

Multichain Trustless Bridge - Draft WIP

John Whitton

October 2022

Contents

1	Introduction	3
2	Our Contributions	5
3	Background and Related Work	6
3.1	Bridging Solutions	6
3.1.1	Horizon Bridge	6
3.1.2	Nomad	7
3.1.3	Map Protocol	8
3.1.4	Wormhole	9
3.2	Foundational Layers	10
3.2.1	On Chain Components:	10
3.2.2	Light Clients:	10
3.2.3	Bridging Components:	10
3.2.4	Communication Protocols	10
3.2.5	Zero Knowledge Proofs:	11
3.2.6	Dedicated Chain:	12
4	Our Solution	13
4.1	Smart Wallet	13
4.2	Universal Token Management	13
4.3	Dedicated Chain	13
4.4	Multichain Bridging	13
4.5	On Chain Components	13
4.6	Zero Knowledge Proof Improvements	13
	References	14
	Documentation and Github References	16
	Bridge Hacks	18

1 Introduction

As of October 4th, 2022, crypto assets totalled \$998 Billion with over 100 disparate chains holding assets.

A unifying vision of the distributed ledger technology is to connect multiple distributed systems (chains) together. However as these systems have been developed over \$2 Billion has been stolen across 13 separate cross-chain bridge hacks.[1].

Here we give an overview of various existing approaches to the bridging of assets between chains and expand on the Horizon Bridge design [2] to cater for bridging to ethereum 2.0 [1] Proof of Stake. Revisiting light client construction, frequency of block Header synchronization, economics and incentives for relayers and multi-chain support.

Global Cryptocurrency Market Cap Charts

The global cryptocurrency market cap today is \$998 Billion, a 2.38% change in the last 24 hours and -55.14% change one year ago. As of today, the market cap of Bitcoin (BTC) is at \$383 Billion, representing a Bitcoin dominance of 38.39%. Meanwhile, Stablecoins' market cap is at \$149 Billion and has a 14.92% share of the total crypto market cap.

Total Crypto Market Cap Chart

The chart below shows the total market cap & volume of cryptocurrencies globally, a result of 13,189 cryptocurrencies tracked across 590 exchanges.



Figure 1: Crypto Market Capitalization

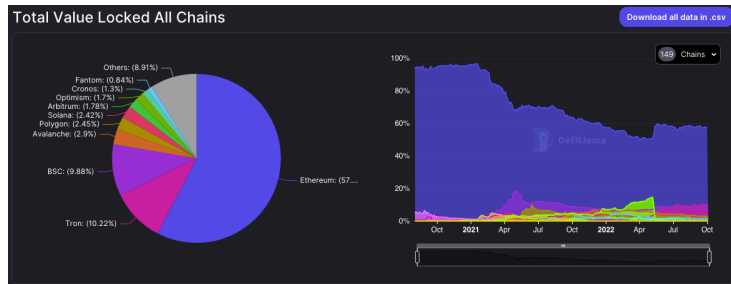


Figure 2: Total Value Locked by Chain

2 Our Contributions

We propose a top down design starting with the following components

1. **Smart Wallet:** A Multichain secure, frictionless, programmable non-custodial wallet infrastructure. [20]
2. **Universal Token Management:** provides the ability to track deposits via a token management layer which can be used to reduce gas fees and also allow idle funds to be leveraged. (similar to Ocean Protocol[23], Sushi's bentobox [22] and balancer's vault[21]).
3. **Dedicated Chain:** We propose a purpose driven relay chain which includes consensus support for destination chains similar to map's atlas chain [16] and additional on-chain infrastructure similar to wormchain [10].
4. **Multichain Bridging:** A hub and spoke model building with Guardians to validate the authenticity of the transactions and Relayers to relay the messages to the various chains (similar to wormhole [9]).
5. **On Chain Components:** Smart Contracts to bridge tokens between chains and a Gas Management Layer to reduce gas fees and user friction by enabling users to pay gas across the multichain transactions using a single token. (similar to acala's flexible fees [24]).
6. **Zero Knowledge Proof Improvements:** We propose improving the cryptographic proof construction and checking of proofs as part of the light client process similar to map protocol [4]. And an ultralight blockchain client (similar to Plumo [3]).

3 Background and Related Work

3.1 Bridging Solutions

3.1.1 Horizon Bridge

Horizon Bridge is a gas-efficient, cross-chain bridge protocol to transfer assets from a BFT blockchain to another blockchain (e.g., Ethereum) which supports basic smart contract execution.[2]

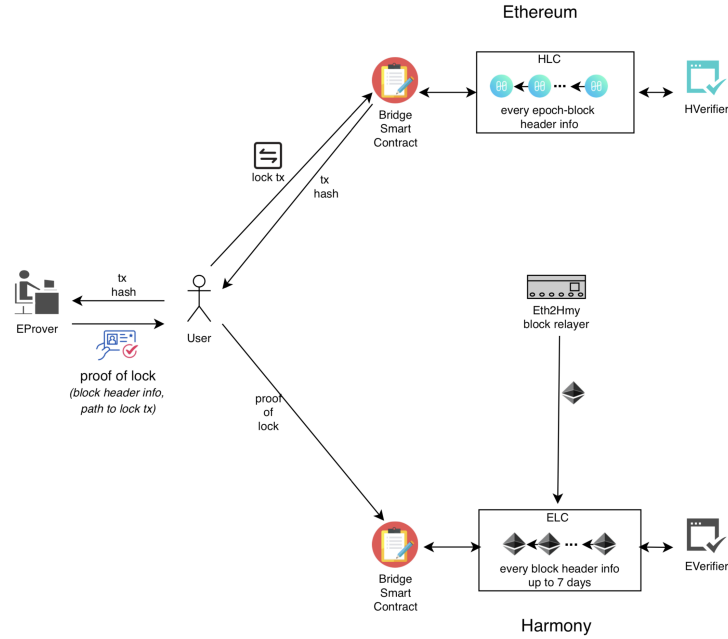


Figure 3: Horizon Ethereum to Harmony Flow

Hack: The initial version of the horizon bridge (not the trustless version currently under development [13]). Was hacked for \$100 million in tokens when the private keys of the multisig holding the locked funds were compromised.[5]

3.1.2 Nomad

Nomad is an optimistic interoperability protocol that enables secure cross-chain communication. [5] Optimistic verification doesn't use light clients or natively verify cross-chain messages. Instead, messages are optimistically signed on the origin chain, and a timeout period is enforced on the destination, during which the message can be inspected and vetoed if anything awry is noticed.

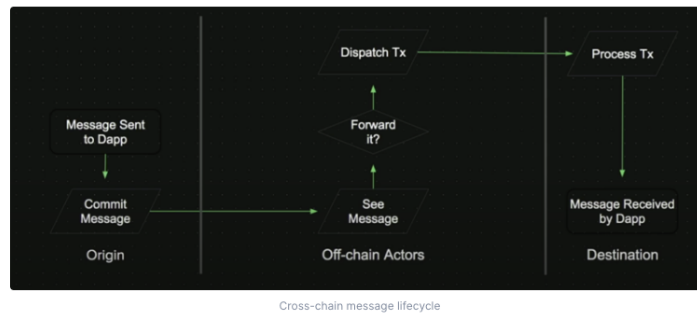


Figure 4: Nomad Cross Chain Message Life Cycle

Hack: Nomad was hacked for \$190 Million after a hacker found a vulnerability in Nomad's code[14] that failed to properly validate that a message was approved before processing it.[6]

3.1.3 Map Protocol

Map Protocol is an Omnichain Layer of Web3 with Provably Secure Cross-chain Communication Built upon Light-client and zk-SNARK technology.[4] It includes it's own relay-chain [16] which embeds each destination chains consensus algorithm as precompiled contracts to facilitate light client construction.

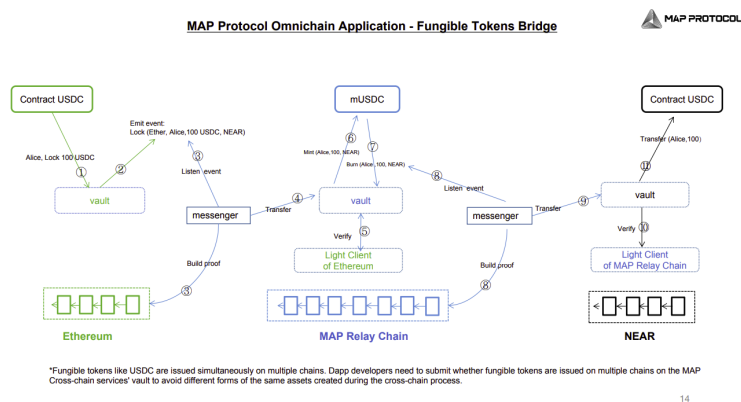


Figure 5: Map Protocol Fungible Token Flow.

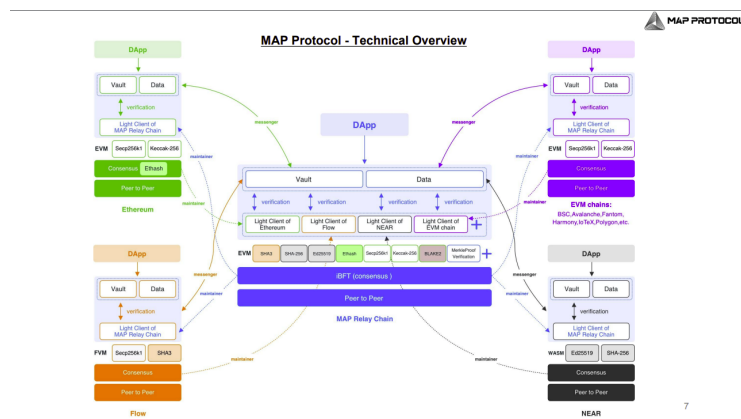


Figure 6: Map Protocol Architecture

Hack: As of Oct 4th, 2022 no hack of map protocol has been reported.

3.1.4 Wormhole

Wormhole is a generic message passing protocol that connects to multiple chains including Ethereum, Solana, Terra, Binance Smart Chain, Polygon, Avalanche, Oasis, Fantom, Karura, Celo, Acala, Aurora and Klaytn.[9].

Wormhole has a Core Bridge contract deployed on all the connected networks. Wormhole Guardians run a full node for each of the connected chains, specifically listening to any events from the Core Contracts. The core contracts emit a message, which is picked up by the Guardians. The Guardians verify the message and sign it, creating a VAA (Verified Action Approval). This VAA then sits on the Guardians network where it can be retrieved by the user or by a relay to be submitted to the target chain to process the message. Unlike other bridges, a relay in Wormhole has no special privileges, it's just a piece of software that shuttles messages between the Guardian network to the target chain, and is not a trusted entity.

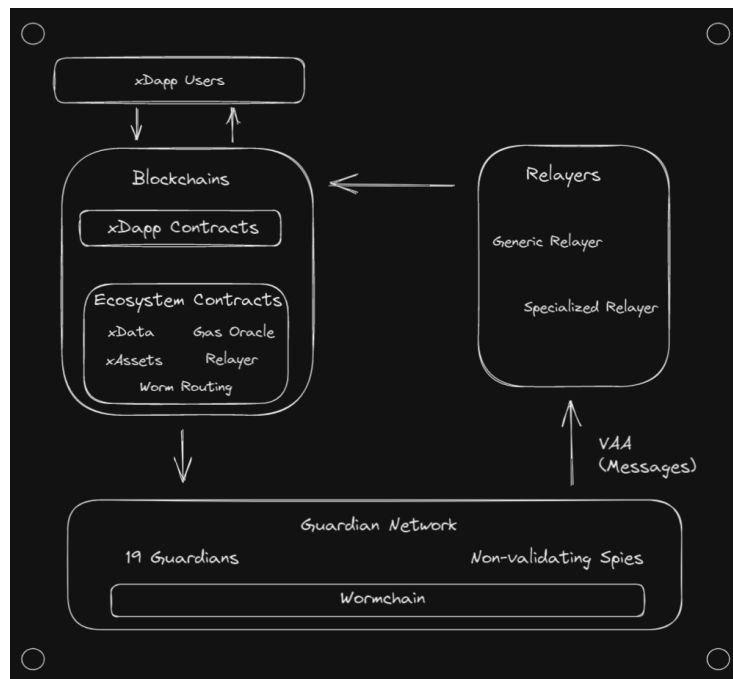


Figure 7: Wormhole Architecture

Hack: Wormhole[17] was hacked for \$326 million after an attacker exploited the use of a deprecated, insecure function to bypass signature verification.[3]

3.2 Foundational Layers

3.2.1 On Chain Components:

- **XCLAIM**
- **Horizon**
- **Wormhole**

3.2.2 Light Clients:

- **Proof of Work:**
- **Proof of Stake:**
- **Bitcoin:**
- **Solana:**
- **Polkadot:**
- **Near:**

3.2.3 Bridging Components:

- **Guardians :**
- **Relayers:**

3.2.4 Communication Protocols

- **Cosmos IBC**
- **Polkadot XCMP:**

3.2.5 Zero Knowledge Proofs:

Refactor Light-Client With zkSNARK

- Light clients in MAP Protocol are constructed as bulletproof smart contracts on MAP Relay Chain and all connected blockchains
 - While transparency and decentralization are well preserved with on-chain smart contracts, gas consumption is non-negligible
 - With the recent development of zkSNARK, the foundation of MAP protocol, light client construction and cross-chain proof verification can be reshaped
 - Correctness of light client's state transition is vital to MAP protocol as it determines the validation of cryptographic proof for cross-chain message
- Current light-client and cryptographic proof construction for PoS + BFT chains, as of MAP Relay Chain
 - Light client tracks latest validator set, as new validator set is normally authorized by the old one, light client checks that the new one is approved via enough signatures
 - Cryptographic proof for an event includes the Merkle proof of the event in the receipt tree as well as the corresponding block header
 - Proof check: check the block header is signed by enough validators, then check the Merkle proof against the Merkle root carried by the block header
- Current light-client and cryptographic proof construction for PoW chains, as of Ethereum
 - Light client tracks latest block headers, the hash link and accumulated work are checked for new block headers submitted to the light client
 - The cryptographic proof of cross-chain message for an event is just the Merkle proof of the event in the receipt tree
 - Proof check: check the Merkle proof against the Merkle root carried by the corresponding block header, that is maintained by light client
- With zkSNARK technology, both the light-client construction and proof check process can be improved
 - Signature check, Merkle proof check against certain Merkle root, as well the hash link and accumulated work check are all suitable to be certified via zkSNARK
 - On the light client construction side, instead of store tons of validator set info or block headers, a simple commitment should be enough
 - The commitment is about the validator set (PoS) or latest block header set (PoW), and each time the set change, the commitment is updated
 - Here we employ the zkSNARK to prove that the change from old commitment to new commitment reflects a valid change of validator set or block header set
 - E.g., in PoS setting, it means that the new validator set related to the new commitment stored in the light client is valid regarding to the old set related to the old commitment
 - The constraints imposed by zkSNARK mainly includes checking that enough old validators have approved the new set and the voting weight passes a certain threshold
 - Basically, the complexity is dealt with inside the zkSNARK constraints where the smart contract implementation is freed from the scary details
- Similar ideas by Celo project, aka. the PLUMO ultralight blockchain client, but MAP Protocol has a different goal
 - PLUMO utilizes two specialized curves, aka BLS12-377 and BWT, to realize a recursive style zkSNARK to gain better efficiency
 - The curve selection fits the need of PLUMO's goal, to enable light client running on smart phones
 - Yet, with MAP protocol, we are trying to connect with more blockchains via smart contracts and these two specialized curves contradicts our goal
 - With MAP protocol, we are trying to find the greatest common divisor among all blockchains and hence we prefer the BN256 curve which is already widely adopted in EVM world
 - We are still exploring the idea listed above to find the proper balance between the engineering difficulty, gas consumption, off-chain computation resource required, etc.

13

Figure 8: Map Protocol zkSNARK Improvements

Main circuit
<p>We first define the following helper methods:</p> <ul style="list-style-type: none"> – EncodeEpochToBits($i, r, \delta, \delta', t, apk, \{pk_i\}_{i=1}^n$) : <ol style="list-style-type: none"> 1. Encode i, the epoch index, as a 16-bit integer. 2. Encode r, the consensus round number, as an 8-bit integer. 3. Encode t, the maximum number of non-signers, as a 32-bit integer. 4. Encode δ, the current epoch entropy, in 128 bits. 5. Encode δ', the parent epoch entropy, in 128 bits. 6. Encode each public key in $\{pk_i\}_{i=1}^n$ as a G_2 compressed point. If there are fewer public keys than the maximum defined in the system parameters, pad with G_2 until the maximum number of public keys is reached. – EncodeEpochToBitsEdges($i, \delta, \delta', t, apk, \{pk_i\}_{i=1}^n$) : <ol style="list-style-type: none"> 1. Encode i, the epoch index, as a 16-bit integer. 2. If this is the first epoch, encode δ', the parent epoch entropy in 128 bits. If this is the last epoch, encode δ, the current epoch entropy in 128 bits. 3. Encode t, the required signer threshold, as a 32-bit integer. 4. If this is the last epoch, encode apk, the aggregated public key of this validator set, as a compressed G_2 point. 5. Encode each public key in $\{pk_i\}_{i=1}^n$ as a G_2 compressed point. If there are fewer public keys than the maximum defined in the system parameters, pad with G_2 until the maximum number of public keys is reached. <p>Next we describe the main circuit. In the following let</p> $E_j = \{i_j, r_j, \delta_j, \delta'_j, t_j, apk_j, \{pk_{j,k}\}_{k=1}^n\}$ <p>A subroutine taking as input some E_j is assumed to discard those elements included in it which are not a part of the subroutine's input.</p> <ul style="list-style-type: none"> – MainCircuit($H'(e_1), H'(e_N) : \sigma_{agg}, \{H(e_j)\}_{j=2}^N, \{\mathbf{b}_j\}_{j=1}^{N-1}, \{E_j\}_{j=1}^N$): <ol style="list-style-type: none"> 1. For each $j = 2 \dots N$ perform: <ol style="list-style-type: none"> (a) Check that $apk_{j-1} = ? \sum_{i=1}^n b_i \cdot pk_{j-1,i}$ where b_i is the i-th bit of \mathbf{b}_{j-1}. (b) Check that $\delta_{j-1} = ? \delta'_j$ (c) Check that $i_{j-1} = ? i_j + 1$ (d) Encode E_j as e_j using EncodeEpochToBits and hash it using BHPedersenHash. Then, run Blake2Xs on the intermediate result to obtain the final result of the composite hash. Finally, complete the hash following the hash-to-group method described in Sections 5 and 6.1. Check that the result is equal to $H(e_j)$. 2. Check that $apk_N = ? \sum_{i=1}^n pk_{N,i}$. 3. Check that $e(\sigma_{agg}, G_2^{-1}) \cdot e(H(e_2), apk_1) \cdot \dots \cdot e(H(e_N), apk_{N-1}) = ? 1_{G_T}$ 4. Encode E_1 as e_1 and E_N as e_N each using EncodeEpochToBitsEdges. Hash individually both e_1 and e_N directly with Blake2s. Tightly pack, individually, the first and last epoch resulting hash bits into elements of \mathbb{F}. Check that the results of this packing are equal to $H'(e_1), H'(e_N)$ respectively.

Figure 9: Plumo Main Circuit

3.2.6 Dedicated Chain:

4 Our Solution

Here we give an overview of bridging an ERC20 token from Ethereum to Solana, Polkadot and Harmony.

4.1 Smart Wallet

4.2 Universal Token Management

4.3 Dedicated Chain

4.4 Multichain Bridging

4.5 On Chain Components

4.6 Zero Knowledge Proof Improvements

- Metadata Transaction:
- Metadata Relay:

References

- [1] Maksym Zavershynskiy, 2020 *ETH-NEAR Rainbow Bridge* <https://near.org/blog/eth-near-rainbow-bridge/>
- [2] Riongjan Lan, Ganesha Upadhyaya, Stephen Tse, Madhi Zamani (Harmony, Visa Research) 2020 *Horizon: A Gas-Efficient Trustless Bridge for Cross-Chain Transactions* <https://github.com/harmony-one/horizon/blob/main/docs/assets/horizon-whitepaper.pdf>
- [3] Psi Vesely, Kobi Gurkan, Michael Straka, Ariel Gabizon, Philipp Jovanovic, Georgios Konstantopoulos, Asa Oines, Marek Olszewski, and Eran Tromer. 2021 *Plumo: An Ultralight Blockchain Client* <https://eprint.iacr.org/2021/1361.pdf>
- [4] Map Protocol, 2022 *Omnichain Layer of Web3 with Provably Secure Cross-Chain Communication* https://files.maplabs.io/pdf/mapprotocol_Litebook_en.pdf
- [5] Nomad.xyz, 2022 *The Nomad Protocol* <https://docs.nomad.xyz/the-nomad-protocol/overview>
- [6] ParityTech, 2022 *BEEFY (Bridge Efficiency Enabling Finality Yielder)* <https://github.com/paritytech/grandpa-bridge-gadget/blob/master/docs/walkthrough.md>
- [7] Snowfork, 2020. *A trustless, general-purpose Polkadot $\dot{\jmath}$ Ethereum bridge.* <https://snowbridge-docs.snowfork.com/concepts/>
- [8] Christopher Goes, 2020 *The Interblockchain Communication Protocol: An Overview* <https://arxiv.org/pdf/2006.15918.pdf>
- [9] Wormhole Foundation, 2022 *Wormhole: a generic multichain message passing protocol* <https://docs.wormhole.com/wormhole/>
- [10] Wormhole Foundation, 2022 *WormChain* https://book.wormhole.com/wormhole/8_wormchain.html
- [11] Polkadot Wiki, 2020. *Cross-chain Message Passing (XCMP).* <https://wiki.polkadot.network/docs/en/learn-crosschain>
- [12] Research at W3F, 2020. *XCMP Overview.* <https://research.web3.foundation/en/latest/polkadot/XCMP/index.html>
- [13] Research at W3F, 2020 *HRMP Channels.* <https://research.web3.foundation/en/latest/polkadot/XCMP/HRMP%20channels.html>
- [14] Polkadot Wiki, 2020. *Bridges* <https://wiki.polkadot.network/docs/en/learn-bridges>

- [15] Parity Technologies, 2020. *Parity Bridges Common*. <https://github.com/paritytech/parity-bridges-common>
- [16] Ruitao Su, 2020. *Bringing BTC to Polkadot: Acala x Ren*. <https://medium.com/acalanetwork/bringing-btc-to-polkadot-acala-x-ren-e7959855d5aa>
- [17] ChorusOne, 2020. *wormhole: A Cosmos-Substrate bridge*. <https://github.com/ChorusOne/wormhole-bridge>
- [18] Ethan Buchman, University of Guelph, 2016 *Tendermint: Byzantine fault tolerance in the age of blockchains* https://atrium.lib.uoguelph.ca/xmlui/bitstream/handle/10214/9769/Buchman_Ethan_201606_MAsc.pdf
- [19] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. Knottenbelt, 2019 *XCLAIM: Trustless, Interoperable, Cryptocurrency-Backed Assets* <https://eprint.iacr.org/2018/643.pdf>
- [20] Modulo.so, 2022 *modulo.so secure, frictionless, programmable non-custodial wallet infrastructure* <https://modulo.so/assets/products.pdf>
- [21] Balancer, 2021. *Balancer The Vault* <https://docs.balancer.fi/products/the-vault>
- [22] Sushi, 2021 *BentoBox* <https://docs.sushi.com/docs/Developers/Bentobox/>
- [23] Shell Protocol, 2022 *The Ocean, Shell Protocol v2 White Paper, Part 2* https://shellprotocol.io/static/Ocean_-_Shell_v2_Part_2.pdf
- [24] Acala Foundation, 2022 *Acala Flexible Fees* <https://wiki.acala.network/learn/flexible-fees>

Documentation and Github References

- [1] Vitalik Buterin, 2020 *Vitalik's Annotated Ethereum 2.0 Spec* <https://notes.ethereum.org/@vbuterin/SkeyEI3xv>
- [2] Ethereum Github, 2022 *Consensus: Phase 0 – The Beacon Chain* <https://github.com/ethereum/consensus-specs/blob/master/specs/phase0/beacon-chain.md>
- [3] Vitalik Buterin, 2022 *Extended light client protocol* https://notes.ethereum.org/@vbuterin/extended_light_client_protocol
- [4] Ethereum Github, 2022 *Altair Light Client – Light Client* <https://github.com/ethereum/consensus-specs/blob/dev/specs/altair/light-client/light-client.md>
- [5] Ethereum Github, 2022 *Altair Light Client – Sync Protocol* <https://github.com/ethereum/consensus-specs/blob/dev/specs/altair/light-client/sync-protocol.md>
- [6] Vitalik Buterin and Virgil Griffith, 2019 *Casper the Friendly Finality Gadget* <https://arxiv.org/pdf/1710.09437.pdf>
- [7] Ethereum Org, 2022 *Blocks* <https://ethereum.org/en/developers/docs/blocks/>
- [8] Go Ethereum, 2022 *Geth Documentation* <https://geth.ethereum.org/docs/>
- [9] Ledger Watch, 2022 *Erigon, State Storage* <https://github.com/ledgerwatch/erigon#more-efficient-state-storage>
- [10] Prysmatic Labs, 2022 *Prysm: An Ethereum Consensus Implementation Written in Go* <https://github.com/prysmaticlabs/prysm>
- [11] Ethereum Wiki, 2022 *Dagger Hashimoto* <https://github.com/ethereum/wiki/wiki/Dagger-Hashimoto>
- [12] Ethereum Org, 2022 *ETHASH* <https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/mining-algorithms/ethash>
- [13] Harmony Protocol, 2022 *Github: Horizon v2* <https://github.com/johnwhitton/horizon/tree/refactorV2/docs>
- [14] Nomad XYZ, 2022 *Github: Nomad monorepo* <https://github.com/nomad-xyz/monorepo>
- [15] Map Protocol, 2022 *Github: Map Protocol* <https://github.com/mapprotocol>
- [16] Map Protocol, Atlas Chain 2022 *Github: Map Protocol Atlas Chain* <https://github.com/mapprotocol/atlas>

- [17] Wormhole Foundation, 2022 *Github: Wormhole* <https://github.com/wormhole-foundation/wormhole>

Bridge Hacks

- [1] Chain Analysis Team, 2022 *Vulnerabilities in Cross-chain Bridge Protocols Emerge as Top Security Risk* <https://blog.chainalysis.com/reports/cross-chain-bridge-hacks-2022/>
- [2] Rob Behnke, 2022 *EXPLAINED: THE QUBIT HACK (JANUARY 2022)* <https://halborn.com/explained-the-qubit-hack-january-2022/>
- [3] Rob Behnke, 2022 *EXPLAINED: THE WORMHOLE HACK (FEBRUARY 2022)* <https://halborn.com/explained-the-wormhole-hack-february-2022/>
- [4] Rob Behnke, 2022 *EXPLAINED: THE RONIN HACK (MARCH 2022)* <https://halborn.com/explained-the-ronin-hack-march-2022/>
- [5] Rob Behnke, 2022 *EXPLAINED: THE HARMONY HORIZON BRIDGE HACK* <https://halborn.com/explained-the-harmony-horizon-bridge-hack/>
- [6] Rob Behnke, 2022 *THE NOMAD BRIDGE HACK: A DEEPER DIVE* <https://halborn.com/the-nomad-bridge-hack-a-deeper-dive/>