Hyper Oracle

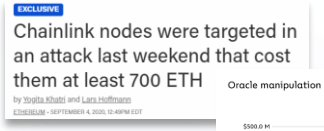
Secure and Trustless Oracle via ZK

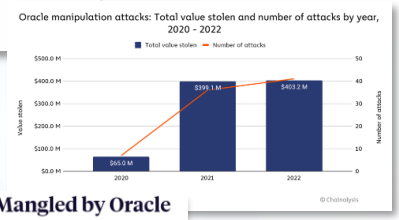
#10 Solidity

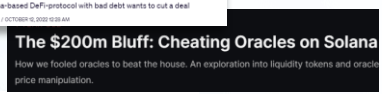
Suede @ Hyper Oracle

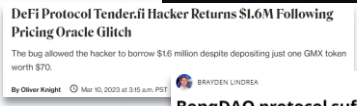
HYPER ORACLE

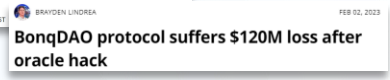
History of blockchain oracle







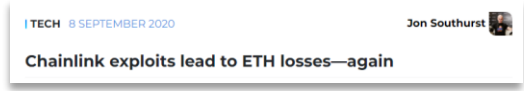




2019 – Chainlink mainnet launch

2020-2022 - Plethora of new staking-based oracle projects

>> Oracle vulnerabilities and manipulation attacks



2023 – Hyper Oracle

>> zkOracle secures against attacks in full decentralization

2024 – Hyper Oracle mainnet

>> Powering zkDApps

HYPER ORACLE

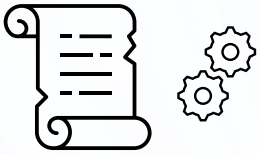
How staking & slashing based oracle works

Security and performance at risk

	Staking-based oracle	zkOracle
Security	Secured by tokenomics	Secured by cryptography
Finality (Performance)	1-2 min	block time (~12s for ETH)
Decentralization	Choke point, centralized	Proof of oracle work
Censorship-Resistance	Minimum (~cloud service)	Maximum

dApp =

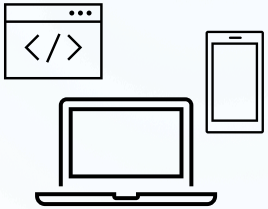
"Backend" coding



With Smart Contract

+

Frontend coding



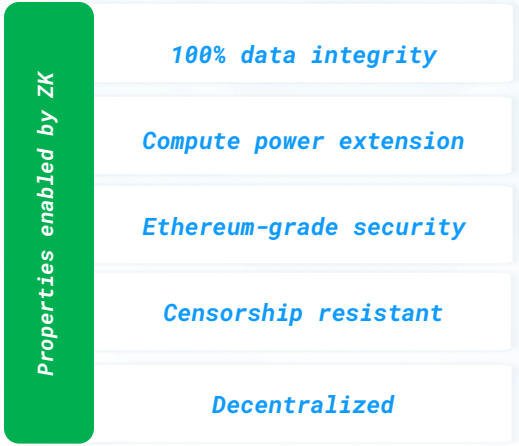
With React, Vue, etc.

Smart contract cannot support complex compute

zkOracle enriches smart contract w/ more computation & more data sources.

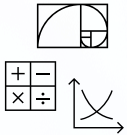
What do ZKPs do?

- Attestation
- Compress compute
- Trust minimization
- Information shielding



zkDApp =

Compute extension

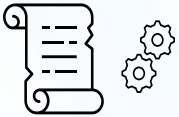


With ZK circuits

Study cryptography & DSL

+

"Backend" coding



With Smart Contract

Learn Solidity

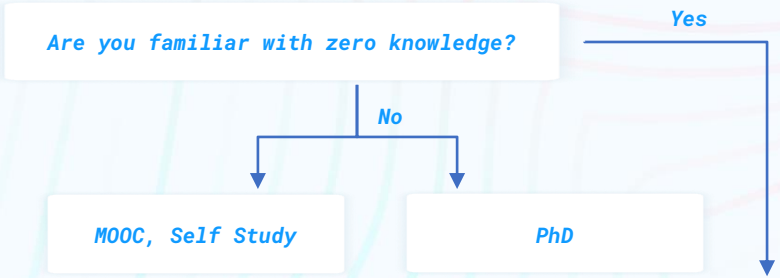
+

Frontend coding

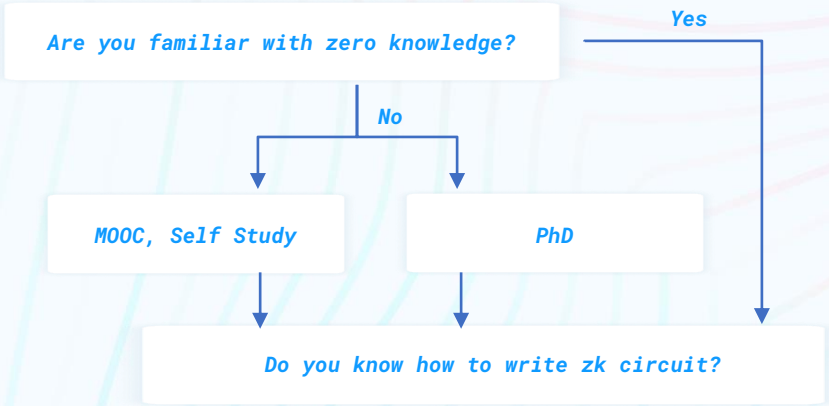
With React, Vue, etc.

Develop as usual

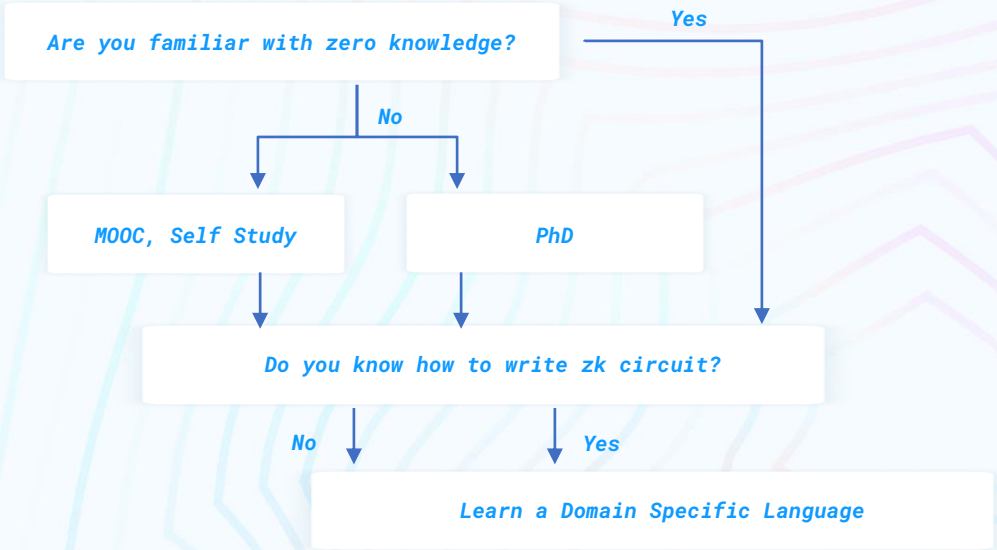
Building a zkDApp yesterday



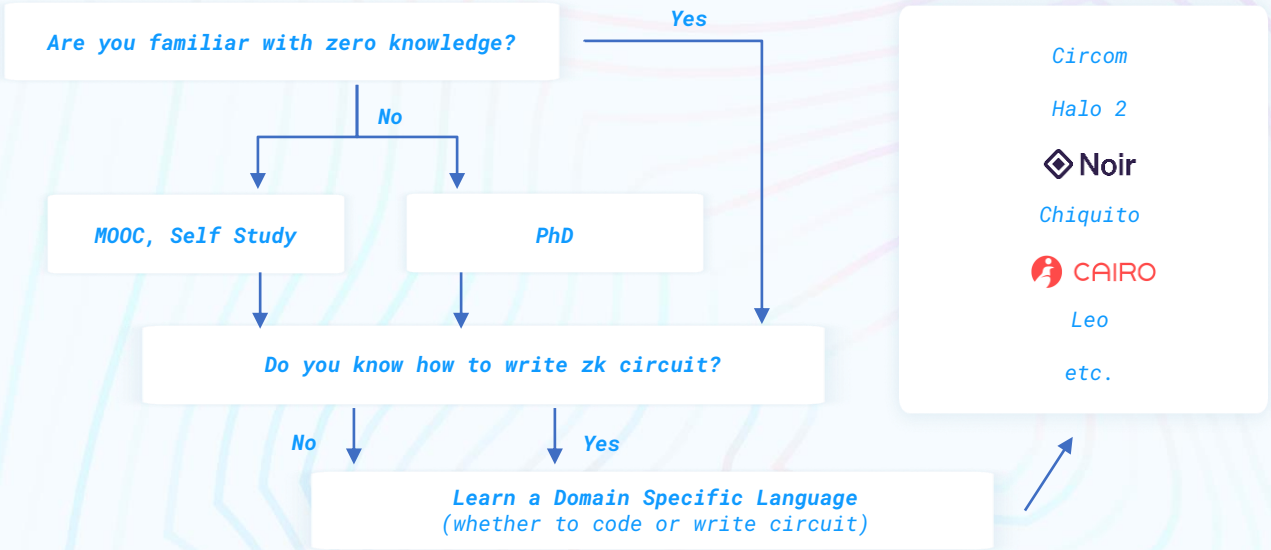
Building a zkDApp yesterday



Building a zkDApp yesterday



Building a zkDApp yesterday



LOOOOOONG learning process in ZK

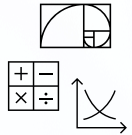
ZK knowledge	3 months – 7 years
ZK circuit / DSL	2 weeks – 2 months
Debug	3 – 6 months

Compute extension

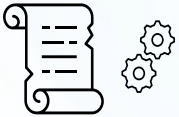
“Backend” coding

Frontend coding

zkDApp =



+

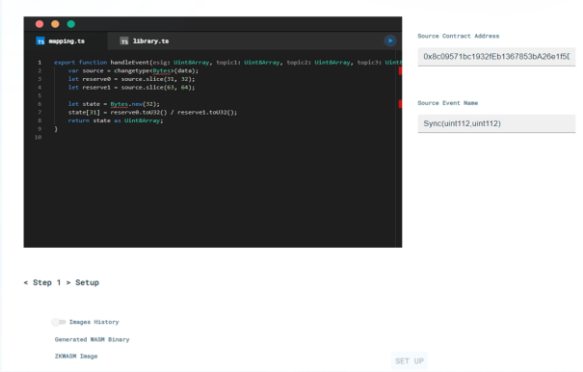


+

zkDApp = zkOracle + dApp

Simplest way to build zkDApps

With zkGraph Studio, no ZK needed



1. *Web IDE, fast onboarding*

2. *Write code in AssemblyScript, not circuits*

3. *Reap ZK benefits today*

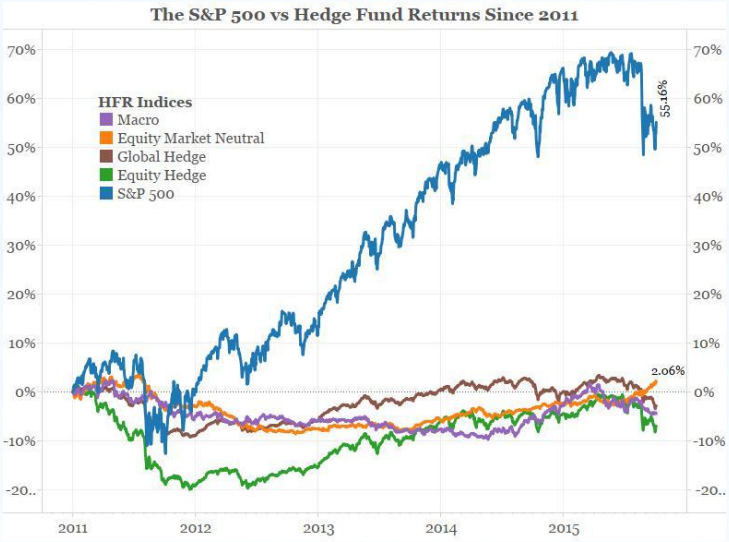
https://www.hyperoracle.io/app/zkGraphStudio | Demo video for creating zkGraph template: https://www.youtube.com/watch?v=peF8AlxIIxA

HYPER ORACLE

Powering a new wave of zkDApps.

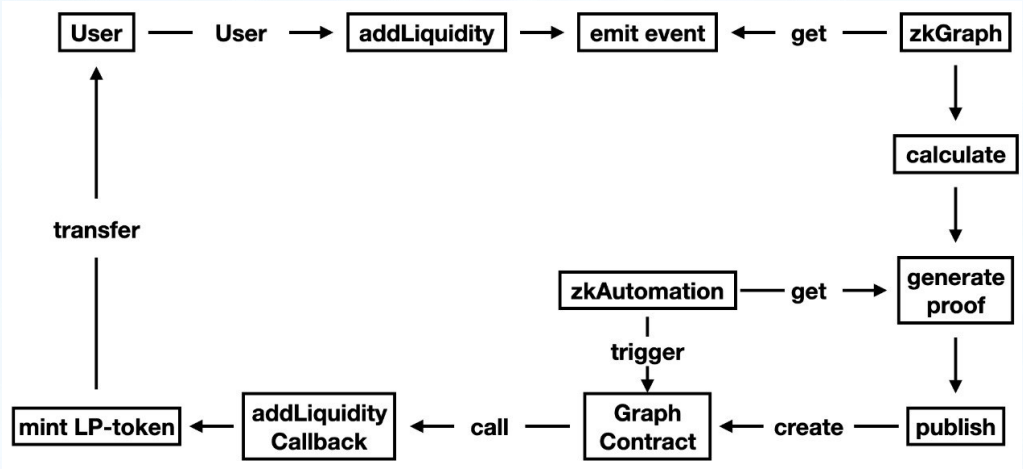
Novel. Secure. Permissionless.
Trustless. Unstoppable. Build in hours.

Better return, anyone?



HYPER ORACLE

Decentralized ETF



HYPER ORACLE

Decentralized ETF destination contract

```
contract dETF is ERC20 {
    address public owner;
    address public graphContract;
    address public demoVaultContract;

    address public token1;
    address public token2;

    uint256 public position1;
    uint256 public position2;

    uint256 public curr_blockNumber;

    // for ratio calculation
    uint256 immutable public DECIMALBASE = 1e18;
    uint256 MAX_INT = ~uint256(0);

    event investSuc(address indexed user, uint256 indexed etfTokenAmount);
    event redeemSuc(address indexed user, uint256 indexed etfTokenAmount);
    event reBalanceSuc(uint256 indexed position1, uint256 indexed position2, uint256 indexed curr_blockNumber);
```

```
function rebalance(uint256 price) public onlyGraph {
    uint256 currTokenRatio = IERC20(token1).balanceOf(address(this)) * DECIMALBASE / IERC20(token2).balanceOf(address(this));

    uint256 etfTotalSupply = totalSupply();
    uint256 halfAmountToSwap;

    // Compare current ratio to the provided ratio to see if the difference is greater than the threshold
    if (currTokenRatio > price) {
        // Calculate the amount of token1 to be swapped to token2
        halfAmountToSwap = ((currTokenRatio - price) * position2 * etfTotalSupply) / (2 * DECIMALBASE);

        // Ensure there is enough token1 for the swap
        require(IERC20(token1).balanceOf(address(this)) >= halfAmountToSwap, "Insufficient token1 balance for swap");

        // Perform the swap using DemoVault
        IDemoVault(demoVaultContract).swapToken1(halfAmountToSwap, halfAmountToSwap * price / DECIMALBASE);
    } else if (currTokenRatio < price) {
        // Calculate the amount of token2 to be swapped to token1
        halfAmountToSwap = ((price - currTokenRatio) * position1 * etfTotalSupply) / (2 * DECIMALBASE);

        // Ensure there is enough token2 for the swap
        require(IERC20(token2).balanceOf(address(this)) >= halfAmountToSwap, "Insufficient token2 balance for swap");

        // Perform the swap using DemoVault
        IDemoVault(demoVaultContract).swapToken2(halfAmountToSwap, halfAmountToSwap * DECIMALBASE / price);

    }
    position1 = IERC20(token1).balanceOf(address(this)) * DECIMALBASE / etfTotalSupply;
    position2 = IERC20(token2).balanceOf(address(this)) * DECIMALBASE / etfTotalSupply;
    curr_blockNumber = block.number;

    emit reBalanceSuc(position1, position2, curr_blockNumber);
```

Decentralized ETF verifier contract

```
function verify(
    uint256 blockNumber,
    bytes32 blockHash,
    bytes memory zkgState,
    uint256[] calldata proof,
    uint256[] calldata verify_instance,
    uint256[] calldata aux
) public view returns (bool) {
    // require(blockhash(blockNumber) == blockHash, "Invalid public input blockhash");
    uint256[] memory encodedPub = encodePublicInput(blockNumber, blockHash, zkgState);
    uint256[][] memory target_instance = new uint256[][](1);
    target_instance[0] = encodedPub;

    IZKVerify(verifier).verify(proof, verify_instance, aux, target_instance); // revert if failed
    return true;
}
```

zkGraph for implementing strategy

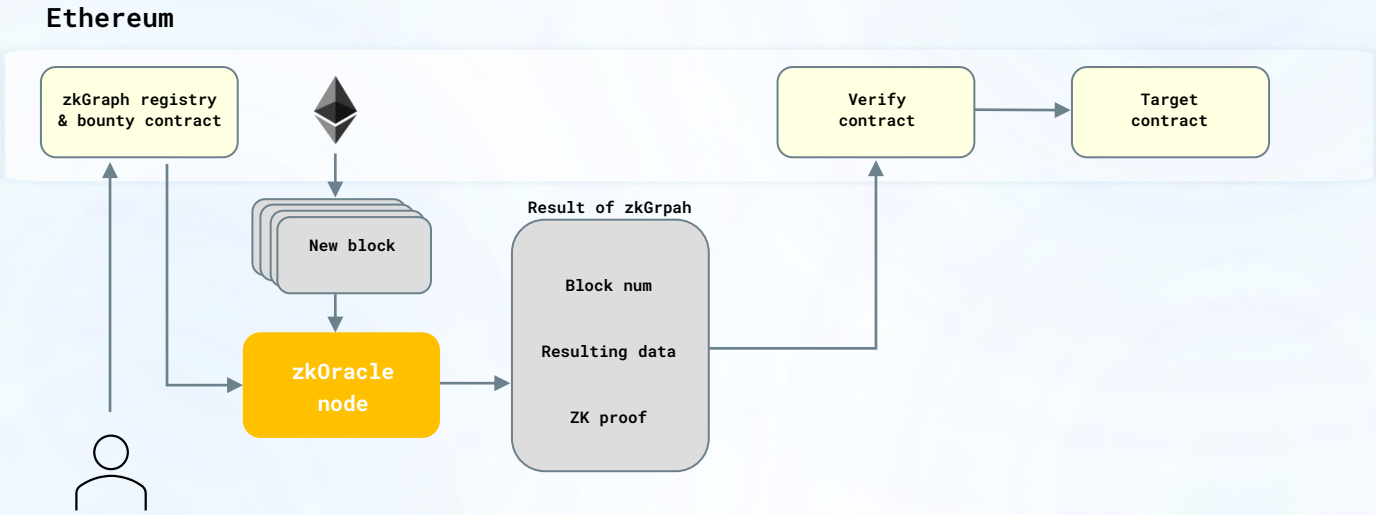
```
export function handleEvents(events: Event[]): Bytes {
  let lastSyncEvent: Event | null = null;

  for (let i = events.length - 1; i >= 0; i--) {
    if (events[i].esig == esig_sync) {
      //   console.log('SYNC event');
      lastSyncEvent = events[i];
      break;
    }
  }

  if (lastSyncEvent == null) {
    // Don't Trigger if there's no event in the block
    require(false);
    return Bytes.empty(); // Omit compile error, never goes here
  } else {
    let ratio = calcRatio(lastSyncEvent);

    // rebalance(uint256)
    //f4993018cf1db379be1053b15816b2c65cb6d0fbf9e77cd3eeba21dd0e135cb5
    let function_selector = Bytes.fromHexString("6ea30ce9");
    // Set payload to the current price0 when triggering destination contract.
    // 32 bytes function selector + 28 bytes ratio
    // 4 bytes selector || 28 bytes parameter. 00000000000000000012345
    let payload = Bytes.fromByteArray(function_selector.concat(Bytes.fromHexString(ratio.toString(16)).padStart(28, 0)));
    return payload;
  }
}
```

Tamper-proof, on-chain verify



HYPER ORACLE

Build with secure & trustless zkOracle

- Print all target events of different contracts
- Opensea order fulfillment amount
- USDT total transfer amount
- Uniswap trade price

Build today w/ zkGraph Studio

Sample use case

Thank You!

Discord

Github

github.com/HyperOracle

Twitter

@HyperOracle

Web

HyperOracle.io