WHİTE PAPER

BITCHAIN (BTC)

A Decentralized Oracle Network

13 Ağustos 2023

Abstract

Smart contracts are poised to revolutionize many industries by replacing the need for both traditional legal agreements and centrally automated digital agreements. Both performance verification and execution rely on manual actions from one of the contracting parties, or an automated system that programmatically retrieves and updates relevant changes. Unfortunately, because of their underlying consensus protocols, the blockchains on which smart contracts run cannot support native communication with external systems.Today, the solution to this problem is to introduce a new functionality, called an oracle, that provides connectivity to the outside world.

Existing oracles are centralized services. Any smart contract using such services has a single point of failure, making it no more secure than a traditional, centrally run digital agreement.In this paper we present Bitchain, a decentralized oracle network. We describe the on-chain components that Bitchain provides for contracts to gain external connectivity, and the software powering the nodes of the network. We present both a simple on-chain contract data aggregation system, and a more efficient off-chain consensus mechanism. We also describe supporting reputation and security monitoring services for Bitchain that help users make informed provider selections and achieve robust service even under aggressively adversarial conditions. Finally, we characterize the properties of an ideal oracle as guidance for our security strategy, and lay out possible future improvements,including richly featured oracle programming, data-source infrastructure modifications, and confidential smart-contract execution

Contents

1 Introduction

Smart contracts are applications that execute on decentralized infrastructure, such as a blockchain. They are tamperproof, in the sense that no party (even their creator) can alter their code or interfere with their execution. Historically, contracts embodied in code have run in a centralized manner that leaves them subject to alteration, termination, and even deletion by a privileged party

In contrast, smart contracts' execution guarantees, which bind all parties to an agreement as written,create a new and powerful type of trust relationship that does not rely on trust in any one party. Because they are self-verifying and self-executing (i.e., tamperproof as explained above),

smart contracts thus offer a superior vehicle for realizing and administering digital agreements.
The powerful new trust model that smart contracts embody, though, introduces a new technical challenge: connectivity. The vast majority of interesting smart contract applications rely on data about the real world that comes from key resources,specifically data feeds and APIs, that are external to the blockchain. Because of the mechanics of the consensus mechanisms underpinning blockchains, a blockchain cannot directly fetch such critical data.We propose a solution to the smart contract connectivity problem in the form ofBitchain, a secure oracle network. What differentiates Bitchain solutions is its ability to operate as a fully decentralized network. This decentralized approach limits the trust in any single party, enabling the tamperproof quality valued in smart contracts to be extended to the end-to-end operation between smart contracts and the APIs they rely on. Making smart contracts externally aware, meaning capable of interacting with off-chain resources, is necessary if they are going to replace the digital agreements in use today.Today, the lion's share of traditional contractual agreements that have been digitally automated use external data to prove contractual performance, and require data outputs to be pushed to external systems. When smart contracts replace these older contractual mechanisms, they will require high-assurance versions of the same types smart contracts and their data requirements include:
• Securities smart contracts such as bonds, interest rate derivatives, and many others will require access to APIs reporting market prices and market reference data, e.g. interest rates.The main use of smart contracts in Binance Smart Chaintoday is management of tokens, which are a common functionality in most smart contract networks. We believe that the current focus on tokens to the exclusion of many other possible applications is due to a lack of adequate oracle services, a situation Bitchain specifically aims to remedy.

• Insurance smart contracts will need data feeds about IoT data related to the insurable event in question, e.g.: was the warehouse's magnetic door locked at the time of breach, was the company's firewall online, or did the flight you had insurance for arrive on time.

• Trade finance smart contracts will need GPS data about shipments, data from supply chain ERP systems, and customs data about the goods being shipped in order to confirm fulfillment of contractual obligations.Another problem common to these examples is the inability for smart contracts to output data into off-chain systems. Such output often takes the form of a payment
message routed to traditional centralized infrastructure in which users already have accounts, e.g., for bank payments, PayPal, and other payment networks. Bitchain's ability to securely push data to APIs and various legacy systems on behalf of a smart contract permits the creation of externally-aware tamperproof contracts.

Whitepaper roadmap

In this whitepaper*, we review the Bitchain architecture (Section 2). We then explain how we define security for oracles (Section 3). We describe the Bitchain approach to decentralization / distribution of oracles and data sources (Section 4),and follow with a discussion of the four security services proposed by Bitchain,as well as the role played by BTC tokens (Section 5). We then describe a proposed long-term development strategy, which includes better confidentiality protections, the use of trusted hardware, infrastructure changes, and general oracle programmability (Section 6). We briefly review alternative oracle designs (Section 7), and conclude with a short discussion of the design principles and philosophy guiding Bitchain
development (Section 8).

2 Architectural Overview

Bitchain's core functional objective is to bridge two environments: on-chain and offchain. We describe the architecture of each Bitchain component below. Bitchain will initially be built on Ethereum [16], [35], but we intend for it to support all leading smart contract networks for both off-chain and cross-chain interactions. In both its on and off-chain versions, Bitchain has been designed with modularity in mind.Every piece of the Bitchain system is upgradable, so that different components can be replaced as better techniques and competing implementations arise.

2.1 On-Chain Architecture

As an oracle service, Bitchain nodes return replies to data requests or queries made by or on behalf of a user contract, which we refer to as requesting contracts and denote by USER-SC. Bitchain's on-chain interface to requesting contracts is itself an on-chain contract that we denote by BITCHAIN-SC.Behind BITCHAİN-SC, Bitchain has an on-chain component consisting of three main contracts: a reputation contract, an order-matching contract, and an aggregating contract. The reputation contract keeps track of oracle-service-provider performance metrics. The order-matching smart contract takes a proposed service level agreement,logs the SLA parameters, and collects bids from oracle providers. It then selects bids using the reputation contract and finalizes the oracle SLA. The aggregating contract
collects the oracle providers' responses and calculates the final collective result of the Bitchain query. It also feeds oracle provider metrics back into the reputation contract. Bitchain contracts are designed in a modular manner, allowing for them to be configured or replaced by users as needed. The on-chain work flow has three
steps: 1) oracle selection, 2) data reporting, 3) result aggregation.

Oracle Selection An oracle services purchaser specifies requirements that make up a service level agreement
(SLA) proposal. The SLA proposal includes details such asquery parameters and the number of oracles needed by the purchaser. Additionally,the purchaser specifies the reputation and aggregating contracts to be used for the rest of the agreement.
Using the reputation maintained on-chain, along with a more robust set of data gathered from logs of past contracts, purchasers can manually sort, filter, and select oracles via off-chain listing services. Our intention is for Bitchain to maintain one such listing service, collecting all Bitchain-related logs and verifying the binaries of listed oracle contracts. We further detail the listing service and reputation systems in Section 5. The data used to generate listings will be pulled from the blockchain,allowing for alternative oracle-listing services to be built. Purchasers will submit SLA proposals to oracles off-chain, and come to agreement before finalizing the SLA on-chain.Manual matching is not possible for all situations. For example, a contract may need to request oracle services dynamically in response to its load. Automated solutions solve this problem and enhance usability. For these reasons, automated oracle matching is also being proposed by Bitchain through the use of order-matching contracts.Once the purchaser has specified their SLA proposal, instead of contacting the oracles directly, they will submit the SLA to an order-matching contract. The submission of the proposal to the order-matching contract triggers a log that oracle providers can

monitor and filter based on their capabilities and service objectives. Bitchain nodes then choose whether to bid on the proposal or not, with the contract only accepting bids from nodes that meet the SLA's requirements. When an oracle service provider bids on a contract, they commit to it, specifically by attaching the penalty amount that would be lost due to their misbehavior, as defined in the SLA.Bids are accepted for the entirety of the bidding window. Once the SLA has received enough qualified bids and the bidding window has ended, the requested number of oracles is selected from the pool of bids. Penalty payments that were offered during the bidding process are returned to oracles who were not selected, and a finalized SLA record is created. When the finalized SLA is recorded it triggers a log notifying the selected oracles. The oracles then perform the assignment detailed
by the SLA.

Data Reporting Once the new oracle record has been created, the off-chain oracles execute the agreement and report back on-chain. For more detail about off-chain interactions, see Sections 2.2 and 4.

Result Aggregation Once the oracles have revealed their results to the oracle contract, their results will be fed to the aggregating contract. The

aggregating contract tallies the collective results and calculates a weighted answer. The validity of each oracle response is then reported to the reputation contract. Finally, the weighted answer is returned to the specified contract function in USER-SC.Detecting outlying or incorrect values is a problem that is specific to each type of data feed and application. For instance, detecting and rejecting outlying answers before averaging may be necessary for numeric data but not boolean. For this reason,there will not be a specific aggregating contract, but a configurable contract address which is specified by the purchaser. Bitchain will include a standard set of aggregating contracts, but customized contracts may also be specified, provided they conform to the standard calculation interface.

2.2 Off-Chain Architecture

Off-chain, Bitchain initially consists of a network of oracle nodes connected to the Ethereum network, and we intend for it to support all leading smart contract networks. These nodes independently harvest responses to off-chain requests. As we explain below, their individual responses are aggregated via one of several possible
consensus mechanisms into a global response that is returned to a requesting contract USER-SC. The Bitchain nodes are powered by the standard open source core implementation which handles standard blockchain interactions, scheduling, and connecting with common external resources. Node operators may choose to add software

extensions, known as external adapters, that allow the operators to offer additional specialized off-chain services. Bitchain nodes have already been deployed alongside both public blockchains and private networks in enterprise settings; enabling the
nodes to run in a decentralized manner is the motivation for the Bitchain network.

Bitchain Core. The core node software is responsible for interfacing with the blockchain, scheduling, and balancing work across its various external services. Work
done by Bitchain nodes is formatted as assignments. Each assignment is a set of smaller job specifcations, known as subtasks, which are processed as a pipeline. Each subtask has a specific operation it performs, before passing its result onto the next
subtask, and ultimately reaching a final result. Bitchain’s node software comes with a few subtasks built in, including HTTP requests, JSON parsing, and conversion to various blockchain formats.

External Adapters. Beyond the built-in subtask types, custom subtasks can be defined by creating adapters. Adapters are external services with a minimal REST API. By modeling adapters in a service-oriented manner, programs in any programming language can be easily implemented simply by adding a small intermediate API in front of the program. Similarly,

interacting with complicated multi-step APIs can be simplified to individual subtasks with parameters.

Subtask Schemas. We anticipate that many adapters will be open sourced, so that services can be audited and run by various community members. With many different types of adapters being developed by many different developers, ensuring compatibility between adapters is essential.Bitchain currently operates with a schema system based on JSON Schema [36],to specify what inputs each adapter needs and how they should be formatted. Similarly, adapters specify an output schema to describe the format of each subtask'soutput.

3 Oracle Security

In order to explain Bitchain's security architecture, we must first explain why security is important—and what it means.
Why must oracles be secure? Returning to our simple examples in Section 1,if a smart contract security gets a false data feed, it may payout the incorrect party,if smart contract insurance data feeds can be tampered with by the insured party

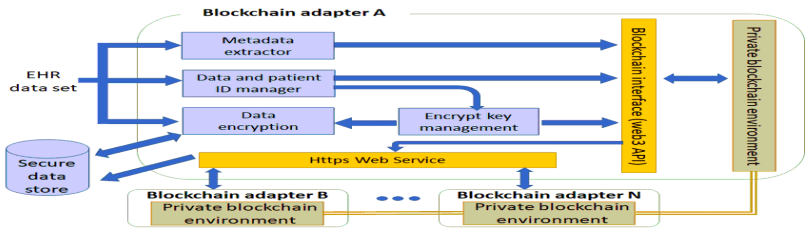


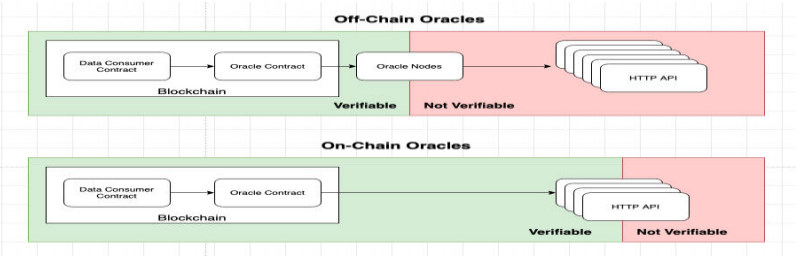
Figure 1: Bitchain workflow: **1) USER-SC** makes an on-chain request; **2) BITCHAIN-SC** logs an event for the oracles; 3) Bitchain core picks up the event and routes the assignment to an adapter**; 4)** Bitchain adapter performs a request to an external API; **5)** Bitchain adapter processes the response and passes it back to the core; **6)** Bitchain core reports the data to **BITCHAIN-SC**; **7) BITCHAIN-SC** aggregates responses and passes them back as a single response to USER-SC.

there may be insurance fraud, and if GPS data given to a trade finance contract can be modified after it leaves the data provider, payment can be released for goods that haven't arrived.

More generally, a well-functioning blockchain, with its ledger or bulletin-board abstraction, offers very strong security properties. Users rely on the blockchain as a functionality that correctly validates transactions and prevents data from being altered. They treat it in effect

like a trusted third party (a concept we discuss at length below). A supporting oracle service must offer a level of security commensurate with that of the blockchain it supports. An oracle too must therefore serve users as
an effective trusted third party, providing correct and timely responses with very high probability. The security of any system is only as strong as its weakest BTC, so a highly trustworthy oracle is required to preserve the trustworthiness of a wellengineered blockchain.

Defining oracle security: An ideal view. In order to reason about oracle security, we must first define it. An instructive, principled way to reason about oracle security stems from the following thought experiment. Imagine that a trusted third party (TTP)—an ideal entity or functionality that always carries out instructions
faithfully to the letter—were tasked with running an oracle. We'll denote this oracle by ORACLE (using all caps in general to denote an entity fully trusted by users), and suppose that the TTP obtains data from a perfectly trustworthy data source Src.
Given this magical service ORACLE, what instructions would we ask it to carry out? To achieve the property of integrity, also referred to as the authenticity property [24], we would simply ask that ORACLE perform the following steps:



One notable application of Apsis is On-chain Oracles which provides end-to-end verifiability to access external data.

There are two critical differences between off-chain Oracles and on-chain Oracles:

Oracle nodes are no longer required in on-chain Oracles since on-chain Oracles can access HTTP API directly. This might imply significant operational cost since no middleman tax is paid to Oracle nodes;
Data from HTTP API to Oracle contract is completely verifiable in on-chain Oracles. This means the data consumers only need to trust the HTTP API providers, in contrast to off-chain Oracles where data consumers need to trust the Oracle nodes to operate honestly.
The presence of on-chain Oracle does not preclude the existence of third-party Oracle providers. Existing Oracle providers can build their Oracle solutions on Apsis which read data from multiple HTTP APIs and aggregate data points. There are several benefits to build on-chain Oracle solutions on Apsis:

On-chain Oracles on Apsis provide end-to-end data verifiability. Oracle data consumers don't have to trust the Oracle providers as the whole data

generation process is visible on-chain;Oracle providers don’t have to manage their Oracle node networks and overpay them with their own tokens. Instead, Oracle providers can better utilize their token to bootstrap the ecosystem;On-chain Oracles can support both realtime and cached data access. Oracle data consumers can pay more to trigger a new round of data update in order to read the latest data, or pay a small portion to access the data cached in the last round update.On-chain Oracles can also be accessed by applications on other blockchains via cross-chain bridge. For example, lending protocols on Ethereum can utilize the Apsis-Ethereum bridge to read the latest price from the on-chain Oracles on Apsis. Apsis chain can become an Oracle hub in the incoming multi-chain era.

Apsis can also empower hybrid blockchain applications to take full advantage of both Web 2.0 and 3.0. For example, an electronics retailer can build a sales DApp to sell their televisions and use their existing systems to provide price and inventory information. The whole purchase can be implemented in a single transaction which is not feasible on any existing blockchains.