Sapphire 101

Developer Office Hours

OASIS NETWORK

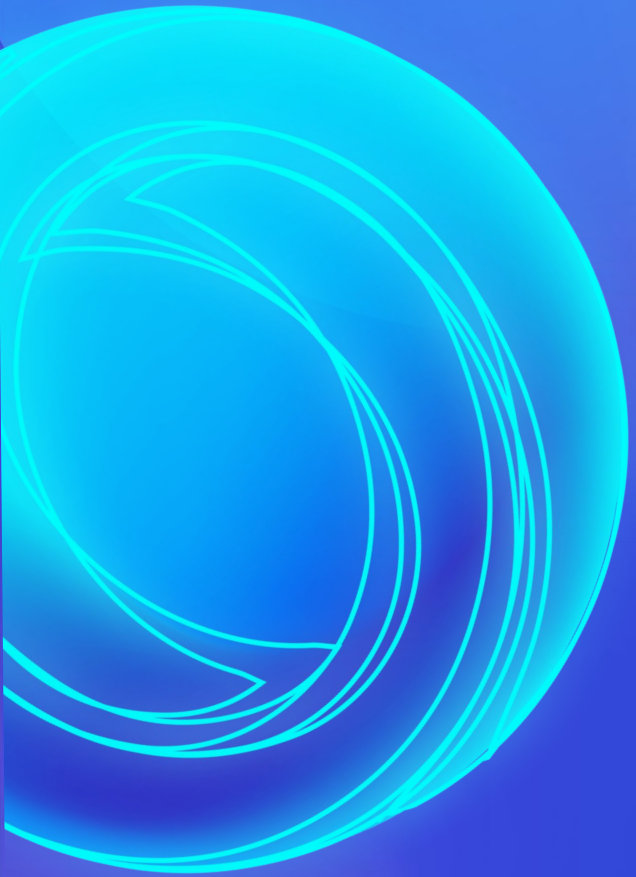
Who am I?

Matevž Jekovec
Software Engineer
at Oasis Protocol Foundation

Working (mainly) on docs.oasis.io, Oasis CLI, tooling, Oasis Core and ParaTimes tests, examples and other learning material.

Overview

1. The Oasis Architecture
2. Task 1: Confidential transactions
3. Task 2: Signed view calls
4. Task 3: The Frontend
5. Task 4: Precompiles
6. What to do next



Prerequisites

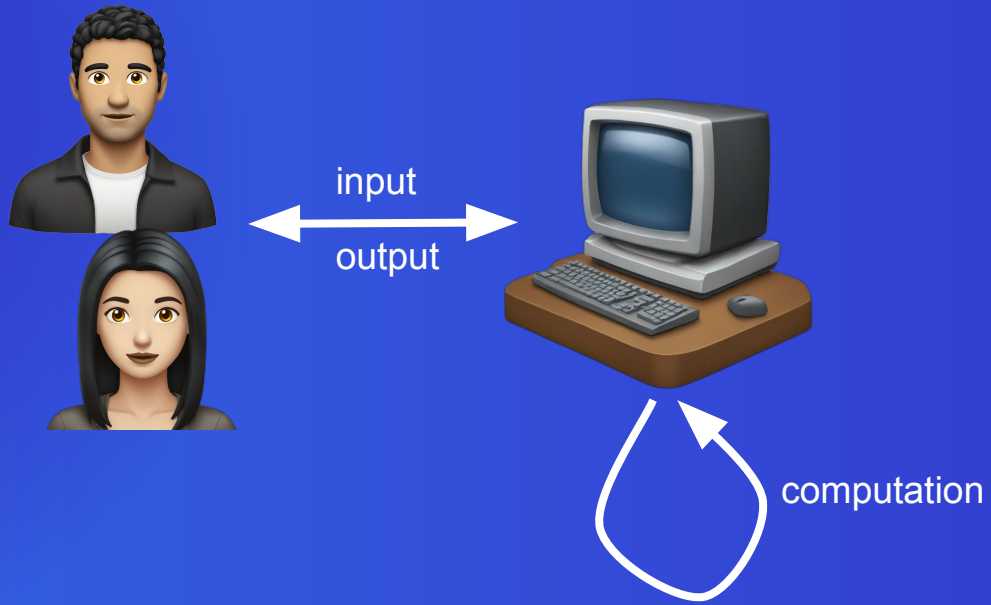
- Basic programming (e.g. JavaScript will come handy)
- Basic knowledge of the command line
- A working Node.js environment + pnpm

The Oasis Architecture

OASIS NETWORK

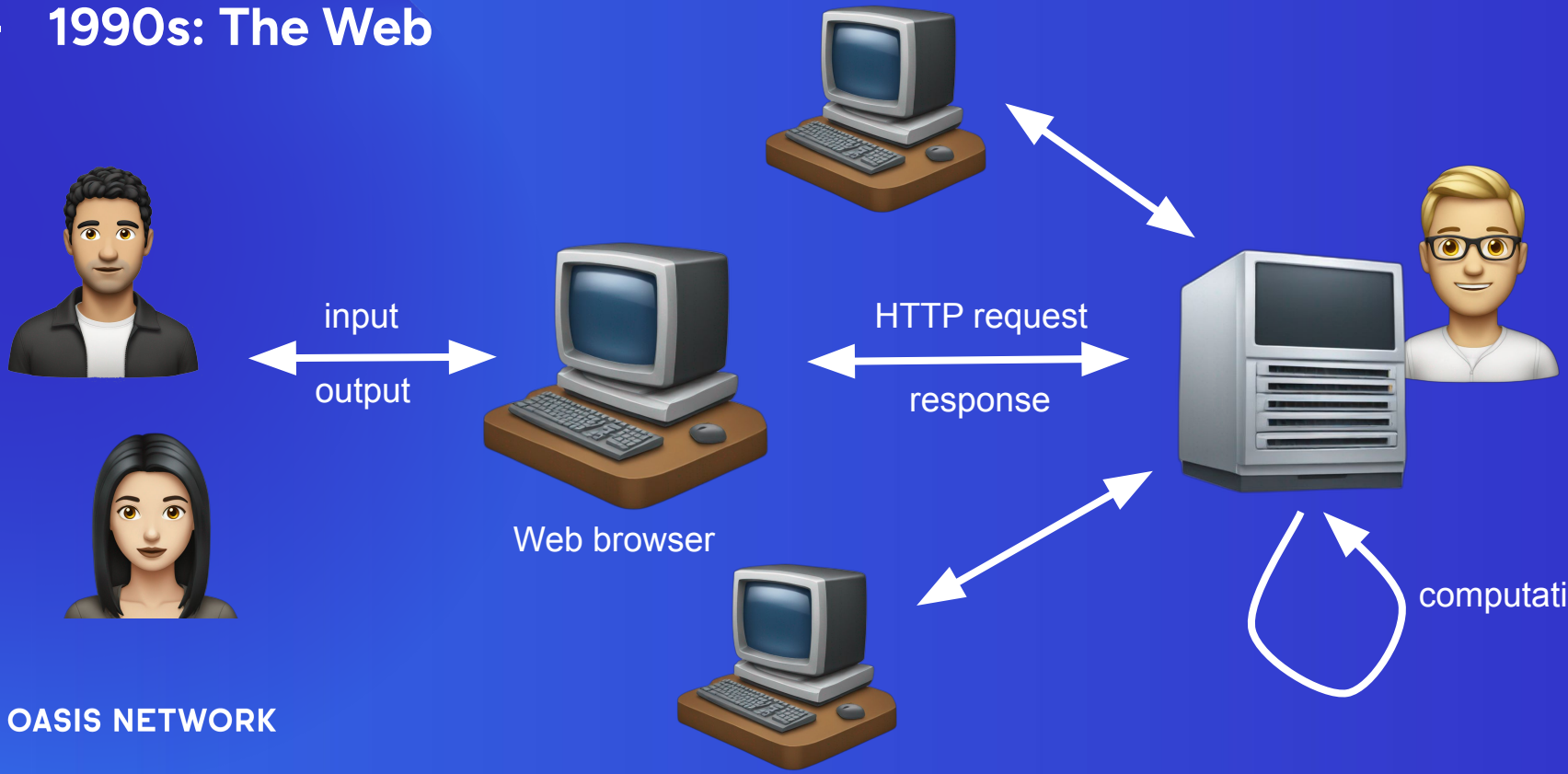
Short evolution of computing for average Alice and Bob

- Before the 90s



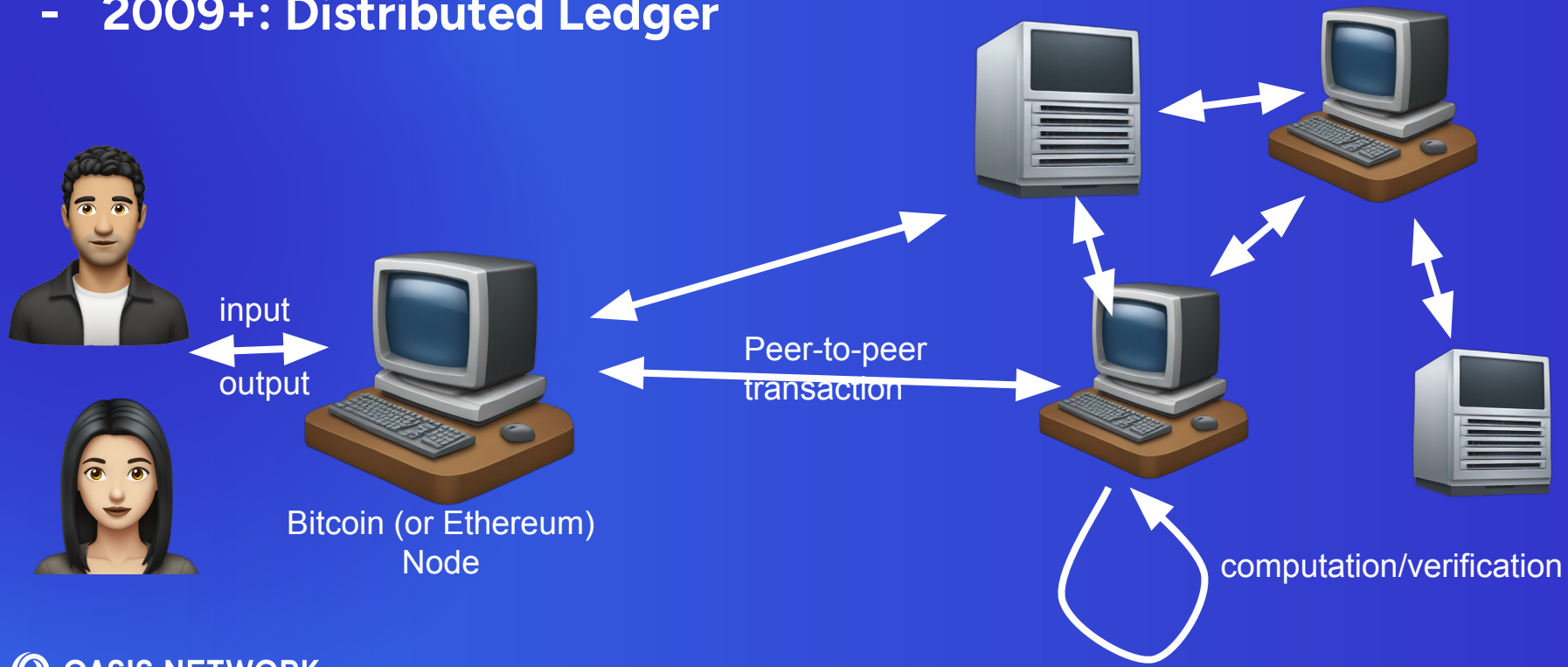
Short evolution of computing for average Alice and Bob

- 1990s: The Web

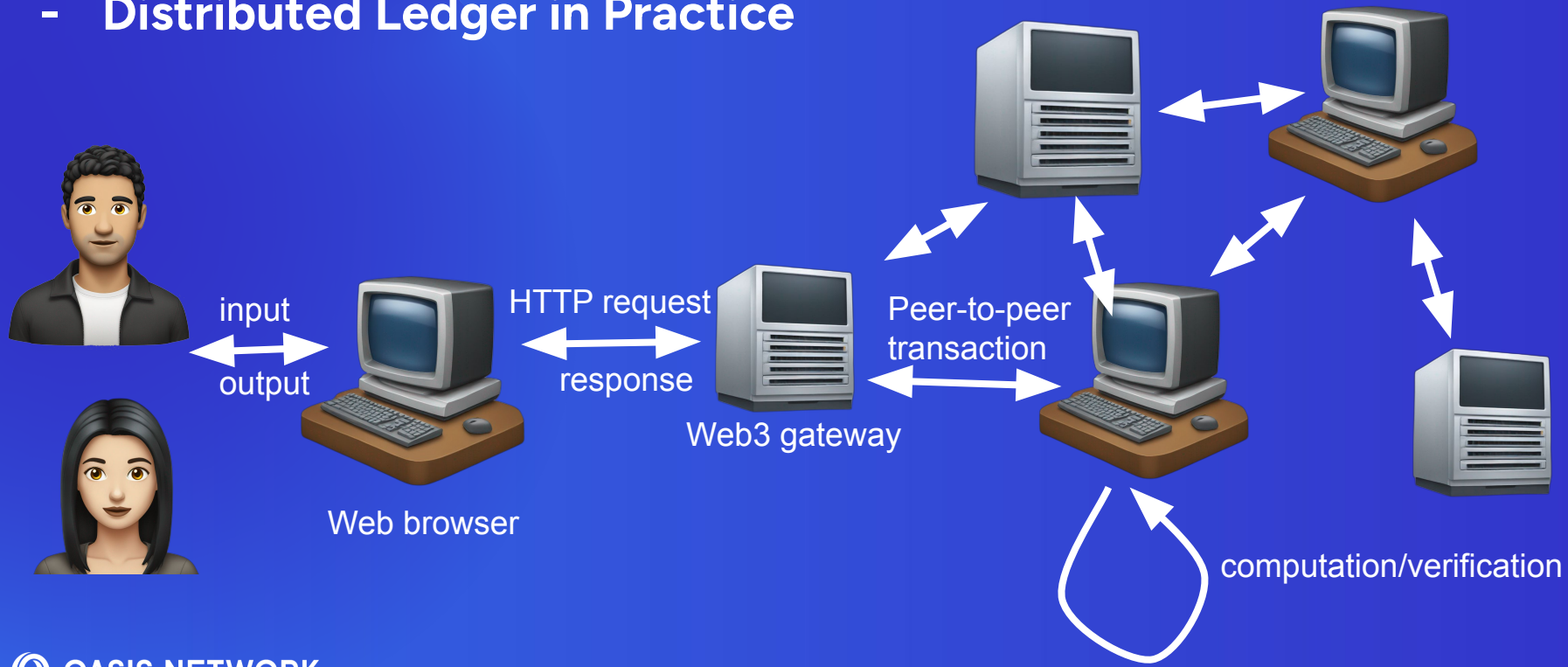


Short evolution of computing for average Alice and Bob

- 2009+: Distributed Ledger



Short evolution of computing for average Alice and Bob - Distributed Ledger in Practice



OASIS NETWORK

Ethereum's Virtual Machine (EVM): Storage and Computation

- How is the contract state stored on Ethereum chain?
  - World state trie
- We can **extract all contract on-chain data**, via `eth_getStorageAt`
- What if we encrypt data?
  - DeFi, MEV, data privacy! ❤️
- But how can nodes *compute* in a *distributed environment*, if the storage is encrypted?
  - Trusted Execution Environment!

What should Confidential EVM Look Like?

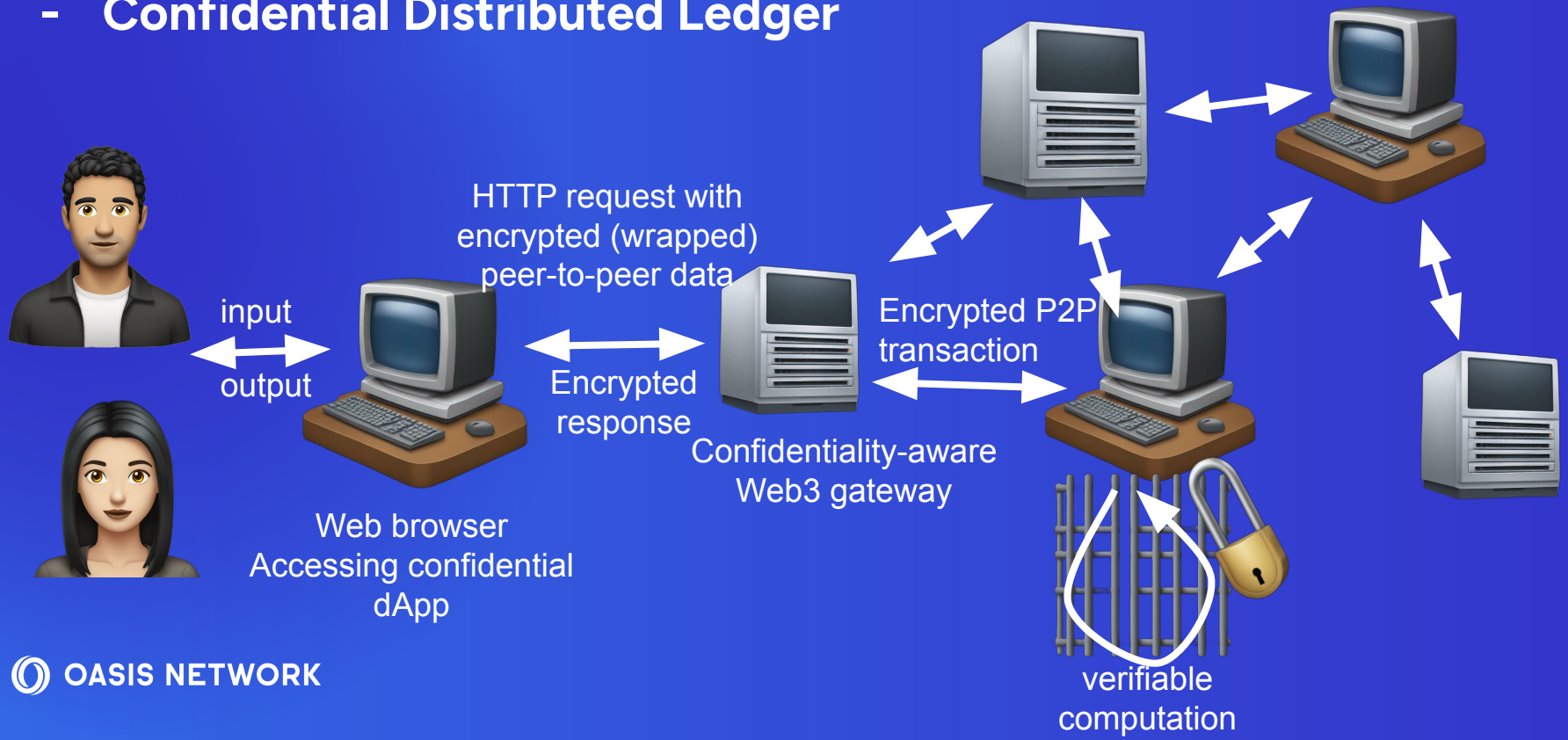
Private Compute

Randomness

Private Storage

End to End Encryption

Short evolution of computing for average Alice and Bob
- Confidential Distributed Ledger



OASIS NETWORK

Sapphire

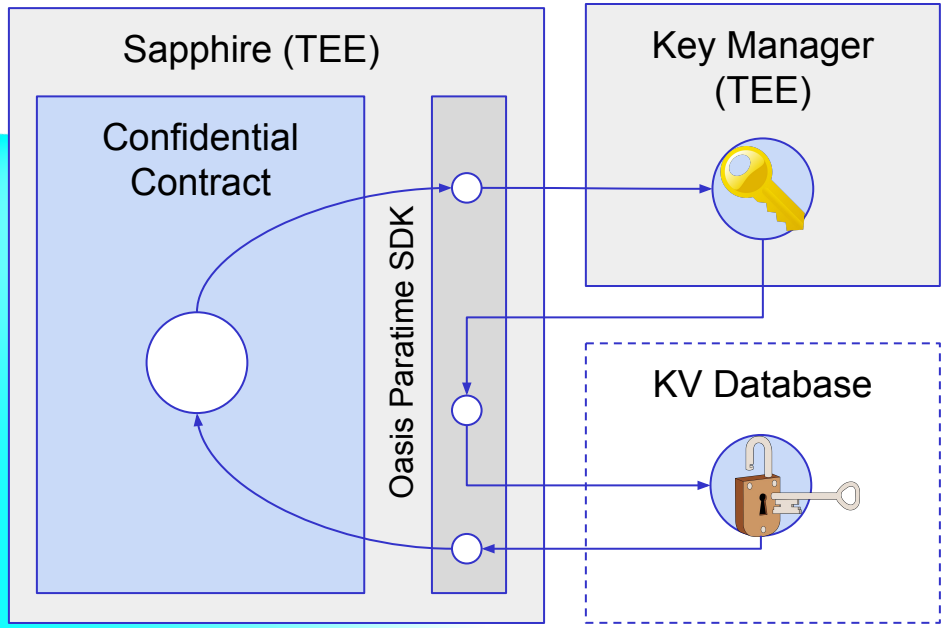
Confidential EVM

OASIS NETWORK

The Oasis Architecture: Sapphire

- Sapphire is a **confidential EVM-compatible** chain running on Oasis network.
- Smart Contracts are executed in **Trusted Execution Environment (TEE)** on a distributed network of compute nodes.
- Smart Contract storage **is encrypted**.
- Sapphire supports **backward-compatible unencrypted and Oasis-wrapped encrypted transactions and calls**.

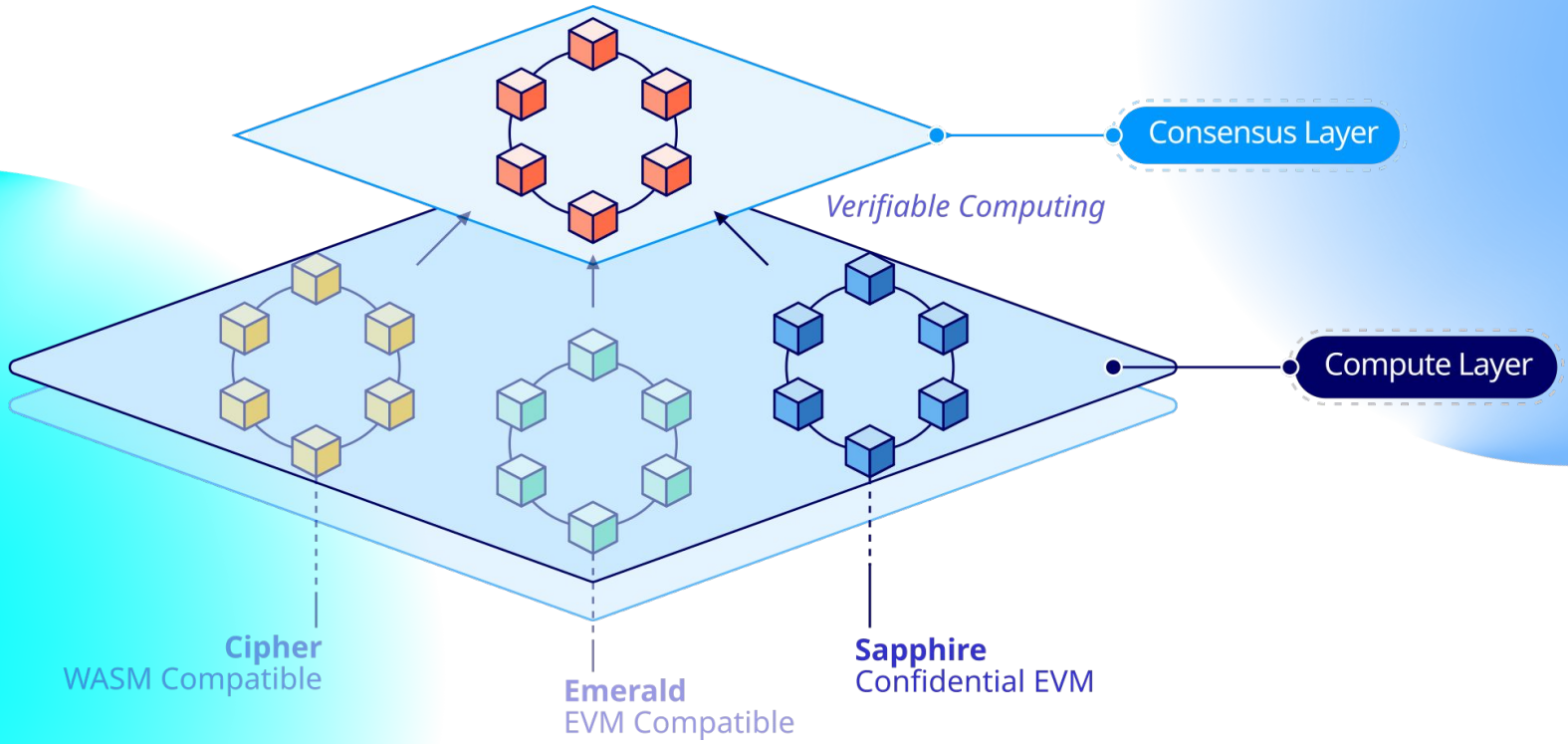
The Oasis Architecture: Sapphire



- Built with **Oasis Paratime SDK**.
- Each contract is **encrypted using a different key**.
- **Isolated Key Manager** improves security by segregating key access.
- The key-value database stores **encrypted data**.

Note: Access patterns are **still visible to node operators**!

The Oasis Architecture: Bigger Picture



OASIS NETWORK

The Oasis Architecture: Sapphire Public Endpoints

Mainnet

- RPC HTTP endpoint: https://sapphire.oasis.io
- RPC WebSockets endpoint: wss://sapphire.oasis.io/ws
- Chain ID:
  - Hex: `0x5afe`
  - Decimal: `23294`
- Block explorer: https://explorer.oasis.io/mainnet/sapphire

Testnet

- RPC HTTP endpoint: https://testnet.sapphire.oasis.dev
- RPC WebSockets endpoint: wss://testnet.sapphire.oasis.dev/ws
- Chain ID:
  - Hex: `0x5aff`
  - Decimal: `23295`
- Block explorer: https://explorer.oasis.io/testnet/sapphire

Let’s get started!

github.com/oasisprotocol/demo-starter

Task 1

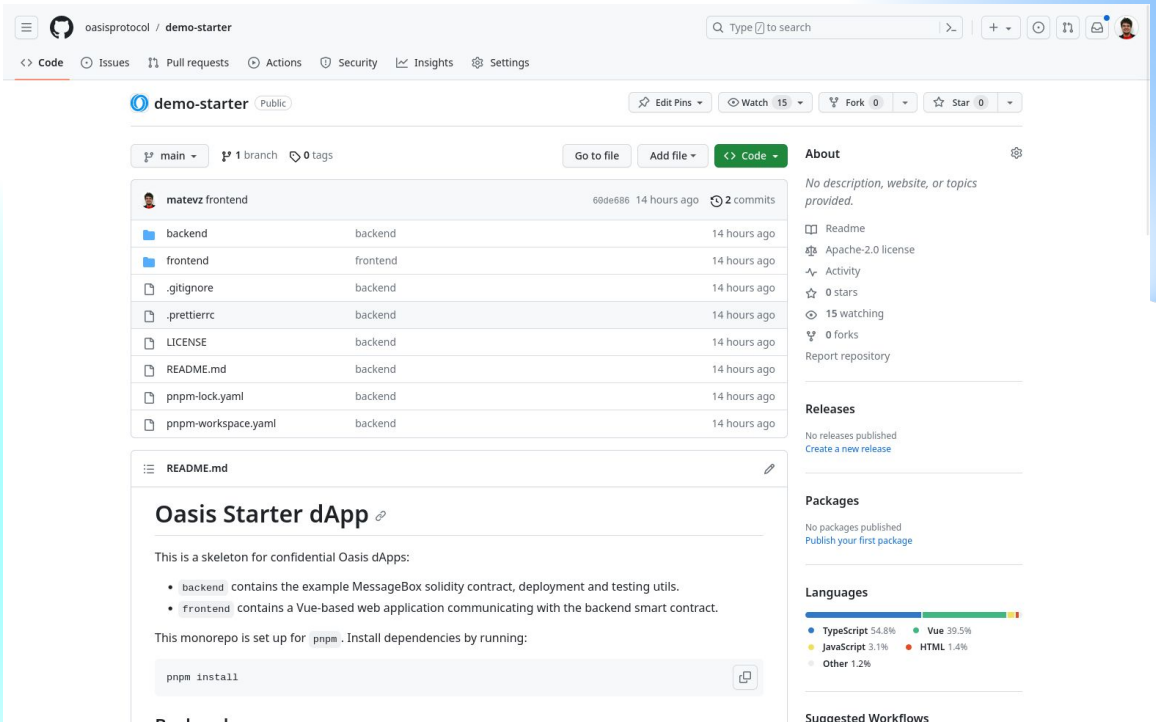
Confidential Transactions

OASIS NETWORK

github.com/oasisprotocol/demo-starter

Task 1: Getting Started

1. Clone `github.com/oasisprotocol/demo-starter`

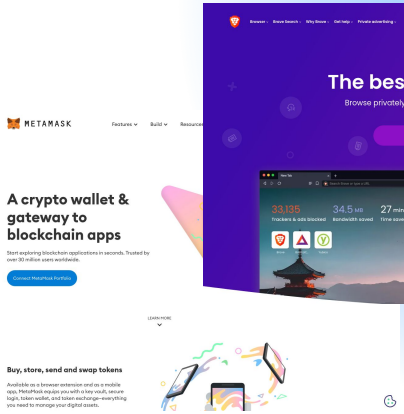


OASIS NETWORK

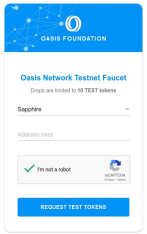
github.com/oasisprotocol/demo-starter

Task 1: Getting Started

2. Generate an Ethereum-compatible wallet (MetaMask, Brave...)



3. Get some Sapphire TEST tokens for your account at faucet.testnet.oasis.dev



OASIS NETWORK

github.com/oasisprotocol/demo-starter

Task 1: Getting Started

4. Deploy contract to Sapphire Testnet.

```
cd backend; pnpm install; pnpm build
PRIVATE_KEY=0x… npx hardhat deploy --network sapphire-testnet
```

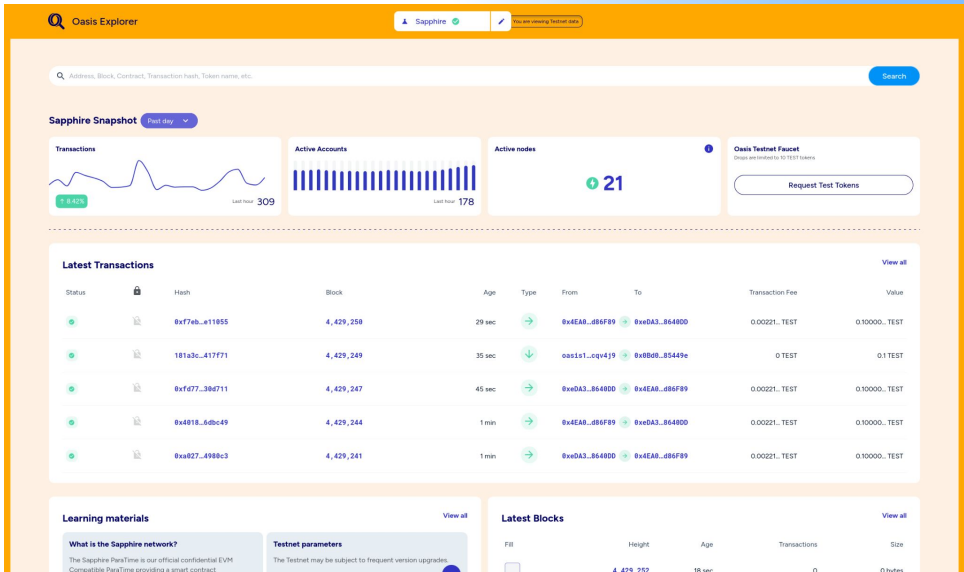
5. Browse Oasis Explorer and try to find your contract:

   explorer.oasis.io

6. Notice:

   Encrypted transactions: 🔒

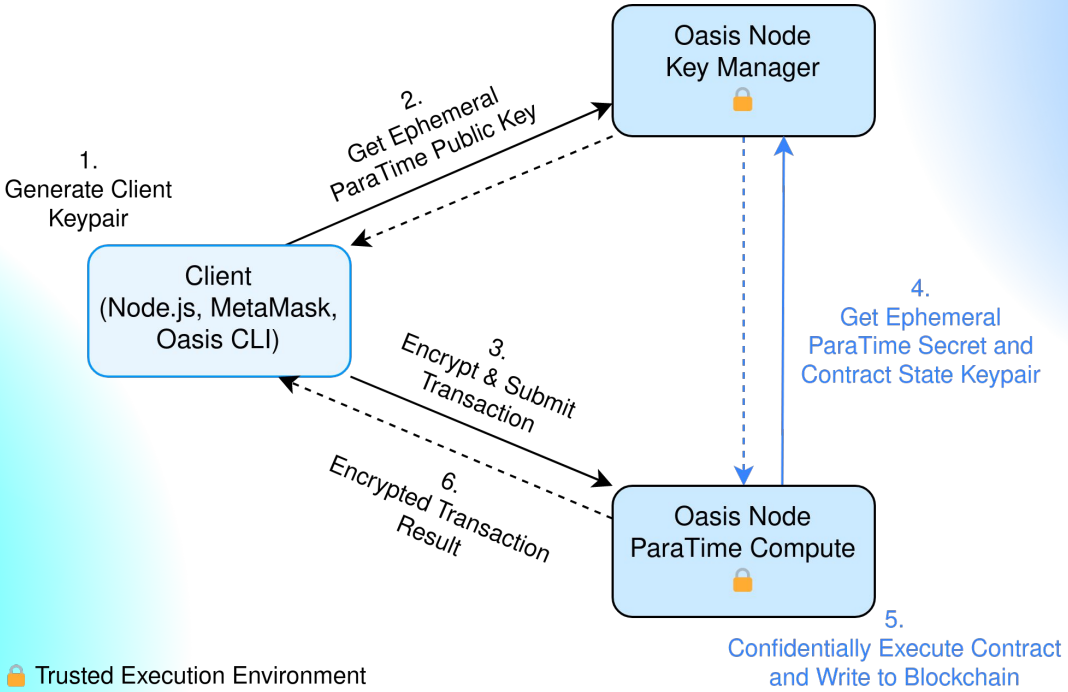
   Unencrypted transactions: 🔓



OASIS NETWORK

github.com/oasisprotocol/demo-starter

Task 1: Confidential Transactions



Task 1: Confidential Transactions

- To enable encrypted transactions **wrap the `ethers` signer**.
- Hardhat: Add at the top of your `hardhat.config.ts`:

```
import '@oasisprotocol/sapphire-hardhat';
const signer = ethers.provider.getSigner(); // signer is automatically wrapped
```

- In other clients (e.g. browser):

```
import * as sapphire from '@oasisprotocol/sapphire-paratime';
const signer = sapphire.wrap(ethers.provider.getSigner());
```


OASIS NETWORK

Task 1: Confidential Transactions

7. Demo starter project already imports the wrapper. Check out how the `setMessage` and `message` Hardhat tasks store and retrieve the message with **encrypted transaction** and **encrypted call**.

```
PRIVATE_KEY=0x… npx hardhat setMessage <CONTRACT_ADDRESS> <YOUR_MESSAGE> --network sapphire-testnet
PRIVATE_KEY=0x… npx hardhat message <CONTRACT_ADDRESS> --network sapphire-testnet
```


OASIS NETWORK

Task 1: Confidential Transactions

But, is it private? Kinda...

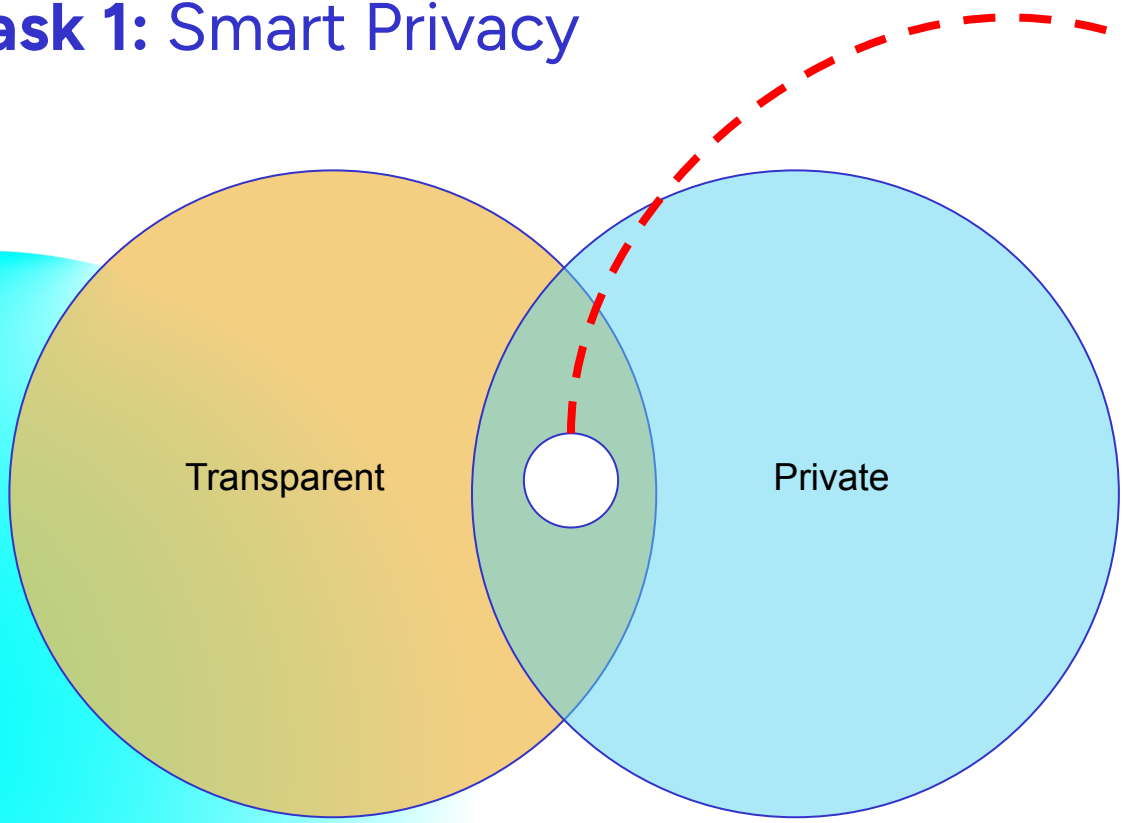
The public parameters for both plain and encrypted transactions are:

- From & To
- Gas Price, Gas Used
- Encrypted Calldata size

OASIS NETWORK

github.com/oasisprotocol/demo-starter

Task 1: Smart Privacy



99% of the world

Business
Regulations
etc.

Task 1: Smart Privacy

8. Try to extend `setMessage` to support **unencrypted transactions** as well. Use Explorer to check the transmitted transaction.

   HINT: Check how `deploy` task was implemented.

OASIS NETWORK

Task 2

Signed View Calls

OASIS NETWORK

Task 2: Impersonation

- When executing a transaction, `msg.sender` contains the address of the caller—the **one who signed the transaction** or the **contract address** of the caller function.
- For **view calls**, Ethereum supports **impersonation**. This means you can set the value of `msg.sender` field to arbitrary value (e.g. some third-party address) and perform read-only queries on behalf of someone else.
- This is perfectly fine in Ethereum, because the contract state is public anyway and anyone can simulate any execution path.

OASIS NETWORK

Task 2: Impersonation

- Sapphire contract storage is **confidential**, it **forbids impersonation**:
  - Anonymous view call:

    `msg.sender` equals `address(0x0)`
  - Signed view call (aka. *signed query*):

    `msg.sender` equals `address(signer)`
- Developers should mitigate **memory access pattern attacks** by requiring specific `msg.sender`.

OASIS NETWORK

Task 2: View calls & Impersonation

1. Make the `message` field in the `MessageBox` contract private and only allow the author of the message to retrieve it.
2. Try out **unwrapped** signer to check whether the `msg.sender` is really zeroed, if view call is unsigned.

OASIS NETWORK

Task 3

The Frontend

OASIS NETWORK

Task 3: Frontend

- The `frontend` folder contains a simple Vue-based app which imports the MessageBox contract from backend.
- Don’t forget to build the backend first!

```
cd backend; pnpm build
```

- In Demo starter, we use Vite to reflect any changes on-the-fly and for deployment. Preview the frontend with:

```
cd frontend; pnpm dev
```

Task 3: Signing in the Frontend

- In Demo starter, we store the *MetaMask instance* into local store so that it's not reloaded each time you open a subpage.
- `useEthereumStore()` returns `{signer, unwrappedSigner, provider, unwrappedProvider}` to access all flavors of the signer or the provider respectively.
- Contract address, expected chain ID etc. is read from `frontend/.env*` files

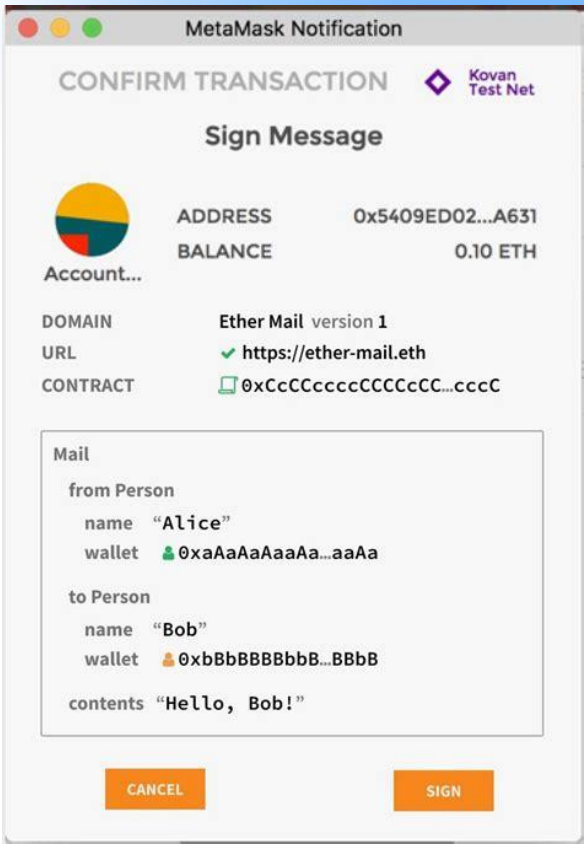
github.com/oasisprotocol/demo-starter

Task 3: Signing in the Frontend

- In `contracts.ts` the wrappers for the contracts are prepared for you:

```
const messageBox = useMessageBox();
const uwMessageBox = useUnwrappedMessageBox();
```

- Calling `uwMessageBox.message()` will execute **Anonymous Unencrypted View Call in background**.
- Calling `messageBox.message()` will activate MetaMask for **Signed and Encrypted View Call**.



Task 3: Signing in the Frontend

- Signing the same query **is cached**. (This helps to reduce annoying popups.)

TASK: Try out **setting the message in the browser** with MetaMask.

Task 4

Precompiles

OASIS NETWORK

Task 4: Precompiles

Precompiles for Solidity are built-in functions that **extend the EVM**.

These are complex operations which are difficult to implement efficiently.

Examples of these on *Ethereum* are:

- SHA256
- Elliptic Curve & Pairing (`ECADD`, `ECMUL`, `ECPAIRING`) for ZK / BN128 curve
- EcDSA Signature Verification (`ecrecover`)

Task 4: Precompiles in Sapphire

```
pnpm install -D @oasisprotocol/sapphire-contracts
```

```
import "@oasisprotocol/sapphire-contracts/contracts/Sapphire.sol";
```

- (Incomplete) list of precompiles:
  - Gas padding
  - Random Number Generation (VRF / CSPRNG)
  - Encryption & decryption (X25519), key generation (Ed25519, SEC P256 k1), signing & verification (EdDSA, EcDSA)
- Learn more: docs.oasis.io/dapp/sapphire/precompiles

Task 4: Sapphire-Localnet

- Sapphire precompiles run on Sapphire only.
- We can use Testnet, but for running the (automated CI) tests each time, this is not viable (slow, gas costs).
- Inspired by `npx hardhat node` and `geth --dev`, the Oasis team provides the `sapphire-localnet` docker image which spins up an isolated Oasis network locally:

```
docker run --rm -it -p8545:8545 -p8546:8546 ghcr.io/oasisprotocol/sapphire-localnet
```

Task 4: Random Number Generator

- Cryptographically secure source of entropy:

```
bytes memory entropy = Sapphire.randomBytes(32, "");
```

- Much easier than using an **external** VRF, it's instant!
- The second parameter seeds the RNG with your own "extra entropy".

TASK: Implement on-chain CAPTCHA (e.g. add two numbers) to prevent spam attacks.

What to do next?

OASIS NETWORK

What to do next? ECDH (Shared Secrets)

- Two people have their own key pairs, say Alice and Bob
- When they have each others public keys they can both derive the point, and nobody else can

  (Unless one of the secrets gets leaked)

$$g^{xa} = A^x = X^a \ : \ A = g^a \ : \ X = g^x$$

What to do next? Encryption & Decryption

1. Our initial example used **end-to-end encryption** between user & smart-contract (and stored on the public ledger in encrypted form).
2. But, we can use X25519 key pairs **inside a smart contract** and derive shared secret on-chain!

Use Cases:

- Encrypted emitted events, stateless view calls.
- Encrypted coupons, proxies, authentication, compliance.

What to do next? Signing & Verification

- Contracts can generate key-pairs (Ed25519, X25519, Bitcoin & NIST)
- Signing with EcDSA and EdDSA

But, what can I do with that?

- Generate Ethereum addresses and sign transactions **on-chain**!
- Verify WebAuthN / TouchID, using Ed25519 & EdDSA
- Broadcast cross-chain without bridges

What to do next? On-chain Single Sign-On

- Viewing Keys: Submit an access token to view your data.
- Permits: Provide access to specific functions (read/write).
- Do we really need so many writes?
- Daily Login with EIP-712:
  - Similar to permits, but application specific.
  - Lets you sign a transaction once per day/week.

Learn more: docs.oasis.io/dapp/sapphire/authentication

Step forward—WebAuthN integration: playground.oasis.io/authzn

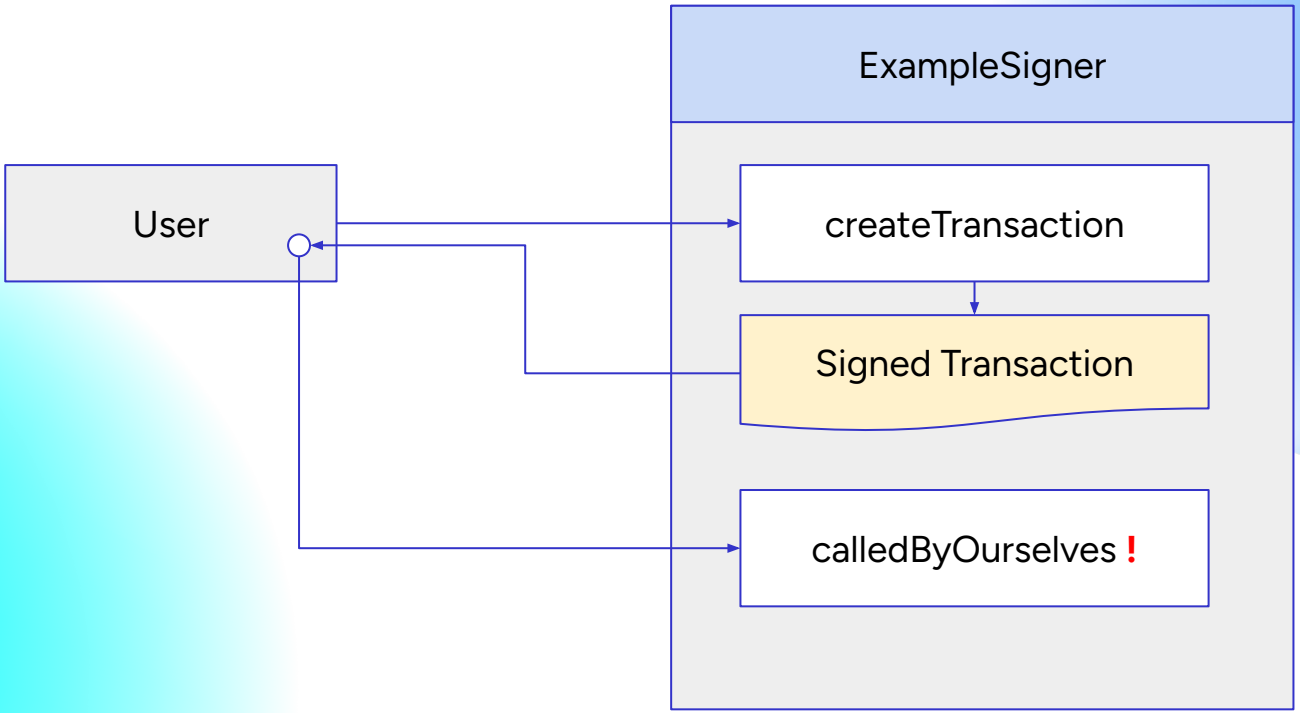
What to do next? Autonomous Contracts

- Using the key generation pre-compiles:

  Sapphire can generate **Ethereum compatible accounts** to **pay their own gas**—either on Sapphire or on other EVM compatible chains.
- Check out the Gasless chapter in the docs:

  docs.oasis.io/dapp/sapphire/gasless

What to do next? Autonomous Contracts

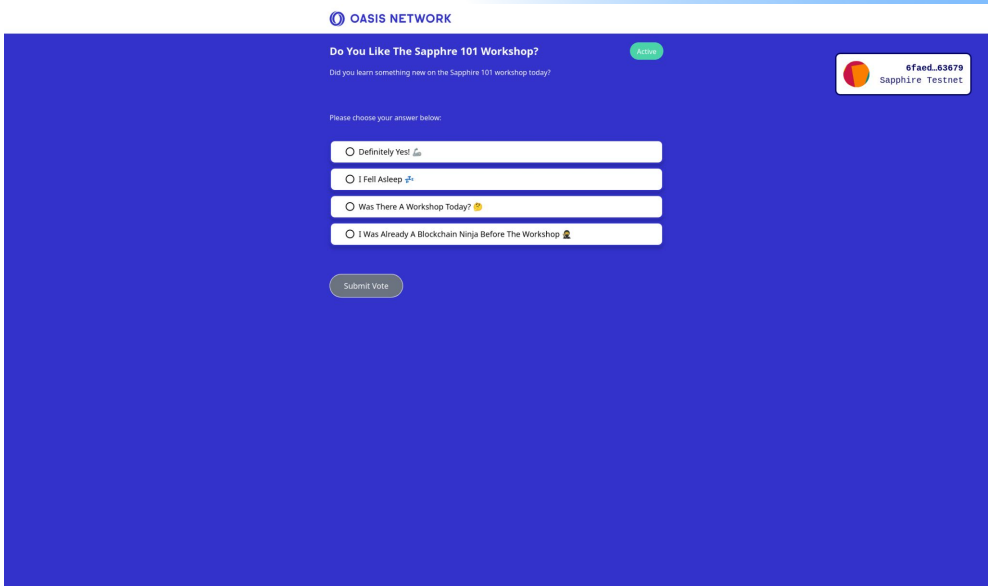


What to do next? Autonomous Contracts

Demo:

playground.oasis.io/demo-voting

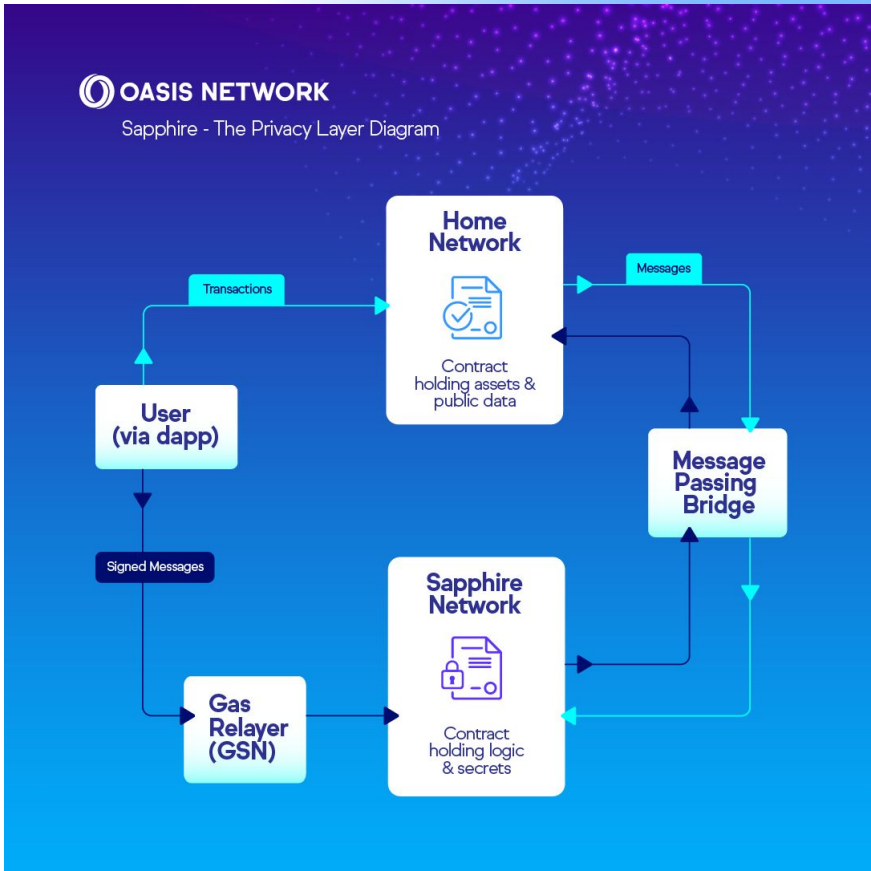
- Submit your votes privately.
- Use MetaMask to sign (EIP-712).
- No need for gas token.
- Only to be connected to Sapphire Testnet.



OASIS NETWORK

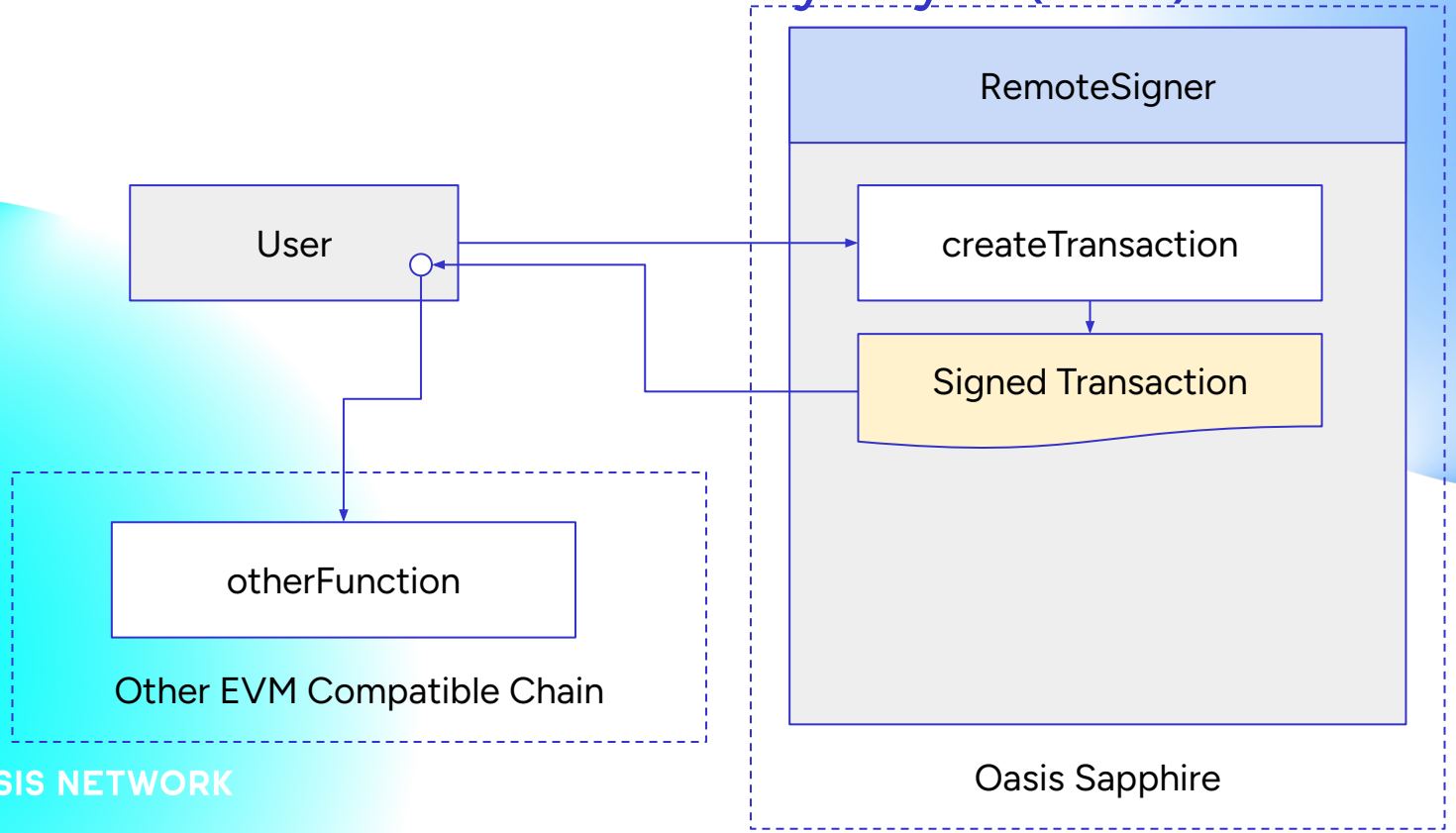
What to do next? Oasis Privacy Layer (OPL)

- Connect Sapphire with the most popular EVM networks (Ethereum, BNB Chain, Polygon…) and add encrypted transactions and confidential state.
- Learn more: docs.oasis.io/dapp/opl

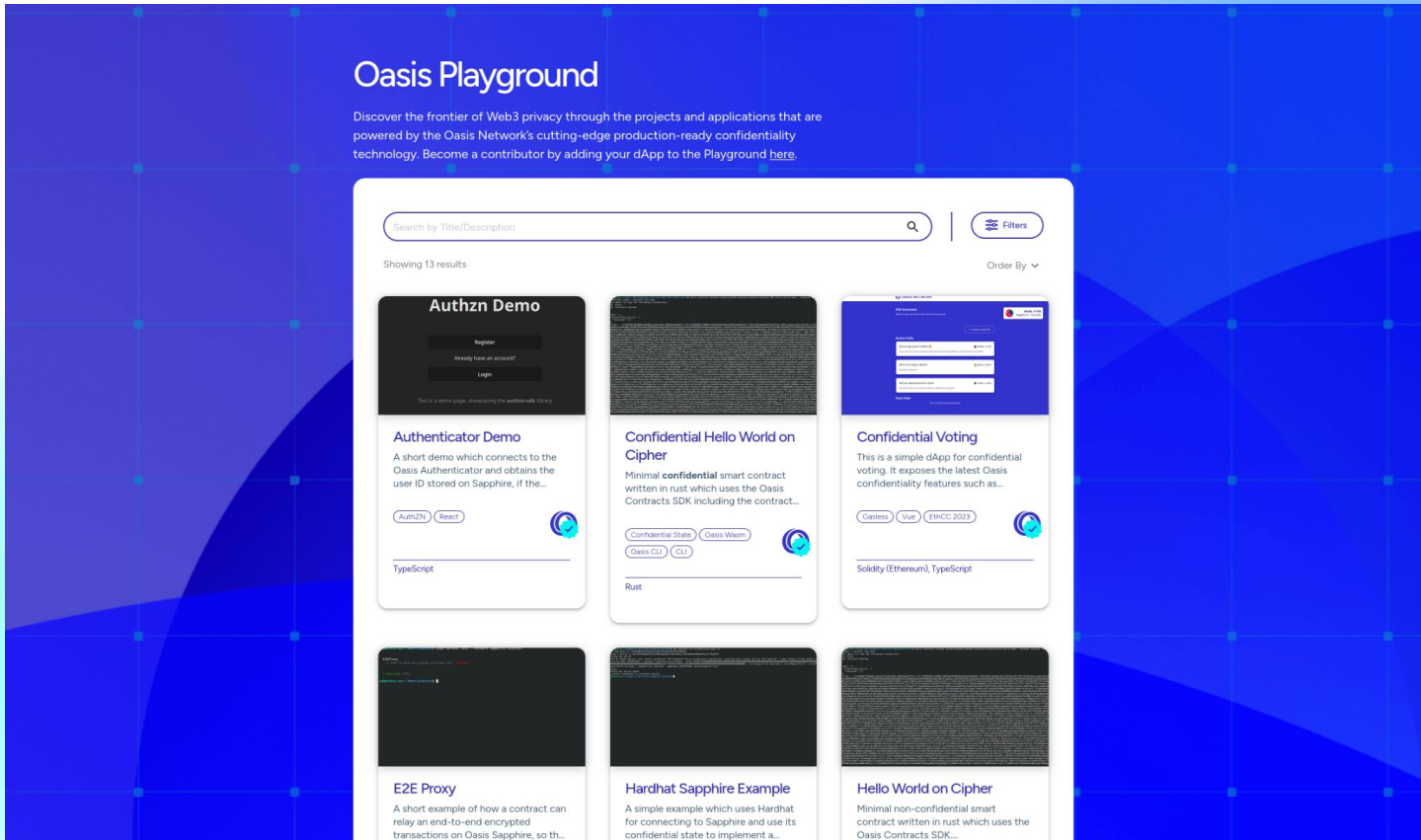


OASIS NETWORK

What to do next? Oasis Privacy Layer (OPL)



What to do next? Oasis Playground at playground.oasis.io

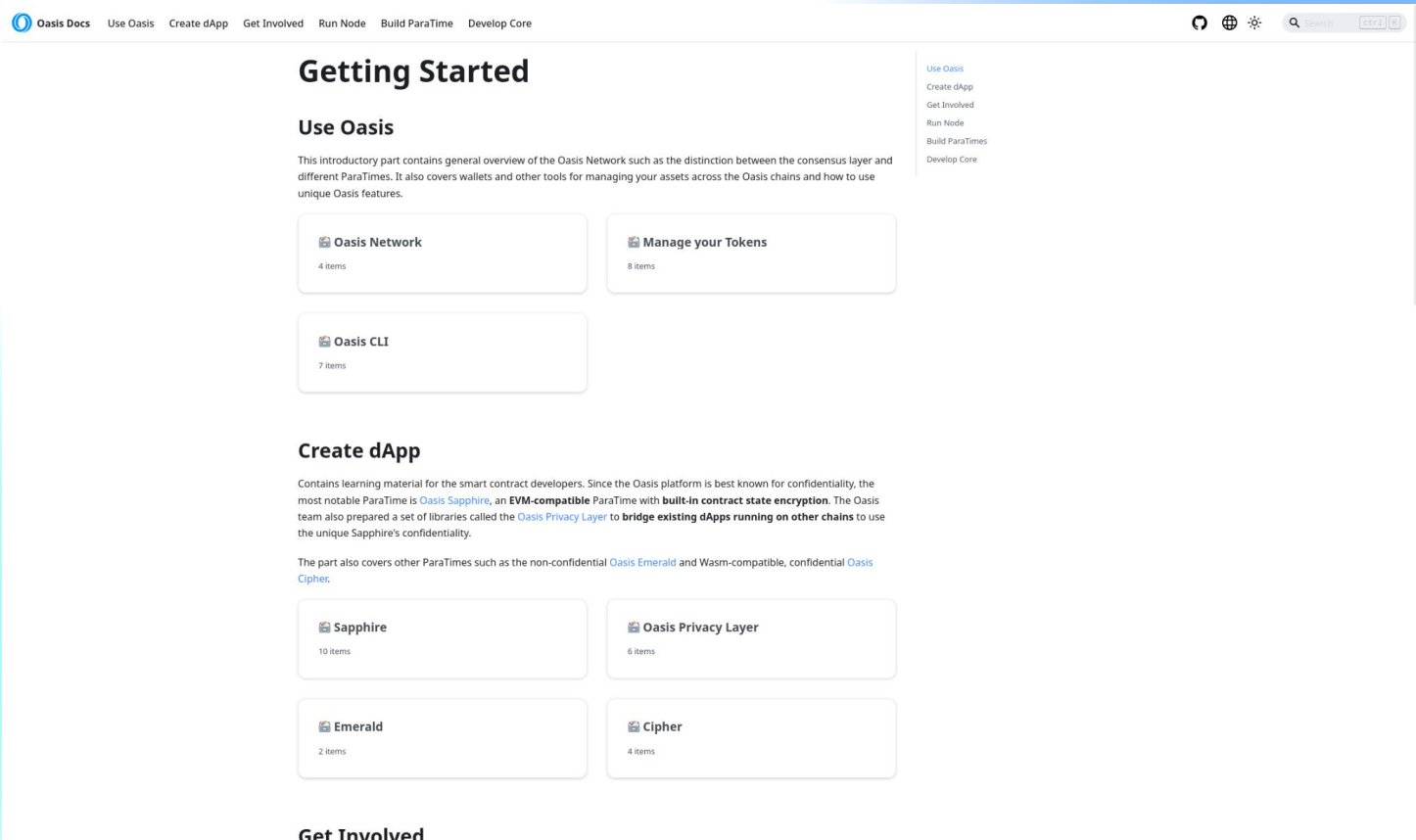


OASIS NETWORK

What to do next? Run your node

- Oasis network is a **decentralized network**
- You can help with the decentralization by **becoming a validator** and/or **running a ParaTime compute node** (Emerald, Sapphire, Cipher)
- Or, you can simply run a consensus and/or ParaTime **client node** along with the **web3 gateway** to avoid public (and malicious?) endpoints
- Learn more: docs.oasis.io/get-involved/run-node

What to do next? Oasis Documentation docs.oasis.io



OASIS NETWORK

Question, Answers & Proposals

OASIS NETWORK

THANK YOU

SEE YOU NEXT TIME!

OASIS NETWORK

Task 5: Encryption (TypeScript)

```
import * as sapphire from '@oasisprotocol/sapphire-paratime'

const e2ePubKey = await e2e.getPublicKey();
const box = sapphire.cipher.X25519DeoxysII.ephemeral(e2ePubKey);

let {nonce, cipherText} = await box.encrypt(plaintextBytes);

const nonceBytes32Hex = ethers.utils.hexlify(nonce) +
"0000000000000000000000000000000000000000";
```


OASIS NETWORK

Task 5: Decryption (Solidity)

```
import "@oasisprotocol/sapphire-contracts/contracts/Sapphire.sol";

contract E2EProxy {
  Sapphire.Curve25519PublicKey internal immutable publicKey;
  Sapphire.Curve25519SecretKey internal immutable privateKey;

  constructor (bytes memory extraEntropy) {
    (publicKey, privateKey) = Sapphire.generateCurve25519KeyPair(extraEntropy);
  }

  function getPublicKey() external view returns (bytes32) {
    return Sapphire.Curve25519PublicKey.unwrap(publicKey);
  }
}
```


OASIS NETWORK

Task 5: Decryption (Solidity)

```
import "@oasisprotocol/sapphire-contracts/contracts/Sapphire.sol";

contract E2EProxy {
  function decrypt(bytes32 peerPublicKey, bytes32 nonce, bytes memory data)
    external
  {
    bytes32 sk = Sapphire.deriveSymmetricKey(
          Sapphire.Curve25519PublicKey.wrap(peerPublicKey),
          privateKey);
    bytes memory plaintext = Sapphire.decrypt(sk, nonce, data, "");
  }
}
```


OASIS NETWORK

Task 5: Signing (Solidity)

```
import "@oasisprotocol/sapphire-contracts/contracts/Sapphire.sol";

contract SigningExample {
  function sign(bytes32 hashed_message)
      external returns (bytes)
  {
      Sapphire.SigningAlg alg =
Sapphire.SigningAlg.Secp256k1PrehashedKeccak256;

      bytes memory pk;
      bytes memory sk;
      bytes memory digest = abi.encodePacked(hashed_message);

      Bytes memory entropy = Sapphire.randomBytes(32, "");
      (pk, sk) = Sapphire.generateSigningKeyPair(alg, entropy);

      return Sapphire.sign(alg, sk, digest, "");
  }
}
```


OASIS NETWORK

Task 5: Verification (Solidity)

```
import "@oasisprotocol/sapphire-contracts/contracts/Sapphire.sol";

contract VerifyExample {
  function verify(bytes32 hashed_message, bytes memory signature)
     external returns (bool)
  {
     Sapphire.SigningAlg alg =
Sapphire.SigningAlg.Secp256k1PrehashedKeccak256;

     bytes memory digest = abi.encodePacked(hashed_message);

     return Sapphire.verify(alg, pk, digest, "", signature);
  }
}
```


OASIS NETWORK

Task 5: Autonomous Contracts

Ethereum compatible key-generation & signing is included in the `@oasisprotocol/sapphire-contracts` package:

- EthereumUtils.sol
  - `a, s = EthereumUtils.generateKeypair()`
- EIP155Signer.sol
  - `struct EIP155Signer.EthTx`
  - `EIP155Signer.sign(address,secret,tx)`

Task 5: Ethereum Keypair Generation and Signing (Solidity)

```
import '@oasisprotocol/sapphire-contracts/contracts/EthereumUtils.sol';

contract KeypairExample {
  address pubkey;
  bytes32 secret;

  constructor () {
    (pubkey, secret) = EthereumUtils.generateKeypair();
  }

  function sign(bytes memory data)
    external view returns (SignatureRSV memory rsv)
  {
    bytes32 digest = keccak256(data);
    rsv = EthereumUtils.sign(pubkey, secret, digest);
  }
}
```


OASIS NETWORK

Task 5: Transaction Signing (Solidity)

```
function signTx(uint64 nonce, uint256 gasPrice, uint256 value)
  external view returns (bytes memory txdata)
{
  return EIP155Signer.sign(pubkey, secret, EIP155Signer.EthTx({
    nonce: nonce,
    gasPrice: gasPrice,
    gasLimit: 250000,
    to: msg.sender,
    value: value,
    data: "",
    chainId: block.chainid
  }));
}
```


OASIS NETWORK

Task #1
Message Box

OASIS NETWORK

Task #1: **Message Box**

1. Use Remix to import Git project **github.com/oasisprotocol/demo-starter**, branch **split2023-task1**
2. Deploy the contract
3. Post a message
4. Find your **contract create** and **contract call** transactions on explorer.
   Are they encrypted? Why?
5. Connect to my contract **0x6cEE2AF84941381Bc5f8eA7122e0a70e9Dc1eDbE** and submit a message!

OASIS NETWORK

Task #2: **C10l Message Box**

1. Git clone project **github.com/oasisprotocol/demo-starter**, branch **split2023-task2**
2. Deploy the contract
3. Post a confidential message **using hardhat task** to your second Metamask account
4. Find your **contract create** and **contract call** transactions on explorer. Are they encrypted? Why?
5. Connect to my contract **0x682A74921B6f0194509C01490e35Ac0063a6a1b1** and submit a message for my account **0x90adE3B7065fa715c7a150313877dF1d33e777D5**!

OASIS NETWORK

Task #3
Message Box in the browser

OASIS NETWORK

Task #3: **Message Box in the browser**

1. Git clone project **github.com/oasisprotocol/demo-starter**, branch **split2023-task3**

2. Deploy the contract and configure the contract address in the Frontend folder. Spin up the Frontend locally!

3. Post a confidential message **from the browser** and view it by switching to the second account

4. Find your **contract create** and **contract call** transactions on explorer. Are they encrypted? Why?

5. Connect to my contract **0x682A74921B6f0194509C01490e35Ac0063a6a1b1** and submit a message for my account **0x90adE3B7065fa715c7a150313877dF1d33e777D5**!

Getting Started: Localnet (sapphire-dev)

Run a full Sapphire node locally.

```
IMAGE=ghcr.io/oasisprotocol/sapphire-dev:latest

docker pull $IMAGE

docker run --rm -it -p8545:8545 -p8546:8546 $IMAGE
```

Differences vs Ethereum

The big questions:

- Calldata Encryption
- Encrypted Storage
- Can you trust `msg.sender` ?

How?

- Trusted Execution Environment (Intel SGX)
- Key Manager & EVM run inside different TEEs

Content: **Lorem Ipsum Dolor**

- Reputation/Credit-based Lending
- Crypto Gaming (w. game logic on chain to enable full transparency and auditability)
- Private DEX to eliminate MEV issues
- Decentralized Identity, KYC/AML, verifiable credentials
- Decentralized Society
- Confidential NFT, Data backed NFT