HELIX2

Link Service and Protocol

Avneet Singh
Interplanetary Company UG
avneet.singh@tutanota.com

ABSTRACT

Double Helix or Helix2 is a next-generation link service and account abstraction protocol on Ethereum, originally designed as a natural successor to generic name services. Helix2 protocol allows names to link to each other in several useful configurations on Ethereum blockchain. Helix2 infrastructure codifies interactions between names, categorises those interactions, assigns them rules and labels, and in some cases, validates those interactions with on-chain records. Due to its unique design crafted to leverage interactions among names, Helix2 enables names to form organised on-chain structures and graphs. Wallets integrating Helix2 can leverage its properties to provide users with a smart contract interface to transact and interact with the Ethereum blockchain; this arguably makes Helix2 an account abstraction infrastructure.

INTRODUCTION

Most blockchains have developed their own versions of naming systems which allow representing addresses with human-readable names. On Ethereum, Ethereum Name Service (ENS) is the first and most notable example, while similar services later became available on Tezos and Solana in the form of Tezos Domains and Solana Name Service by Bonfida. By construction, a name service assigns names to nodes in a network. In a classic web2 world, Domain Name Service (DNS) fulfils this requirement. Crypto-native name services are similar to DNS in the sense that they enable assigning names to addresses similar to how DNS assigns human-readable names to Internet Protocols (IP). There are however clear added benefits to crypto-native naming architecture over DNS since crypto-native services often double as a decent identity framework in their respective blockchain ecosystems. It goes without saying that the immutability and decentralisation properties of typical crypto-native systems add to their desirability owing to their censor-resistant and unruggable nature.

Helix2 is designed as a next-generation successor of these name services. While the set of nodes form a canonical and natural choice for labeling in any distributed system (e.g. addresses on any blockchain), the observation nonetheless is that most nodes do not interact with each other. For instance, there are about 220 million Ethereum addresses whereas an average address will likely interact with no more than a few hundred other addresses its lifetime. This means that most wallets have a limited set of interactions with contracts and addresses in general and their interactions are classifiable to a very large degree. Keeping this in mind, Helix2 is an attempt to provide Ethereum with a next-generation link service, in addition to the name service already provided by ENS. The expected outcome of the Helix2 protocol is a link-native naming ecosystem, a 'linkspace', where an interaction between two names is representable on-chain similar, but not limited to, a human-readable name for an address. A shift in focus from nodes to links has a profound effect on the nature of structures that an ecosystem can support. Several features which are

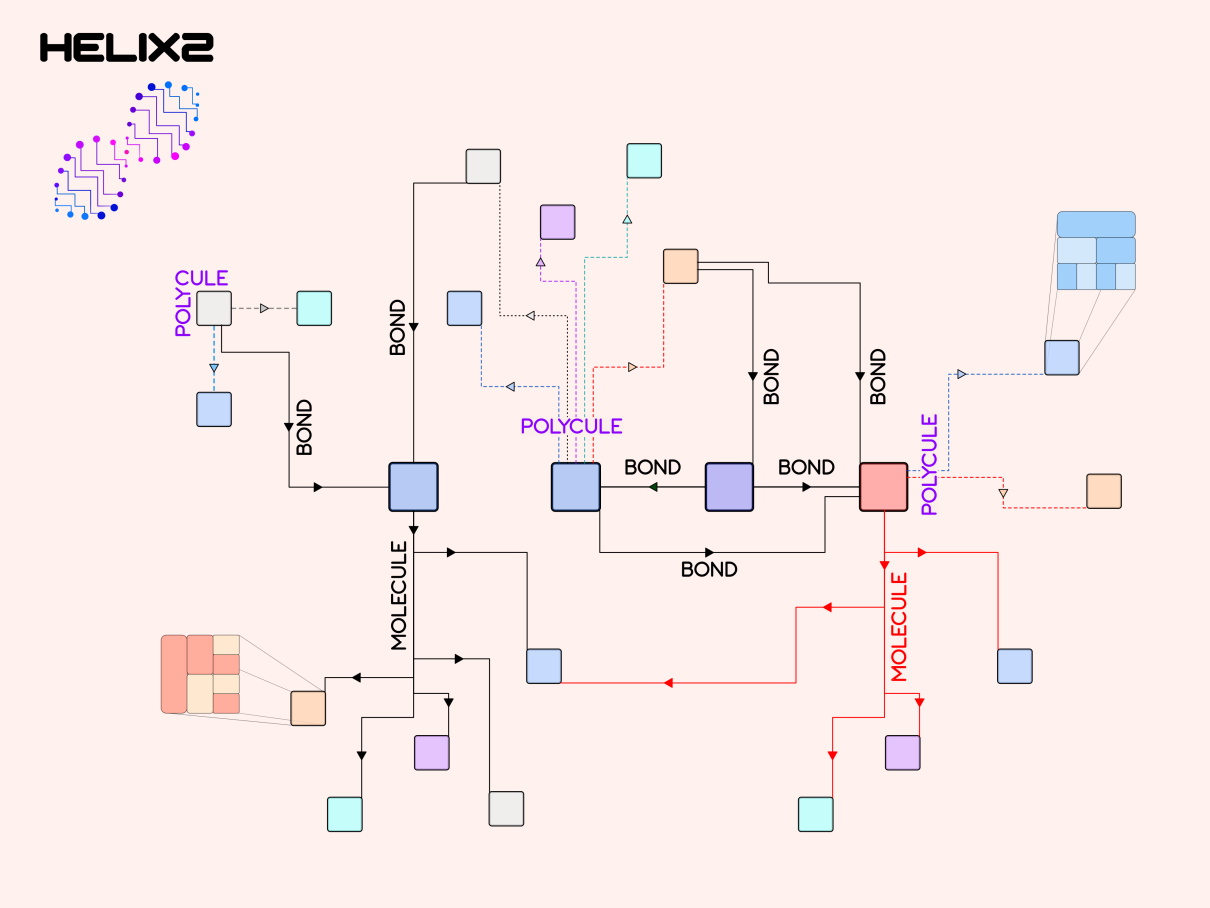

Figure 1: Helix2 Concept

challenging to achieve with a name service alone become extremely convenient with Helix2. For instance, Helix2 comes with several fundamental and easily accessible structural features such as stealth payments, social graphs, DAOs etc.

While Helix2 has its own namespace, it does not replace ENS and is in fact intended to work alongside ENS as an extension. Helix2 is able to import all `.` namespaces by design without any bridging or wrapping. Lastly, Helix2 is not the only link service in the works; Woolball is another link service currently under development although the two implementations arguably have more differences than similarities.

DEFINITIONS

Helix2 (Helix × 2) is roughly motivated by the double helix structure of DNA, where two polynucleotide chains are connected by bonds. The blockchain representation of this structure is two identical copies of the same blockchain connected by secure or unsecure bonds. The nomeclature is therefore borrowed from fundamental chemistry. Below is a list of definitions that will appear in this document:

A **name** is an on-chain label for a node in a distributed system. An example of this is `vitalik.eth` labeling the address of Vitalik on Ethereum.

A **link** is an open relationship between two names where one name (source) may

interact with another name (target) without latter's explicit approval. An example of a 'link' is transfer of ERC20 or ERC721 tokens from one name to another, e.g. `vitalik.eth` and `virgil.eth` are linked via ERC interface.

A source in Helix2 architecture is called a **cation**.
A target in Helix2 architecture is called an **anion**.

A **bond** is a closed relationship between two names that requires explicit approval of both the cation (source) and the anion (target). An example of a bond is granting of controller permissions to an operator by an owner using ERC721 `setApprovalForAll()` function. In this case, the interaction between the owner and controller is of closed nature[1]. In other words, a bond is a secure link since it requires explicit approval. Alternatively, a link is an unsecure bond.

Covalence is a flag in Helix2 architecture that defines whether an interaction or relationship is a link or a bond. Covalence is `true` for a secure bond and `false` for an unsecure bond (aka link).

A **molecule** is a set of similar bonds between a cation and several anions. An example of a molecule is the set of links between a DAO governor contract (source) and the participating token holders (targets). This molecule is secure since such membership of a DAO requires explicit approval of both the engaging entities. The individual bonds in this molecule are similar in the sense that the anions share the same relationship with the contract. A multi-sig vault is also an example of a molecule. A public channel in a Discord server is another example of a molecule.

A **polycule** is a special type of molecule in which each individual bond between the cation and its several anions is unique. An example of a polycule is the set of private channels in a Discord server, where each channel may have its unique requirements as well as unique participating members.

A **hook** is a contractual (or non-contractual) relationship (a contract) between two names. Technically, a hook is a map consisting of two elements `rule` $\rightarrow$ `config`.

A **config** is the address of the contract defining a hook between two names. For non-contractual relationships, `0x0` config is used.

A **rule** is an identifier which encodes the relationship between two names and must accompany a hook as its calldata.

HELIX2 DESIGN

ETHEREUM NAME SERVICE (ENS)

All name services so far have essentially been on-chain scalar databases (e.g. ENS, LENS, LNR, CB.ID), meaning that names are simply isolated nodes representable by one label (see figure textcolorblue1 below). ENS is a good

[1] Although this example uses addresses as nodes, a similar implementation is nonetheless possible with names.

example of such a scalar architecture. Typical name services like ENS are also hierarchical namespaces in the sense that the set of subnodes and nodes form a merkle-tree with labels as leaves of the tree. In such an architecture, names are self-contained and isolated by construction. Refer to figure 2.

HELIX2 SERVICE

Helix2 is essentially two services under one protocol: a native namespace providing a name service and a link service for that underlying namespace. Helix2 is an on-chain vector database. In Helix2, names can bond (or link) with one another; bonds are vectors between names, pointing from one name to another – this is the core premise of Helix2 protocol and essentially what separates it from other services. Helix2 names are not hierarchical, meaning that they cannot have subdomains, i.e. Helix2 is a flat namespace. Subdomains are not necessary in Helix2 since linking provides the same features without constraining the relationships between nodes to within one parent node's hierarchy. This is the only significant divergence of Helix2 namespace from ENS. In addition to this flat namespace, Helix2 has an additional linkspace which is the core utility of the protocol.

While the theoretical premise of a linkspace appears easy, in practise, this premise requires some crucial modifications to work in a resource-constrained environment such as Ethereum Virtual Machine (EVM). One of these modifications consist of allowing one name to link with multiple names within one object with common or unique relationships among them. This is highly consequential in reducing gas costs for serial linkers.

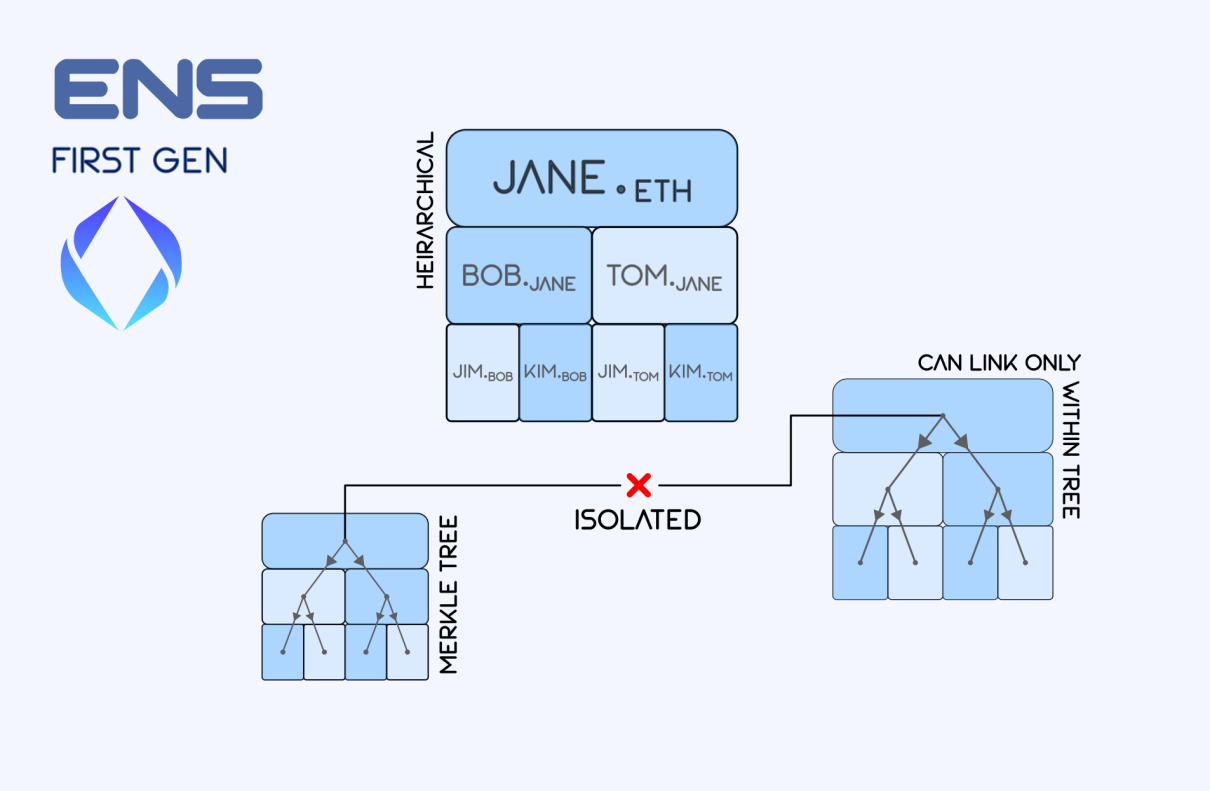

Figure 2: Ethereum Name Service (ENS)

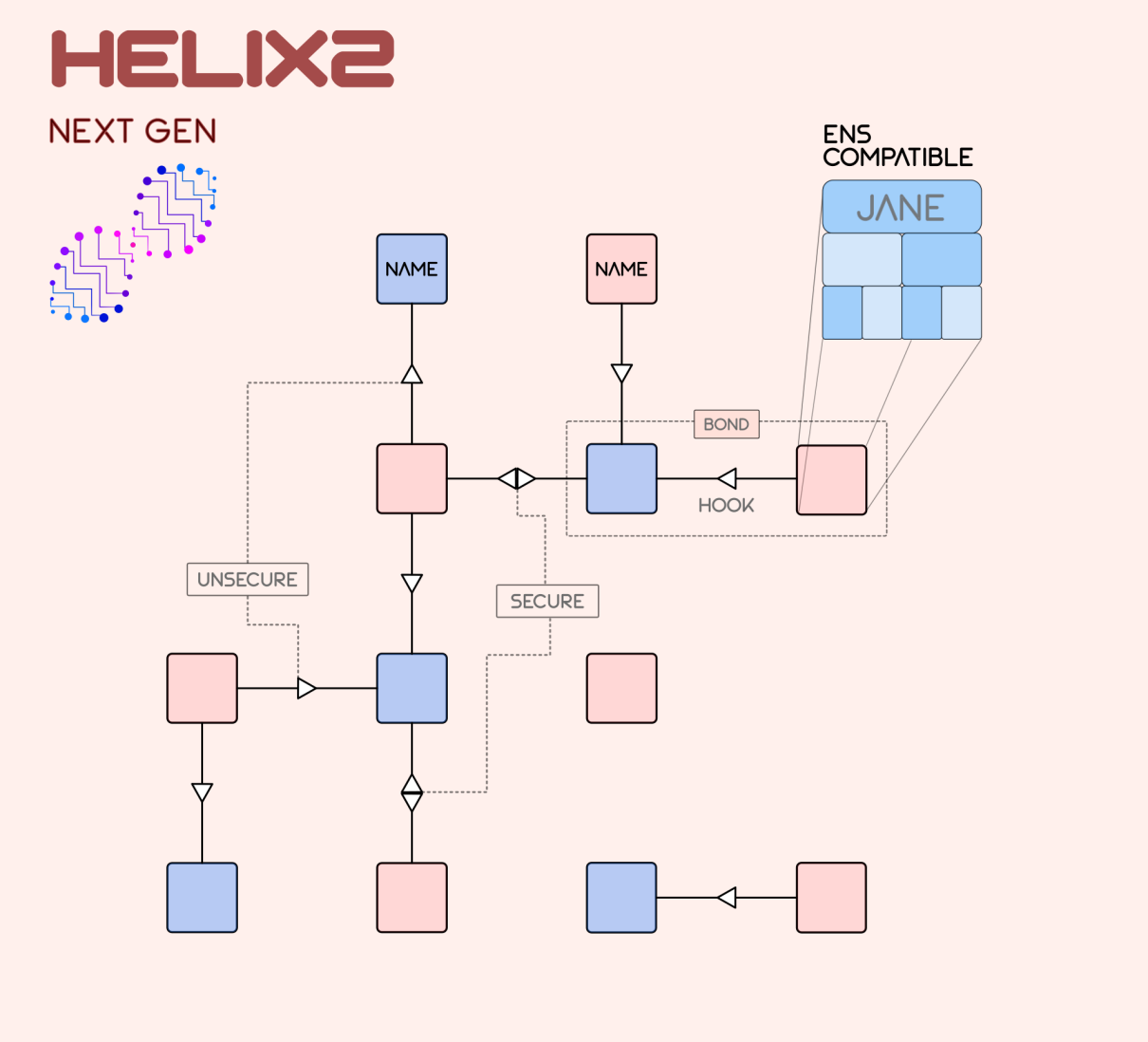

Figure 3: Helix2 Service

HELIX2 PROTOCOL

All native objects (names, links, bonds etc) end with `.`, e.g. `alice.`, where `.` acts as a trailing marker. Consequently, `.` is one of the three reserved characters in Helix2 and cannot be used in any of the object labels (other than as a suffix).

NAMES

```
struct NAME {
  address owner;
  address controller;
  address resolver;
  uint256 expiry;
}
```

Helix2 names are similar to ENS names, except that the suffix for them is `.` instead of `.eth`. Helix2 names are not hierarchical, meaning that they cannot

have subdomains.

All Helix2 names end with `.` and they have a Resolver and Controller. Note again that `.` is a reserved character and therefore forbidden. Additionally, `-` and `#` are also forbidden.

IMPORTING ENS NAMES

Helix2 is capable of importing all `.` namespaces by forbidding `.` in its native namespace. In the prototype implementation of Helix2, ENS name import is already implemented and ENS users can claim their ENS on Helix2. For ENS names, the representation of names then looks like `alice.eth.`, i.e. ENS name `alice.eth` followed by a `.` as the usual trailing marker.

BONDS

```
struct BOND {
   uint8[] rules;
   mapping(uint8 → address) hooks;
   bytes32 anion;
   bytes32 label;
   address resolver;
   address controller;
   bool covalence;
   uint256 expiry;
}
```

A directional bond between two names `alice.` → `bob.` is labeled by its `label`. Bonds start with `-`, end with `.` and can be queried by their label prefixed with `-`, e.g. `-label.`.

The source of the bond is called a cation (`alice.`) and the target is called an anion (`bob.`).

Further, `covalence` flag determines whether the bond is 'secure' or 'unsecure' (i.e. when the bond is in fact a link). To reiterate,

- a bond between alice and bob is insecure when it is uni-directional and requires only alice's approval, or a
- a bond between alice and bob is secure when it is mutual, bi-directional and requires both alice's and bob's approval.

Covalence can be set (`true`) or unset (`false`) through co-signing a message (2/2 multi-sig) by the cation and the anion.

HOOKS, RULES & CONFIG

The most important features of bonds are `config` and `rules`, combining to form `hooks`, which quantify the link between two names and give meaning to the `-` representation. Hooks are contractual mappings `rule → config` between two names mediated by ordered one-to-one mapping inside `uint8[] rules`:

`mapping(uint8 → address) hooks;`

- To query a hook's config inside `hooks` for a bond, one needs its associated `rule`, which is a `uint8` identifier mapping to the contractual address `config`. Hooks are thus queryable as `-label#rule.`, e.g. `-label#404.`.

- A trivial application of a hook is a payment router, i.e. payment sent to `0` hook `-label#0.` is routed to the address of `bob.`; more on hooks in upcoming sections.

MOLECULES

```
struct MOLECULE {
   uint8[] rules;
   mapping(uint8 → address) hooks;
   bytes32[] anions;
   bytes32 label;
   address resolver;
   address controller;
   bool covalence;
   uint256 expiry;
}
```

Helix2 allows for multi-bonding such that a cation can bond with multiple anions within one data structure instead of creating individual (and costlier) bonds; this structure is called a 'molecule' (or 'moly' in short). In a molecule, all individual bonds share the same covalence.

Other features of a molecule are similar to that of a bond, e.g. a molecule can have an label and hooks. To denote a bond with label `label`, we use two consecutive `--` characters, i.e. `--label.` without refering to an anion since molecules are anion-agnostic and hook-independent. By using `--label#rule.`, one can refer to a unique hook for a molecule. Covalence can be set (`true`) or unset (`false`) through co-signing a message by the cation and all the anions.

POLYCULES

```
struct POLYCULE {
   uint8[] rules;
   mapping(uint8 → address) hooks;
   bytes32[] anions;
   bytes32 label;
   address resolver;
   address controller;
   bool covalence;
   uint256 expiry;
}
```

Lastly, we can define another useful abstraction in the form of a 'polycule', which is a molecule comprising of unique bonds between a cation and a set of anions. In a polycule, all individual bonds share the same covalence despite being unique. In short, `rules.length == anions.length`, and `rules` & `anions`

are a one-to-one map.

Other features of a molecule are similar to that of a molecule. To denote a bond with label `label`, we use three consecutive `---` characters, i.e. `---label.` etc. By using `---label#rule.`, one can refer to a unique hook for a molecule by its `rule`. Alternatively, one can refer to a unique anion in a molecule by its indexed `rule`, e.g. `---label#anions[rule]`. Covalence can be set or unset through co-signing a message by the cation and all the anions.

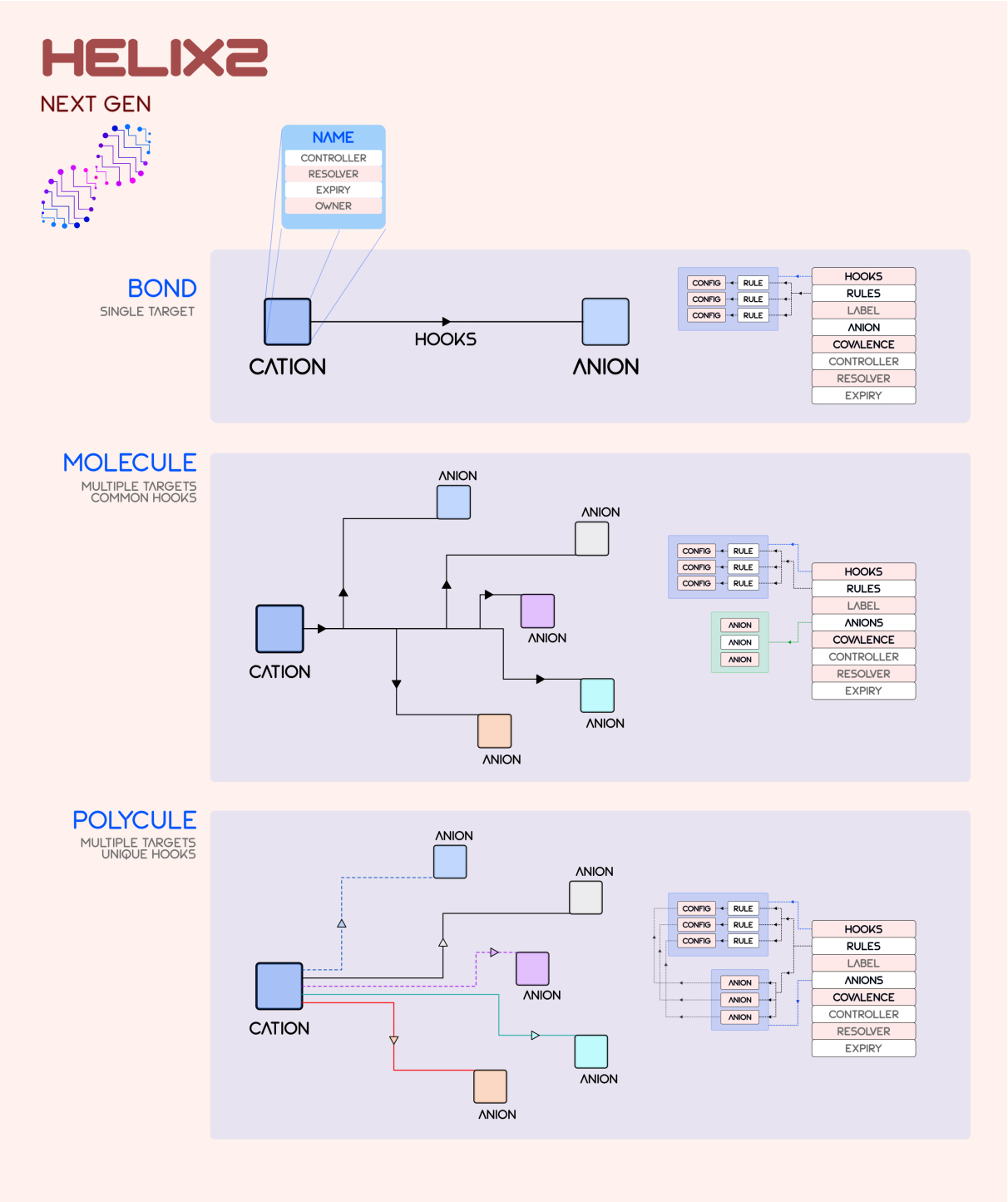

Figure 4: Helix2 Classes

The `molecule` structure is a topological superset i.e. it is possible to derive polycules and bonds from molecules although that'll literally be a gas-guzzling mistake. The seemingly unnecessary differentiation between the three is to optimise gas consumption. Helix2 architecture leaves room for future upgrades with new submodules. These submodules may be referenced by their integer index N as `-N-label#rule.`, e.g `-42-vitalink#404.`.

CONTRACTS

Helix2 Protocol is a fairly large set of code divided into clean modules in a way that all submodules are replaceable and upgradeable without breaking any existing architecture or interfaces. This is achieved through a combination of basic logic & storage separation via proxies and EIP-2535 Diamond Standard.

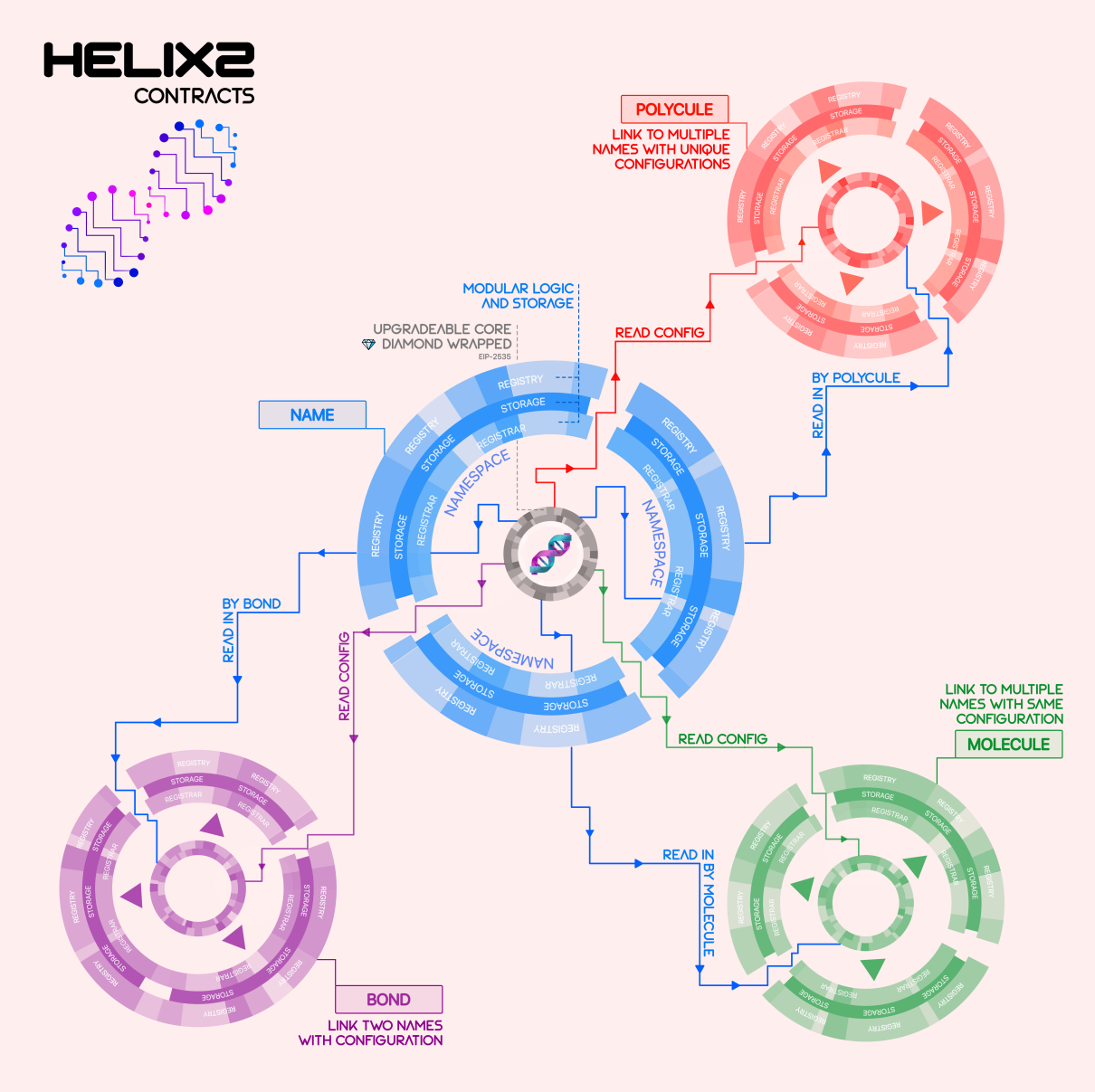

Figure 5: Helix2 Contracts

There are four submodules in the genesis version of protocol – Name, Bond, Molecule and Polycule. Each submodule is composed of three contracts – Registrar, Registry & Storage. In each submodule, the storage contracts are proxied by registry contracts such that the logic and storage is seperated. This enables the protocol to replace the storage and logic in each submodule seemlessly through future protocol upgrades without compromising on-top services. Name submodule is somewhat more central to the architecture than the remaining three link modules since links depend on and derive from names.

The four submodules are connected at the core by the Helix2 Core manager which is the bespoke state-of-the-art Multi-facet Proxy (EIP-2535). EIP-2535 compliant contracts are fully upgradeable and it allows Helix2 protocol to use one single address forever without compromising on future functionalities or breaking existing on-chain or off-chain functionalities. Helix2 manager is capable of excepting new submodules, replace existing submodules or update core configuration.

ACCOUNT ABSTRACTION

Helix2 is a core infrastructure which can be leveraged for perhaps countless utilities. One of the biggest utilities of Helix2 arises in account abstraction. **A**ccount **A**bstraction (AA) is an umbrella term first defined in EIP-2938, to categorise infrastructures that allow the use of smart contracts as wallets instead of the typical logic-free 'externally owned accounts' (EOA) that are currently in use as wallets. The core ideology behind account abstraction is to allow users to arbitrarily customise their handling of transactions, funds and interactions with the Ethereum blockchain in general with smart contract code.

Several decentralised applications (dApps) on Ethereum already allow specific smart control over blockchain interactions (e.g. Gnosis Vault, Umbra Cash, Tornado Cash, Gas Station Network) but this list is rather small and the utility of them rather limited to only a few aspects. These services may additionally require users to pay additional fees on top of the (un)optimised gas costs.

EIP-2938 proposed a new set of `OPCODES` that would allow account abstraction at core level but this required significant non-trivial changes to Ethereum consensus layer. EIP-4337 proposed another infrastructure that uses alternative mempools ('alt mempools') for bundling transactions and then routing them to the consensus layer upon validation (akin to Flashbots); the alt mempools would then be responsible for validating the smart wallet transaction calls and thereby such a validation & routing environment remains independent of the primary mempool and wouldn't require changes to the consensus layer itself. Due to the independent nature of the alt mempools, there is also room for mempool-level optimisations (e.g. of bundlers) to further minimise gas costs which is usually not the case for typical dApps.

Helix2 is also an account abstraction infrastructure that uses a namespace and its associated linkspaces, i.e. data structures containining pointers to smart contracts from names, to act as a router of transactions for any name (see figure 6). Since the name- and linkspace are disjoint from the underlying EOA, the user pays no other gas costs than the standard network transaction fees, excluding the one-time cost of writing the pointers in the Helix2 data structure. Wallets connecting users to the blockchain via their names are therefore abstracted by Helix2 since the user can insert arbitrary logic to their name- and linkspace. ENS already provides a fraction of this functionality but loses out on the ability to enforce dynamic arbitrary

front-facing logic due to its heirarchical nature. The several specific usecases of Helix2 mentioned in the upcoming sections are all examples of account abstraction.

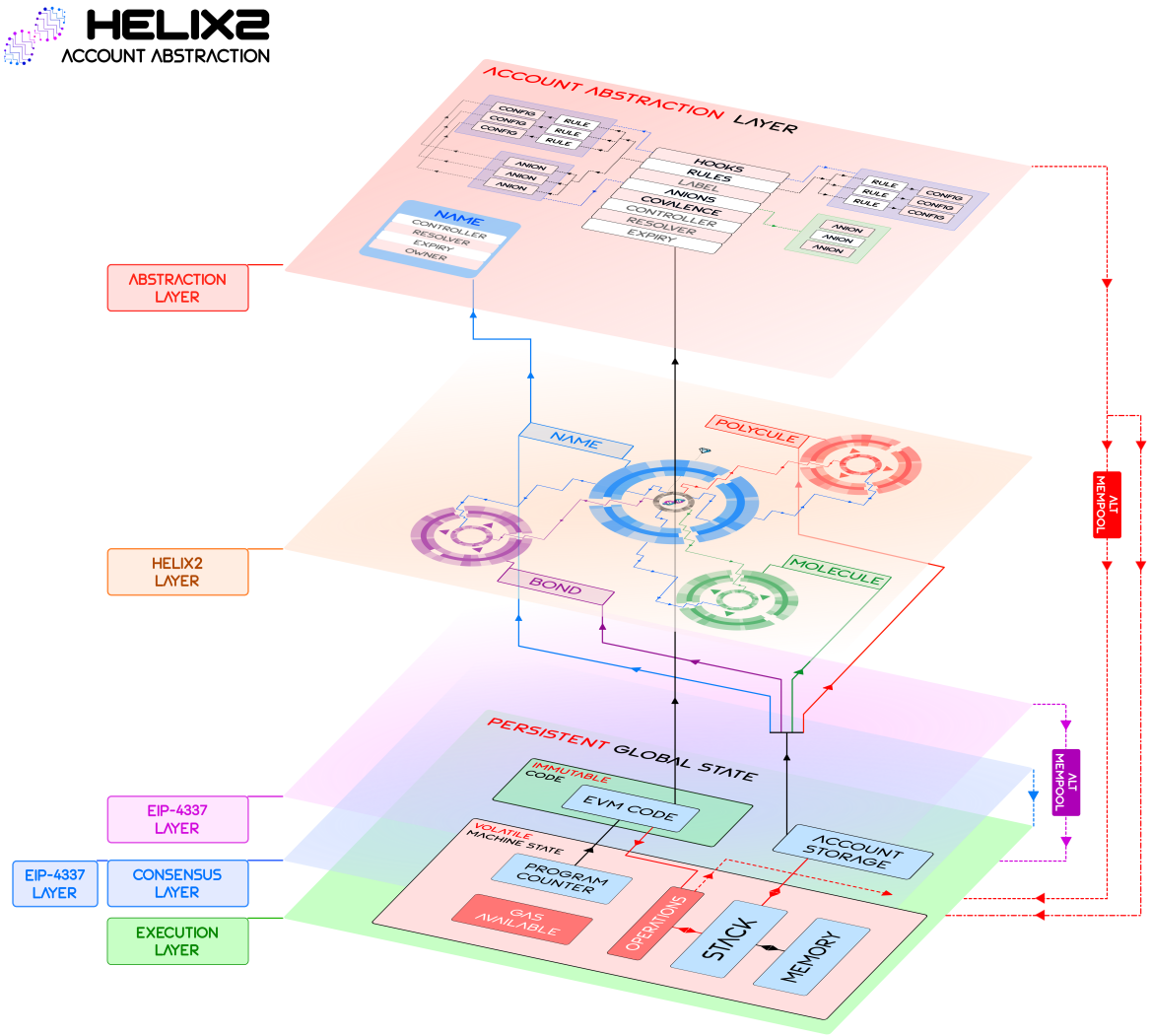

Figure 6: Account Abstraction built on Helix2

MULTI-SIG VAULTS

By design, Helix2 molecules are a perfect fit for multi-sig vault management. Molecule struct is readily mappable to a multi-sig safe architecture and it readily formalises vaults as nameable linkspaces. In fact, Helix2 can store multiple vault agreements among a set of parties inside a single standard struct.

STEALTH PAYMENTS

Helix2 hooks can be configured for bonds to receive stealth payments to a name via a stealth protocol such as Umbra Protocol or Aztec. Note that this feature is also possible with ENS Resolver. In addition however, Helix2 allows performing stealth payments enmasse and with better control over sensitive data streams.

WEB3 SOCIAL GRAPHS

Helix2 molecules are a natural fit for open social graphs and could aid in better decentralised social media protocols. In the current implementation, molecules are an appropriate representation of social groupings, although a complete version requires one or more submodules that can link molecules and polycules among each other. This requirement may vary from one platform to another, and is therefore not part of the Helix2 core protocol.

MESSAGING & CHAT APPS

Helix2 molecules are a natural fit for web3 messaging apps and chat services. In particular, molecules are a canonical choice for organising group chat participants and their permissions whereas polycules could be utilised to manage private channels. Both in combination could help build the elusive web3 version of Discord.

DAO

Helix2 molecules are a canonical fit for DAO governance and associated tooling where delegates can be viewed as members of the same molecule with common relationship among all members of the DAO. Orca Protocol is a similar idea which has been implemented already albeit with limited focus on on-chain coordination. Helix2 in comparison makes the same task significantly easier at protocol level and allows formation of highly complex pods at much lower architectural cost.

Helix2 molecules are a canonical fit for DAO governance and associated tooling where delegates can be viewed as members of the same molecule with common relationship among all members of the DAO. Orca Protocol is a similar idea which has been implemented already albeit with limited focus on on-chain coordination. Helix2 in comparison makes the same task significantly easier at protocol level and allows formation of highly complex pods at much lower architectural cost.

OTHER UTILITIES

Other possible use-cases include

- a unified web3 individual and group reputation system,
- proof-of-humanness by defining human-specific hooks in molecules or polycules,
- and perhaps anything and everything that requires structuring in groups.

FUTURE

The architecture of Helix2 service described in this paper promises several advances and features with clear utility across the Ethereum ecosystem ranging from identity to stealth payments.

Nonetheless, it will be naive to think that its usecases will be limited to what is discussed in this paper. The architecture of Helix2 protocol is fairly low level and it has been designed for maximum upgradeability and dynamism with as little friction as possible. The four submodules introduced in this paper are only the most basic structures and the Helix2 core expects several new

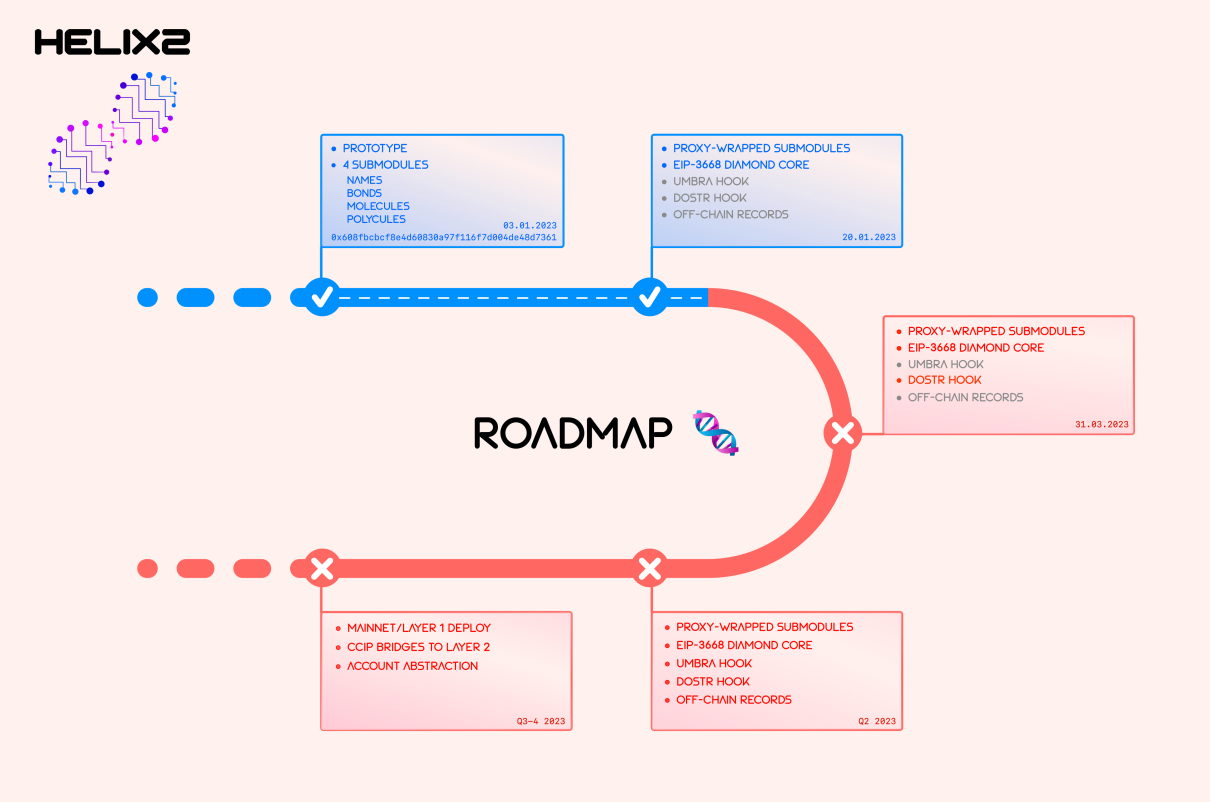

Figure 7: Helix2 Roadmap

submodules in the future under its repertoire.

The codebase of Helix2 at the time of writing is at a fairly advanced stage with its upgradeable core and the initial set of four submodules in place (see figure 6). In near future, Helix2 expects to integrate off-chain lookups for extremely large link structures using Chainlink's CCIP Protocol, integrate presets for stealth payments (via Umbra) and a bridge to Dostr – an Ethereum-flavoured Nostr client. Upon completion of these tasks, full Helix2 Protocol will be deployed on Ethereum Mainnet L1, with CCIP bridges to all primary L2 chains such as Optimism, zkSync, Starknet etc following soon after. Coordinates of the WIP codebase are at the end of the document.

ACKNOWLEDGEMENTS

Helix2 is a work in progress and we thank `0xc0de4c0ffee` for their valuable insights. We invite individuals and parties with interest in contributing to Helix2 protocol to get in touch via Github.

REFERENCES

Ethereum Name Service (ENS): ens.domains
Tezos Domains: tezos.domains
Solana Name Service: naming.bonfida.org
Woolball: woolball.xyz
EIP-2535 Diamond Standard: EIP-2535
Umbra Protocol: umbra.cash
Aztec Protocol: zk.money
Orca Protocol: orca.mirror
Cross-Chain Interoperability Protocol (CCIP): chain.link/cross-chain
Aztec Protocol: zk.money
Nostr (**N**otes and **O**ther **S**tuff **T**ransmitted by **R**elays): nostr.com
Dostr (Ethereum-flavoured Nostr): dostr-eth
EIP-2938 (Account Abstraction): EIP-2938
EIP-4337 (Account Abstraction Using Alt Mempool): ERC-4337
Maximum Extractable Value (MEV) & Flashbots: Flashbots
Tornado Cash dApp: Tornado.Cash
Gas Station Network dApp: OpenGSN.org

METADATA

Github: github.com/helix-coupler
Contracts: github.com/helix-coupler/helix2-contracts
Source: github.com/helix-coupler/resources
SHA-1 Checksum: B83B2967956314E753391BFEC7F227CD
Date: March 28, 2023