XY0 2.0 Platform: The Sovereign Internet Platform based on the XYO Protocol

Arie Trouw *, Joel Carter †, Matt Jones ‡

January 2024

Abstract

The XYO 2.0 Platform is a system implementation of the XYO Protocol as defined in the XYO Protocol Whitepaper published in January 2018. It focuses on providing a solution that achieves high performance without sacrificing the sovereignty, provenance, and permanence that is the goal setout by the whitepaper. This XYO 2.0 Platform also expands the usage of the core concepts defined in the White Paper to be useful in a much broader set of use-cases, specifically not limiting its use to location. The implementation set forth in this Yellow Paper adds additional protocol definitions to provide guidelines through which future components and alternative implementations can be created while maintaining the ability for them to work together to form a singular XYO Network.

1 Introduction

During the last decade, Web 3 development has been primarily focused on expanding the use of shared ledgers to create decentralized systems. Even though there have been great strides on this front, the very core of this effort is flawed in two ways.

First, shared ledgers moves the control of a system from being fully centralized (effectively a kingdom model) towards a decentralized solution that is based on majority rule and finality (effectively a democracy or republic). Like with all governance systems, the natural evolution of these systems have pulled back from maximizing decentralization towards more centralized concepts for practical, regulatory, or other, potentially sinister, reasons. Even if this pull-back did not occur, the ceiling of shared ledger decentralization is that of majority rule and not true sovereignty.

Second, the performance of shared ledger technology has been and will always be substantially (orders of magnitude) slower and more costly than their centralized equivalents. By its very definition, a shared ledger must either has massive redundancy of data and validation, or lean on trusted systems to improve performance.

*XYO Network, arie.trouw@xyo.network
†XYO Network, joel.carter@xyo.network
‡XYO Network, matt.jones@xyo.network

This implementation of the XYO 2.0 Platform combined with the core concepts of the XYO Protocol strives to provide full decentralization with nodes that are 100 percent sovereign while using cryptographic technologies and concepts to deliver a trustless network that has performance at scale comparable or better than the performance of an equivalent Web 2 system and orders of magnitude better than equivalent Web 3 systems. This combination not only delivers on the goals of Web 3 visionaries, but also delivers on the goals set forth by the original founders of the internet. The current Web 2 implementation of the internet is completely devoid of sovereignty, provenance, and permanence and we must reverse that trend by delivering a solution that is the foundation for the Sovereign Internet by combining the core tenants of Web 2 and Web 3 combined with the concepts of the XYO Protocol as set out in the original XYO White Paper.

2 Practical Decisions

In producing the XYO 2.0 Platform, various practical decisions have been made to facilitate interoperability and reduce ambiguity in the protocol.

2.1 Programming Language

The initial version of the XYO 2.0 Platform has been developed using TypeScript. We chose this due to the expansive tools that are available for developing with TypeScript and the compatibility that Javascript (the output of compiling TypeScript) allows for. As a result, this implementation can be used on browsers and with the NodeJS runtime, both on desktop and on mobile devices. The sacrifice of this decision is that running the technology stack on IoT devices, especially battery powered devices will be negatively impacted. This can be addressed by creating limited native implementations of the Platform for those devices.

WebAssembly is used for various high performance cryptographic algorithms since WebAssembly can be seamlessly interacted with from Javascript. Over time, it is possible that more TypeScript based code could be replaced with WebAssembly, but that will be done with care since there are costs of doing this when it comes to understandability of code and debugging.

2.2 Payload Encoding

Payloads are encoded as JSON objects for easy use in TypeScript/Javascript and universal use in other languages. This decision does however required various considerations which are outlined in other sections below. Meta fields in the data object are fields that are not included in the hash. All meta fields have their names prefixed with an underscore character.

2.3 BoundWitness Encoding

A BoundWitness object is a specialized payload that conforms to all the rules of a payload when it comes to encoding and hashing along with these additional requirements.

1. The list of addresses that are partaking in the binding are listed in an array under the field name 'addresses'. These conform to the address encoding rules and have a fixed order that is required for matching to the signatures provided by them.

2. The list of signatures are listed under the field name of 'signatures' prefixed with an underscore character. This causes this field to be considered meta data and is excluded from the hash. The order of the signatures must match the order of the address that are listed in the 'addresses' field and there must be a signature for each address for the object to be considered valid.

2.4 Hashing

For consistent hashing a specific procedure must be followed when generating the hash.

1. Sort the object by field name. This is a recursive operation, where each sub object also has its fields sorted by name. 2. Remove meta fields by removing all fields that are prefixed with an underscore character. 3. Stringify the JSON object without whitespace using utf-8 encoding. 4. Generate a SHA256 hash on the resulting string.

2.5 Addresses

2.6 Signing

2.7 Big Numbers

3 Module System

3.1 Manifests

3.2 Addressing

3.3 Discovery

3.4 Eventing

4 Component System

We chose React as the framework to create user-interface components for the XYO 2.0 Platform. This does not preclude using other frameworks and in the cases where native user-interfaces are required, alternative frameworks will be required. In all these cases, the paradigm which we embrace in our React implementation should be followed.

4.1 Renderers

A Renderer is a component that takes a working set of payloads and renders them. In many cases, the renderer is so simple that it only has the ability to render a single payload of a specific type. The most basic of renderers simply renders a payload's raw data, which usually is a JSON object. In most cases, a dApp strives to have much friendlier renderers than the basic renderer. Currently, the only way to make custom renderers is to create a new React component usually based on an existing renderer such as the basic raw renderer. We have a future goal to produce a codeless system to producing renderers.

4.2 Hooks

Hooks are a specific system used in React, but as a paradigm, exists in most user-interface frameworks. The purpose of hooks to to separate the acquisition/manipulation of data from the rendering of that data. In most cases, the provided generic hooks that provide access to XYO modules are sufficient to gather the data in a user interface for a dApp.

5 Codeless Development

A primary goal of the XYO 2.0 Platform is to allow users to fully customize nodes through Codeless Development using the module system with manifests, configurations, and parameters. The only reason to create custom modules using TypeScript directly should be for performance reasons, not because what is being created is not possible via the Codeless Development paradigm supported by the platform.

6 Security

Security concerns in the XYO Platform follows that of the XYO Protocol exclusively using cryptographic mechanisms to provide security. This includes hashes, addresses, and cryptographic signing of data. As for individual security on specific modules, it is left up to that module to mange access either through lists of allowed/disallowed addresses, crypto-economic incentives, or a combination of the two.

7 Privacy

Privacy itself is not addressed in the core XYO Protocol, however, the ability to optionally keep payloads private is accommodated by the protocol. There are two primary paradigms that we utilize in the XYO 2.0 Platform.

7.1 Just-In-Time Privacy

Just-In-time Privacy (JITP) for XYO is the paradigm where a node shares hashes of payloads without sharing the actual payloads. This allows sovereign Bound Witnesses to be created, establishing provenance and maintaining immutable permanence, while maintaining privacy of the originating payload. JITP can be used to create sovereign games, where multiple parties can lock in moves without exposing what those moves are, and then only exposing those moves when declaring victory.

7.2 Subnet Privacy

Subnet Privacy is the ability to run a private XYO Subnet. This is accomplished by running one or more nodes that are networked together, allowing each other to have access that is not available to nodes outside of that network. This is similar to a traditional intranet. The security concerns are also very similar to intranets in that there will be nodes that have access to both the private XYO Subnet and to external networks such as other private subnets or the public XYO Network. Since those nodes are on multiple networks at the same time, it is possible for them to intentionally or accidentally leak private data from the private subnet to the external network, so great care must be taken when allowing nodes to connect to a private subnet.

8 Acknowledgements

This white paper is the product of an inspiring team effort that was made possible through the belief in our vision from the following individuals:

References

[1] Trouw, Arie; Levin, Markus; Sheper, Scott *XYO Protocol White Paper*. XYO Website. January 2018