Mithrl: The Future of Distributed Smart Contracts

Abstract

Much has been said of smart contracts and blockchains, and the potential they have to revolutionise industries by eliminating transaction friction, automating business flows, and increasing intra-industry cooperation. While these goals make them attractive for enterprises, few have incorporated them into core business applications because of the risks associated with a self-enforcing, self-executing, and immutable system. We propose a solution that is able to achieve all these goals and also mitigates the risks through the use of high assurance software development paradigms, and a newly designed hybrid ledger called the metachain.

1 Introduction

The idea of a smart contract has been around since the early 90s when they were first proposed by Nick Szabo. However, it was the successful implementation of decentralised transaction ledgers in projects such as Bitcoin, and later Ethereum, that reignited interest in its potential to reshape the way we handle transactions. In an era of unprecedented global communication and trade, it has become necessary for contracts of any kind to keep up with the speed, and complexity of modern business. Traditional contracts have proven to be too costly, subjective and inefficient for a world economy that is growing increasingly reliant on digital transactions. Just as how stamps, seals, and physical signatures have been replaced by digital signatures, certificates, and public key cryptography; we envision a world where traditional contracts are supplemented, or even supplanted by contracts written in code and stored on a global ledger. This global ledger forms the substrate onto which asset management systems are built upon, with these contracts providing the mechanisms by which these assets are transferred. In this future, transactions between parties are frictionless, seamless, and instantaneous.

This paper pursues this notion by introducing Mithrl, a platform that we believe represents the future of smart contracts and distributed ledgers.

The first section will be focused on the Mithrl scripting language including the motivation behind our design choices. The next section introduces the Metachain, our unique blockchain, that forms the backbone for our platform.

2 Mithrl Scripting Language

One of the main issues we identified in current smart contracting languages is that it is extremely difficult to assert the correct execution of a smart contract. That is to say, given a smart contract, is it possible to prove that the contract will only execute in the way it was intended?

With the complexities of modern business it is difficult for developers to maintain complex state transitions and edge cases unassisted. In the wake of the DAO hack, which occurred despite the technical prowess of its developers and an extensive security audit, it became clear that smart contracting languages needed to help developers reason about their contract's execution.

Mithrl smart contracts are written in Haskell: a strong, statically typed, purely functional language. It was chosen for its rich type system, concurrency performance, domain language support, and its ecosystem of software verification tools. Many languages approach software verification by conducting extensive unit and edge case testing - a process that is limited by the tester's perseverance and creativitity. Purely functional languages like Haskell are able to leverage the Curry-Howard isomorphism - a mapping between the type system and formal logic that implies that type signatures for functions in Haskell act as theorems or logical predicates that the function themselves are designed to prove. By pairing this statically analysable type system with the referential transparency of a pure system, the Haskell compiler is able to make optimisations and move many run time errors to compile time - leading to the " if it compiles it works" statement often made by Haskell developers.

In addition to the full suite of Haskell packagaes, Mithrl also provides a set of embedded domain specific languages (EDSLs) to make writing smart contracts easier, quicker and more secure. Firstly, our construction of EDSLs lend themselves to an imperative style of programming which makes the language accessible to programmers who may be unfamiliar with functional programming designs. Secondly, the EDSLs make development quicker by providing useful functionality that is common across contracts such as basic cryptography and contract storage, thereby reducing the boilerplate that a developer has to write. Finally, each of our EDSLs form a denotation or domain that can be composed with each other form progressively larger programs. A smart contract in Mithrl can then be thought of as a collection of these larger programs built out of sub-programs with the EDSLs at its foundation. This approach is referred to as denotational semantics and results in smart contracts that remain easy to reason about, and importantly helps to ensure that invalid states remain unrepresentable. An important feature for smart contracts that govern mission-critical applications.

3 Metachain

The Metachain is our proposal for a unique hybrid blockchain that looks to provide the benefits of both permissioned and public blockchains. While the following section will provide a high level overview of the Metachain, it will hopefully help to elucidate the benefits of such a system. An understanding of hash pointers and Merkle structures will be helpful in grokking the concepts.

Data is stored in blockchains through the use of Merkle-ized data structures that "hang" off the main blockchain and is referred to by a hash pointer (Merkle root) stored in the block header. The data in these structures are stored outside of the blockchain yet gain the immutability of the blockchain because of the transitivity of hash pointer with respect to cryptographic validation. That is to say, if A has a hash pointer referencing B, and B has a hash pointer referencing C, then A reinforces that both B and C remain unchanged. B is trivially unchangeable because it would result in the hash pointer from A referencing a now non-existent B, and similarly, a change in C would result in a change in the hash pointer from B that cascades back up to A. Therefore, it can be shown that as long as A is kept unchanged, every descendent of it will have to remain unchanged as well. In order to keep A, the tip of a hash chain, from being changed, blockchains such as Bitcoin

employ various cryptoeconomical incentives.

The Metachain is constructed by threading multiple data structures together, via hash pointers, through a central structure. The data structures that "hang off" the Metachain can themselves also be blockchains or any piece of data that can be reduced to a single hash. An important aspect of the construction of the Metachain is the fact that hash pointers between the Metachain and other blockchains are bidirectionally connected. This means that the state of the external connected blockchains and of the Metachain can be reduced and stored within opposing blockheaders.

3.1 Properties

3.1.1 Cumulative Security

The security of a blockchain is a function of the total hash rate securing the blockchain and the distribution of this hash rate across all the miners. Mining puzzles play an important role in the security of public blockchains - helping to provide a value anchor for cryptographic assets in terms of real-world expenditure. Bitcoin uses SHA256 as its hashing function in its Proof-of-Work (PoW) puzzle, however, there are many other variations of hashing function such as Ethhash and Scrypt as well as different puzzles entirely such as Proof-of-Stake (PoS) and Proof-of-Burn (PoB). For each hashing function there are highly specialized hardware (ASICs) that are used to solve them at increasingly faster rates. ASICs are expensive and can only be used to solved a specific mining puzzle, therefore an attacker wanting to attack multiple blockchains must purchase large amounts of different ASICs to be successful.

A beneficial property of the Metachain is that the threading of other blockchains extends the security of the central data structure. If we stipulate that some of the blockchains that are attached to it have orthogonal mining puzzles, then an attacker wanting to produce a forgery about the state of the Metachain would need to attack all the connected blockchains simultaneously. Thus, the Metachain gains the cumulative security of each of the blockchain it is bidirectionally pegged into.

3.1.2 Interoperability

Blockchain interoperability opens up the possibility of assets, transactions, and information being validated or transferred from chain to chain. In the context of enterprises, the idea of industries using different asset management systems to govern their specific use cases is not new. We envision that in the world of smart contracts and blockchains that this will continue to be true; different industries will construct their own permissioned chain and even within industries, there may exist smaller subsections of this chain. With Mithrl, entangled permissioned chains are able to verify events similar to the way an SPV client verifies events with the main difference being that the proof can instead span across blockchains by navigating through the central Metachain.

3.2 Block Generation

Mithrl blocks are created and chained together using a method introduced by Dr Bob McElrath at a Scaling Bitcoin conference in 2015, which he termed "Braiding the Blockchain". While the details of its implementation are beyond the scope of this paper they can be found in various videos and his Github profile, I will instead provide a high level overview of the general concepts.

Firstly, to motivate the reasons behind our design decision it is useful to understand what role block times play in the running of a blockchain. Absolute time is difficult to get correct in

distributed systems: firewalls,bandwith,the speed of light, and the curvature of the earth makes it very difficult for precise times to be agreed upon by differing parties. One of the roles of blocks in a blockchain is to be able to "chunk" time into discrete blocks where each chunk represents a snapshot of the state of the world(blockchain) and the chains (hash pointers) represent the chronological ordering of these snapshots.

It is tempting for permissioned blockchains to replicate Bitcoin's 10 minute block time considering how well it has served the cryptocurrency to date. However, the block size debate, mempool backlog crisis, and the rapidly increasing transaction fees among others is evidence of the limitation of Bitcoin blocks in handling a large volume of transactions. On the other end of the block time spectrum there sit blockchains that use much shorter blocktimes, such as Ethereum that use a 15 second block time. While this helps with processing incoming transactions, the reduced blocktime also makes the effects of network latency more pronounced - increasing the probability of conflicting block solutions being broadcasted (orphan rate).

In the context of permissioned blockchains, faster blocktimes can result in sequences of (near) empty blocks being produced if the supply of transactions is low. This has the effect of increasing the complexity of transaction validation, assuming an SPV-style proof, as the size of the proof scales linearly with the number of blocks.

3.2.1 Metachain Braiding

Our approach is based on McElrath's idea of relaxing the contraints on the blockchain being a linked list of blocks, so that it instead forms a restricted directed acyclic graph (DAG). In this DAG, blockchain blocks are allowed to have multiple parents as well as siblings, with the only restriction being that blocks cannot refer to their grandparents. Siblings are allowed to have duplicate transactions, double-spends are still disallowed, because they will eventually be reconciled by a future descendant. When all siblings have been resolved to some future descendant, the section of the DAG is referred to as a cohort. A cohort represents a total ordering of blocks such that they are the ancestors of the next cohorts and a descendant of all previous cohorts.

Simply, a braided blockchain can be thought of as a blockchain of cohorts where each cohort contains a DAG of blocks with a block defined in the traditional blockchain sense. The advantage of this setup is that there exists a zero parameter targeting algorithm that can be used to find the optimal cohort time based on the transaction demand. Furthermore, in traditional blockchains only one node may write to the blockchain at any time, therefore all nodes must be able to understand & enforce the semantics of every possible operation in the blockchain - if not they may incorrectly classify transactions as being invalid. We realise that if Mithrl is to support an ecosystem of blockchains specialised in purpose it would be intractable to force all participating nodes to update their rules when new blockchains are connected to the Metachain. Braiding allows the Metachain to handle multiple concurrent writers which means that participating nodes need only concern themselves with transactions that they think are valid and build blocks containing those transactions, allowing continuous addition of new blockchains with minimal impact to already participating nodes.

3.2.2 Consensus

The requirement for consensus in permissioned blockchains is still up for debate - if all nodes correspond to some known real world identity then any detected malicious behaviour can be dealt with simply through real world repercussions. Consensus mechanisms must operate differently in

the context of permissioned blockchains as the cryptoeconomics that incetivise miners in public blockchains now have a negligible effect on a nodes decision to be Byzantine. At Mithrl we are looking to implement an on-chain random beacon , as proposed by String Labs , to enable leader - free block generation. A random beacon is a public source of verifiable, cryptographic randomness whose next value is unpredictable.The periodic random beacon "heartbeat" is used to select a group of blockmakers who construct and validate the transactions for the next block in the chain, ensuring fair transaction selection. Within the smaller blockmaker group, Byzantine Fault Tolerant (BFT) consensus mechanisms using the heartbeat can be used to ensure that attacks do not prevent blocks from being generated.

4 Mithrl for Enterprise

Permissioned blockchains are preferred by enterprises because of the confidentiality and control that they afford. However, they suffer from the problem that only having a small number of participants (relative to public ledgers) leaves them open to attacks by internally colluding or malicious users. Where public blockchains are trustless, permissioned blockchains are more trustful - though not completely, where the subtle difference between the two are the size of the participants motivations set, and the probability of failure given a set of motivations. While permissioned blockchains excel at the latter, the former is limited by the size of active participants.

Mithrl's Metachain enables permissioned blockchains to leverage the greater security of public blockchains without compromising their transactions confidentiality or data integrity. By embedding the hash of their state into the Metachain, enterprises can later cryptographically verify that the state of their blockchain has not been modified. Because the hash pointers between the Metachain and other blockchains are bidirectional, enterprises are able to check the embedded state of the Metachain on their own blockchain to ensure that the Metachain itself has also not been compromised before verifying their state stored on it. Additionally, anyone is able to verify the state of the external blockchains stored on the Metachain and vice-versa.

This system of checks and balances between public blockchains, permissioned blockchains and the Metachain ensure that the information stored in the Metachain is tamper resistant, despite not being fully decentralised, even from ourselves. By exploiting the disparate motivations of the entities, we are able to provide an unparalleled degree of auditability and verifiability to all blockchains connected to the Mithrl's Metachain.

At the application level, the Mithrl scripting language means that enterprises can have greater confidence about the execution and correctness of their smart contracts. The security and immutability of a blockchain means little if the contracts that exist on them are exploitable. The immutability of blockchains means that patching bugs in contracts, especially those that contain values, can be difficult since they cannot be taken down and redeployed. The self execution of smart contracts makes them efficient and automatable, however, it also allows vulnerabilities to be wholly exploited over a short period of time - in 6 hours the DAO hacker was able to siphon USD$50 million. Ethereum's push to add formal verification to Solidity is further evidence of the importance of provably correct smart contracts. Mithrl produces high assurance smart contracts - that is contracts which execute as intended and only as intended.

5 Conclusion

We took the lessons learnt from Bitcoin and Ethereum, and best-in-class software verification techniques to come with a new approach to smart contracts and blockchains. This approach, grounded in mathematics and equational reasoning, provides a solution that is able to mitigate many of the risks and potential vulnerabilities of existing blockchain and smart contract platforms.