for them on the blockchain. We cannot directly use peoples' wallet addresses as there are specific behaviors each counterparty needs to support in order to trade on the blockchain. Therefore, in order to participate in our financial system, one must create and use a trading account contract. Two of the inputs to our contracts are the buyer and seller's trading accounts.

Once someone creates a trading account, they are obliged to keep it funded adequately to meet margin calls and to make cash settlements. When two counterparties create and activate a financial contract, the contract is implicitly authorized to interact with the counterparties' trading accounts. The financial contract is then able to ask the trading accounts for funds whenever required. Trading accounts can be plugged into a reputation systems to reward good behavior on the blockchain financial system. Trading accounts also provide access control as they ensure third parties are not able to interfere with the actions of the actual counterparties.

Call Options

Our feed-backed call option contract represents an individual call option on any of the underliers supported by the price feed. This contract is designed for modularity and can be used as both a standalone option agreement or as a leg in our call spread (described below). In the former scenario, the buyer and seller of the option interact directly with this contract during its lifecycle. The basic flow of call option initialization, counterparty validation and exercising is illustrated here.

(2) Counterparty validates



Our option and derivative contracts can be initialized by either the buyer or the seller. A contract is initialized by providing it with an address for the buyer and seller's trading accounts as well as the underlier, strike price, notional and date of maturity. When a buyer or seller initializes this contract, it is automatically authorized to interact with their own trading account. To prevent malicious access to funds in a trading account, a trading account contract can only authorize other contracts to interact with it in a function call made by the owner of the account. Therefore, the seller is only capable of authorizing the option contract to access their own trading account, not the buyer's (and vice versa).

The address of the option contract must then be transmitted to the counterparty through an external channel, e.g., over email. In order to make the call option active, the counterparty must 'validate' the contract. This also authorizes the option contract to interact with their own trading account, and consequently it is able to access the funds of both the buyer and the seller. Naturally, the contract cannot be validated if trading account authorization fails for either counterparty. If the contract has not yet been validated, either party may withdraw it, thereby ensuring that it cannot be activated in the future.

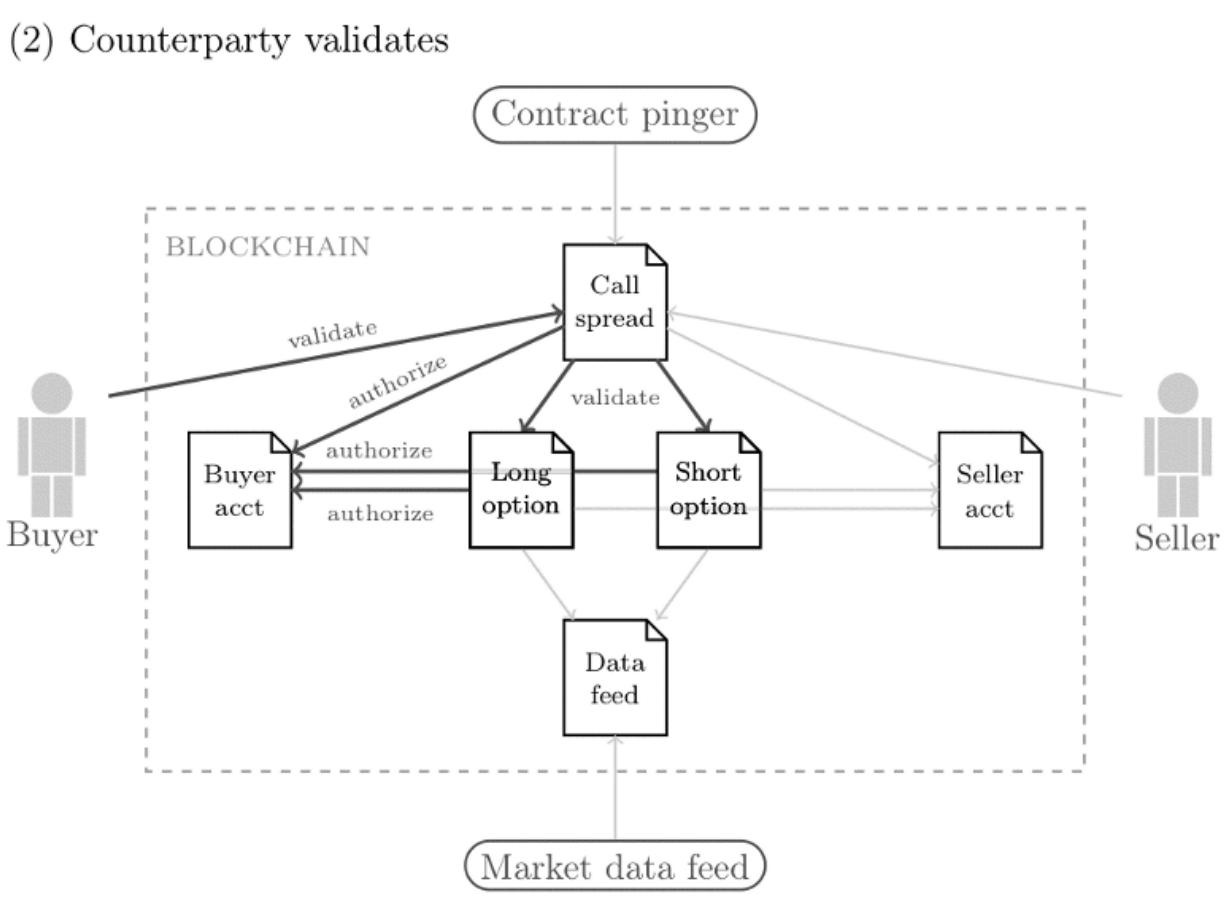
Once active, the option contract cannot be interrupted as long as the blockchain remains resilient. If the maturity conditions are met and the underlier exceeds the strike price, the buyer may now choose to exercise the option, purchasing the underlier at the strike price. We use cash settlement (in

ether) for all our contracts, although they can easily be extended to cover on-blockchain securities. Exercising the option initiates cash flows between the buyer and seller for the full notional, with the seller paying the market price and the buyer paying the strike price.

Our eventual goal is to have a standardized modular interface for all our financial contracts so that more complex contracts can be composed from simpler ones. To approach this, we extend the simple option above to a model a call spread, described below.

Call Spreads

The call spread contract corresponds to a call spread over a single underlier. It is provided with two call options as described above which it uses as its legs. The call spread contract takes over the management of the legs and interfaces with them on behalf of the counterparties. In addition to counterparty-initiated actions, the call spread also maintains a margin account which it rebalances at regular intervals as the value of the underlying security changes. The basic flow of call spread initialization, counterparty validation, margin rebalancing and exercising is illustrated here.



As with the simple call option, either the buyer or the seller of this derivative can initialize the contract on the blockchain. Call spread initialization takes the same parameters as the simple option along with addresses of two unini-

tialized call options, two strike prices for the two legs and a parameter for the amount of margin collected from the seller. The call options are then initialized in turn to represent a long option and a short option (with buyer and seller reversed) which serve as legs for the call spread, thereby authorizing them to interact with the initializing party's trading account. The call spread itself is also authorized with that trading account in order to rebalance margin.

Constructing call options ahead of time and supplying their addresses to the call spread is admittedly a compromise. We also investigated having the call spread directly construct the leg contracts during initialization but the resulting function calls exceeded the current block gas limit of 4,712,388 wei. However, as a single point of initialization is very helpful for multi-contract systems with many moving parts, we pre-construct these sub-contracts and then supply them to the call spread so that all contract parameters originate in a single function call.

Because contract interaction is typically limited to the owner of the contract and set at construction, having one of the counterparties pre-construct the options prevents the call spread from interacting with them. However, allowing non-owners of contracts to interact with them can introduce security holes. Similarly, adding off-blockchain steps to reassign the owner of the options to that of the call spread can be problematic, because a failure in any of these transaction could leave the call spread in a degenerate state. Instead, we modified the standard approach to contract ownership by allowing super-contracts (like the call spread) to claim ownership of sub-contracts (options) with which they share an owner, as long as this is requested within a function call initiated by that owner. The DELEGATECALL opcode was not implemented in a release version of Solidity at the time, but will likely provide an elegant solution to this problem as well.

After the call spread has been initialized, the counterparty must then validate the call spread to activate it. This validation is transmitted to the legs and all three contracts are authorized to interact with the counterparty's trading account. Validation will not succeed if any of the contracts failed to authorize with either of the trading accounts. As with the simple call option, the call spread can be withdrawn if it has not been validated, consequently leading to withdrawal of the legs as well.

Once validated, the call spread is active. Prior to maturity, any party (including a third party) may induce the call spread to check the value of the underlier and rebalance the margin it holds. We use a simplified approach to margin computation approach in order to avoid the limited support for float-

ing point math which is described in our overview post. To ensure that margin is rebalanced regularly, we require an off-blockchain service that we term a *contract pinger*, described below.

Finally, when the call spread is mature, the buyer and seller may be eligible to exercise their respective legs based upon the market value of the underlier and their respective strike prices. The buyer may exercise the long option, which also returns the seller's margin to their trading account, and the seller may exercise the short option independently from the buyer. In our current implementation, the margin is not returned if the buyer does not attempt to exercise (e.g., if the long option is out of the money) but it is straightforward to incorporate margin return based on time, reputation or third-party administration.

Contract Pinger

As we described in our overview, scheduling was one of the primary challenges we faced, as the delivery of cash flows is critical in the financial world. Every contract that requires scheduling implements a ping() method which can be invoked periodically to ensure that all housekeeping, including margin calculation and cashflow generation, is done regularly. As the Ethereum virtual machine has no support right now for scheduled invocation of contract methods, we cannot schedule all lifecycle events during contract creation. Systems like the Ethereum Alarm Clock rely on financial incentives to crowdsource contract invocation. However, they cannot provide the guaranteed execution needed for financial contracts as each invocation is essentially stochastic.

For demo purposes, we created a simple single-page web application which pings a given contract for a given number of times at a given frequency. This serves to both regularly perturb the values of the price feed as well help the call spread process its life cycle events. Outside of a demo setting, a full-scale centralized server could be constructed to ensure all registered contracts are pinged at the right time. This server could have a smart contract serving as its proxy on the blockchain to let other smart contract register themselves with a fee and also to ping back client smart contracts.

Private blockchains do not encounter this limitation as they can simply ping every resident smart contract at a regular interval. Participants could simply be charged contract management fees, perhaps billed to dedicated accounts for the execution of all contracts they own. A similar mechanism is planned for a future release of Ethereum as well, permitting the use of gas-free transactions as long as miners can be compensated by the ensuing contract method

execution. This could avoid the challenges of gas cost estimation and billing currently faced by scheduling services for public blockchains.

The Flow

We will now run a demo of sorts by going through the full flow of a toy contract. We first create a price feed which amounts to booting up our financial system. In this demo, the buyer and seller correspond to the same account.

Each counterparty must now create and fund a trading account. First, the buyer of the call spread creates their account and funds it with ether.

Then, the seller does the same.

The buyer now creates a call spread, which in turn initializes two call options as its legs. The buyer then sends the address of the call spread to the seller, who loads the contract details in their browser.

The seller must verify the economics of the deal before it is considered active.

We now ping the price feed to perturb the value of USD/ETH. The perturbation should guarantee that some margin will be collected.

We can see the effect of the ping on the price feed:

We now ping the call spread itself to trigger margin calculation. Indeed, 13469 wei is collected as margin from the seller.

Upon maturity, the buyer tries attempts to exercise his option. The exercise in this case succeeds and the contract is marked complete. Margin is returned and all required cash flows are executed.

Here is a table that shows how all required cash flows are sent back and forth at the end in a granular manner.

Conclusion

Once again, we are pleased to report that financial instruments can indeed be encoded on the Ethereum blockchain. We hope to extend this work by creating a library of standard smart financial instruments and adding the ability to compose them to create more complex bespoke instruments. With the advent of token-based DAOs, such instruments might be of value in the blockchain ecosystem by allowing people to hedge and diversify their risk. Even with the limitations we described, it is possible to create several useful classes of financial smart contracts which increase liquidity on the blockchain by allowing participants to take more sophisticated positions on tokens, sub-currencies and DAOs. These contracts could also be plugged into identity and reputation systems and also possibly use on-blockchain calculators and schedulers.

We look forward to your questions and comments about both our article and the general direction of our work.

Kapil and Vishakh