Building an NFT Merkle-Drop

and other lazy-minting cryptography tricks.

zpl.in/contracts-workshop

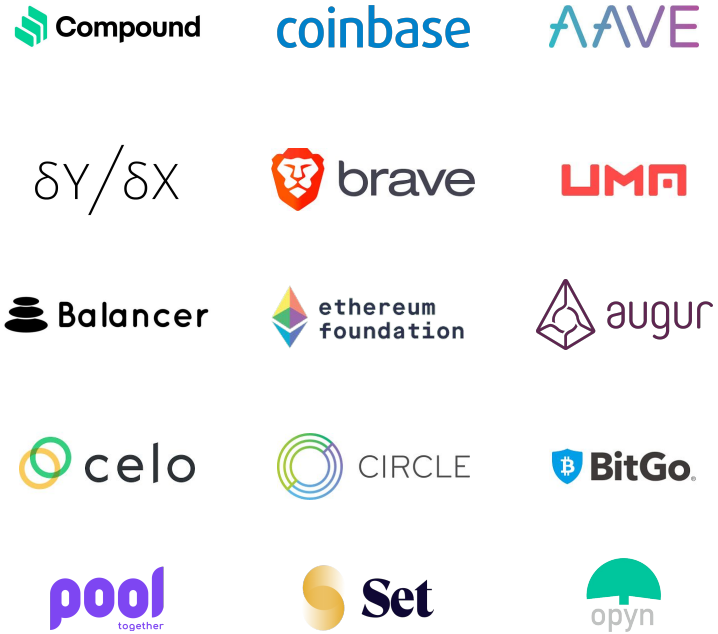
Hadrien Croubois
hadrien@openzeppelin.com
@amxx

OpenZeppelin

Our mission is to protect the open economy

OpenZeppelin is a software company that provides **security audits** and **products** for decentralized systems.

Projects from any size — from new startups to established organizations — trust OpenZeppelin to build, inspect and connect to the open economy.



Security, Reliability and Risk Management

OpenZeppelin provides a complete suite of **security and reliability products** to build, manage, and inspect all aspects of software development and operations for Ethereum projects.

We’re hiring!

Open Roles

- Security Researcher
- Full Stack Ethereum Developer
- Technical Community Manager
- Technical Product Marketing Manager

Apply here

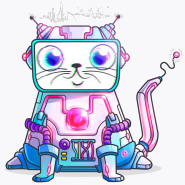
openzeppelin.com/jobs

ERC721 NFTs

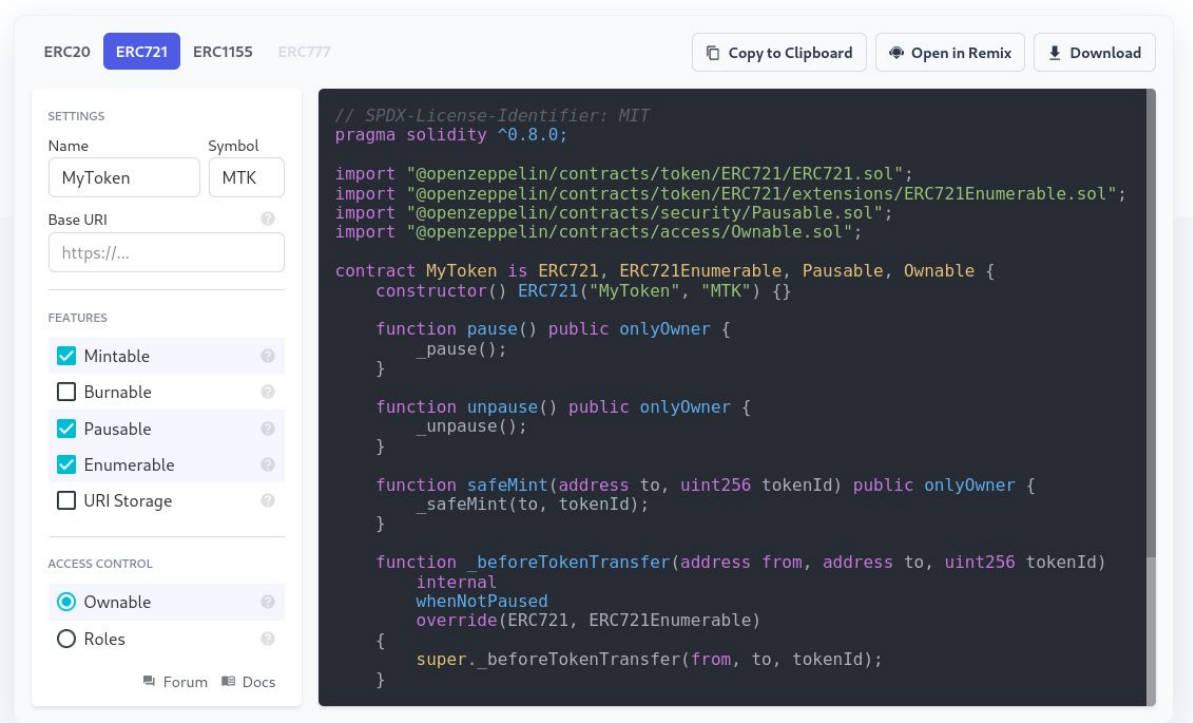
A brief overview

NFTs track the ownership of digital assets

Collectible, Domains, Financial positions, ...



OpenZeppelin Contract's ERC721 is highly customizable through modules



ERC721 Modules

- **ERC721Burnable:**

  Add a public `burn` function

- **ERC721Enumerable:**

  Add token enumerability: `totalSupply`, `tokenByIndex` and `tokenOfOwnerByIndex`

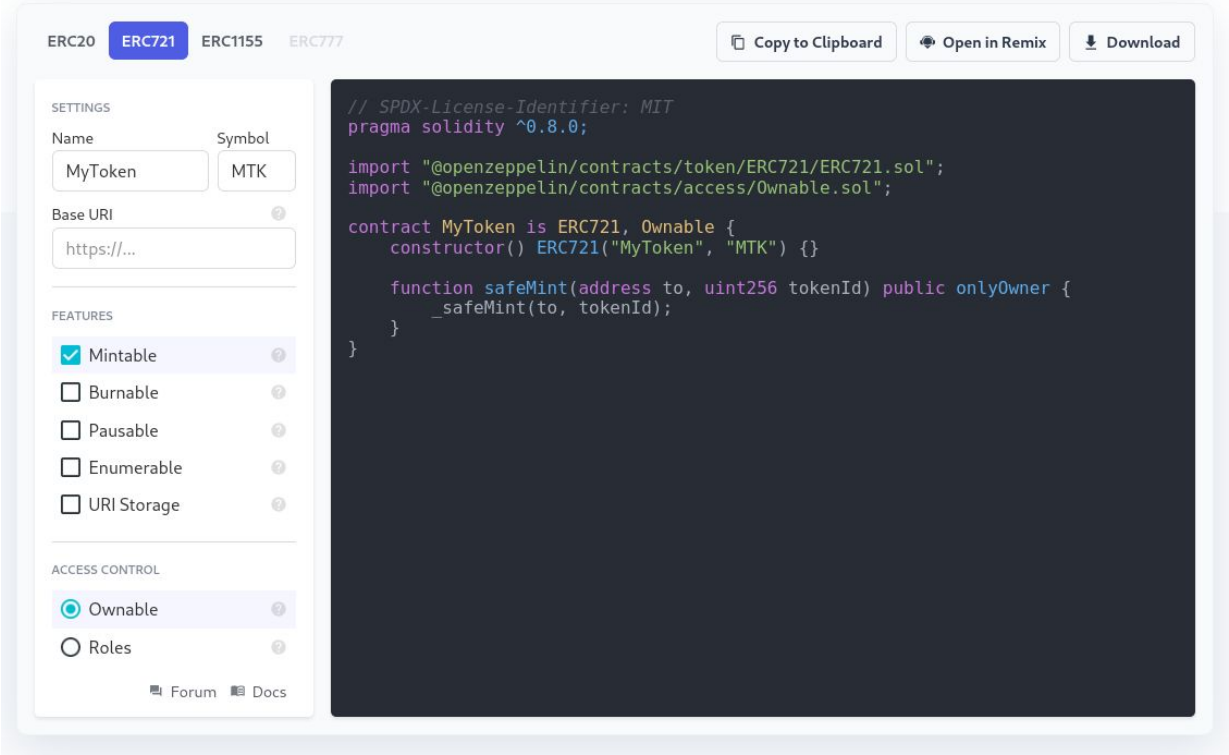
- **ERC721Pausable**

  Inherit from pausable and disable transfers when contract is paused

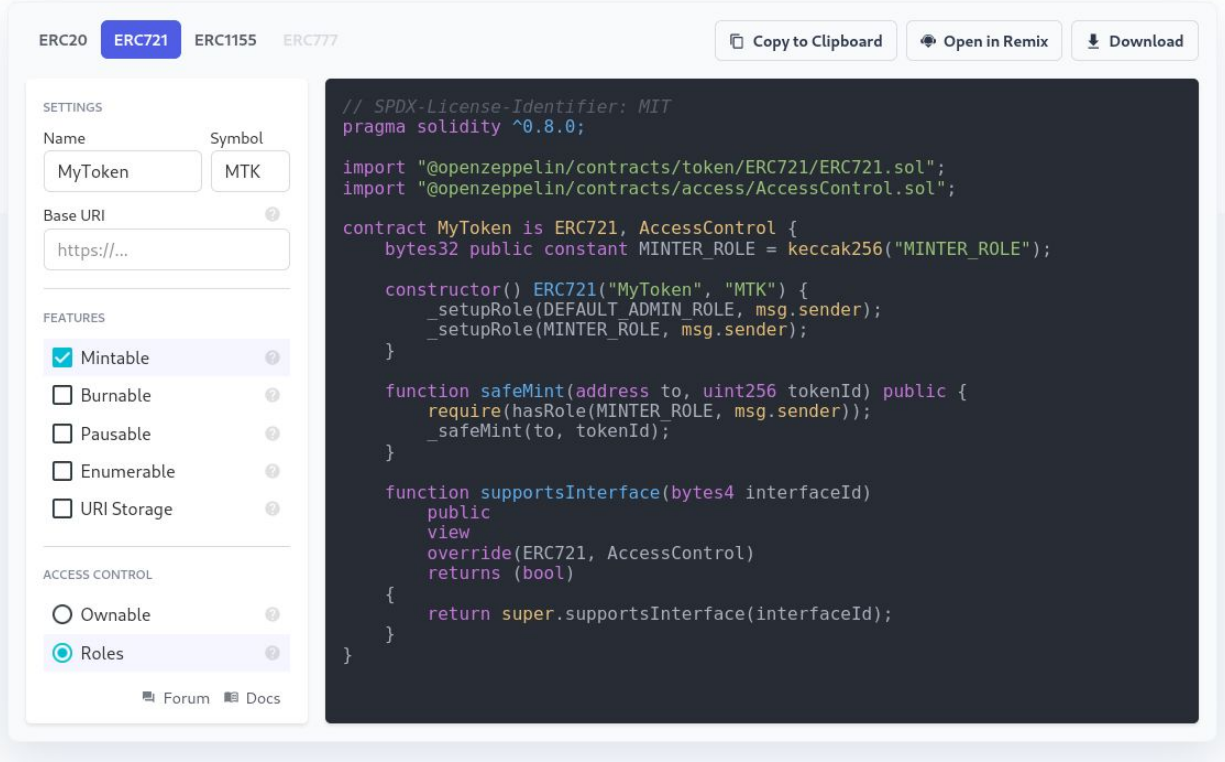
- **ERC721URIStorage**

  Add token specific URI

ERC721 Minting



ERC721 Minting



What is lazy minting?

Cost of minting an NFT: $19.63 per token

Gas price: 80 gwei, Eth value: $3549

What if the initial owner could mint the token ?

Idea:

The minting account “authorizes” the minting of a token

↓

The user does the minting

Challenge:

Providing authorization without sending a transaction

Lazy minting options

- “PersonalSign”

```
import "@openzeppelin/contracts/utils/cryptography/ECDSA.sol";
```

- ERC712 “signTypedData”

```
import "@openzeppelin/contracts/utils/cryptography/draft-EIP712.sol";
```

- ERC1271 signature validation method for contracts

```
import "@openzeppelin/contracts/utils/cryptography/SignatureChecker.sol";
```

More lazy minting options

- **Merkle-trees**

```
import "@openzeppelin/contracts/utils/cryptography/MerkleProof.sol";
```

Lazy minting with signatures

Signature:

Data + Private Key → Signature

Recovery:

Data + Signature → Public Key

About ECDSA signature

- **Private keys are used to sign many things:**
  - **Transactions**
  - **Messages**

```
// ethers
signer.signTransaction(transactionRequest) → Promise<string<RawSignature>>;
signer.signMessage(message) → Promise<string<RawSignature>>;

// web3
web3.eth.accounts.signTransaction(tx, privateKey [, callback]);
web3.eth.accounts.sign(data, privateKey);
```

About ECDSA signature

- **Signature can be recovered onchain & offchain**

```
// ethers
ethers.utils.recoverAddress(digest, signature) → string<Address>;

// web3
web3.eth.accounts.recover(message, signature [, preFixed]);

// solidity
import "@openzeppelin/contracts/utils/cryptography/ECDSA.sol";
address signer = ECDSA.recover(bytes32 digest, bytes memory signature);
```

About messages, prefix, and digest

- In order to avoid mistakes between transaction and messages, messages are prefixed
- What is **signed** is not the message, but a **digest** of the message.

```
keccak256("\x19Ethereum Signed Message:\n" + message.length + message);
```

- Messages are usually fixed length (32 bytes hashes produced using keccak256)

```
address signer = ECDSA.recover(
     ECDSA.toEthSignedMessageHash(keccak256(message)),
     signature
);
```

Demo Time

Hands-on with the code

zpl.in/contracts-workshop

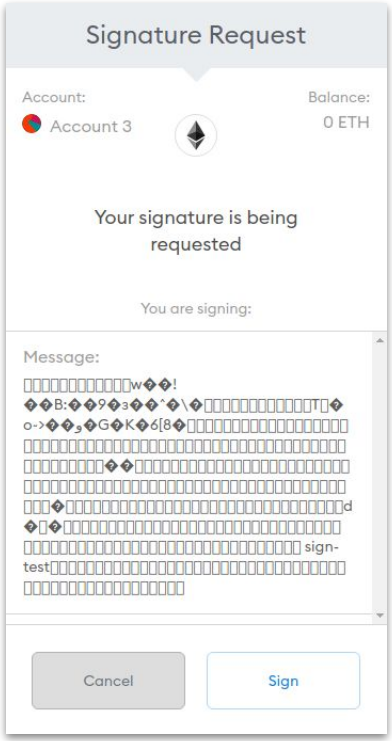
Better digest construction with ERC712

- ERC712 improves the way digest are built
  - Uses a “domain separator” to describe the targeted verifier (chainid, address, …).
  - Uses a “structure hash” that describes the data structure.
- Onchain verification is more complex, OpenZeppelin provides the `EIP712` helper contract.

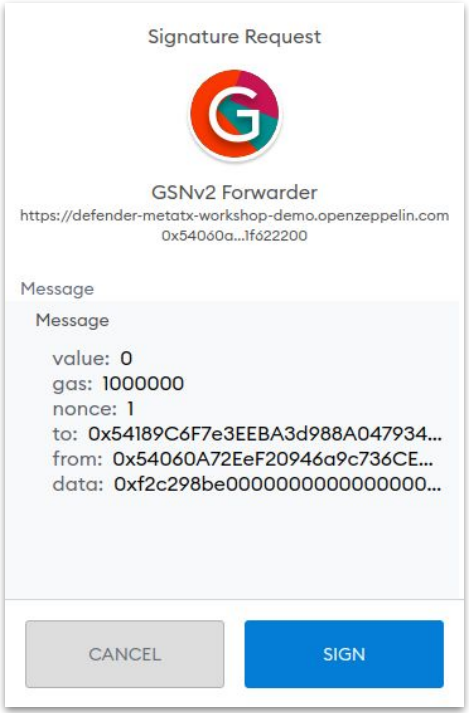
```
import "@openzeppelin/contracts/utils/cryptography/draft-EIP712.sol";

address signer = ECDSA.recover(
     _hashTypedDataV4(keccak256(abi.encode(
          keccak256("Structure(address sender,address receiver)"),
          sender,
          receiver
     ))),
     signature
);
```

Sign Message



Sign Typed Data (EIP712)



Demo Time

Hands-on with the code

zpl.in/contracts-workshop

What about large airdrops?

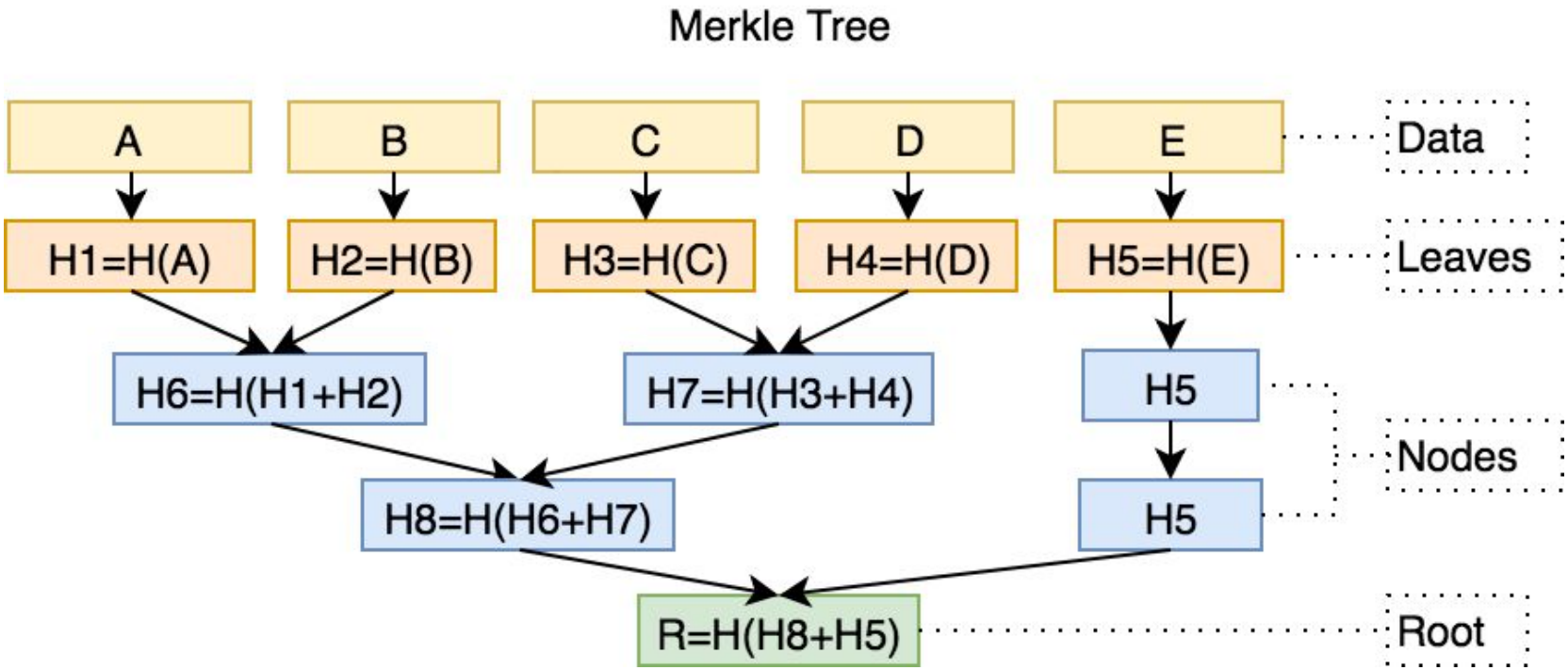
From Merkle-trees to Merkle-drops

What is a merkle-tree ?

In cryptography and computer science, a hash tree or ***Merkle tree*** *is a tree in which* ***every leaf node is labelled with the cryptographic hash of a data block****, and* ***every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes****. Hash trees allow efficient and secure verification of the contents of large data structures. Hash trees are a generalization of hash lists and hash chains.*

Wikipedia

What is a merkle-tree ?

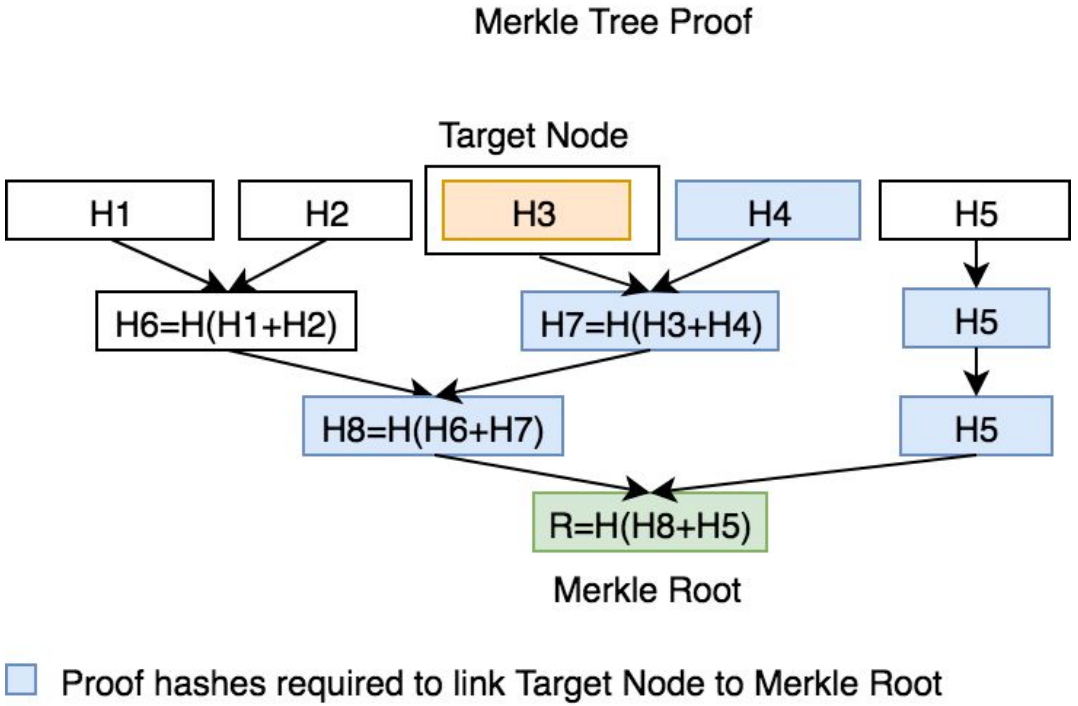


Verifying a merkle-proof

- **Leaf data + hash of neighboring nodes = Merkle-proof**
- **Verification:**
  - Hash the leaf data,
  - Iteratively hash the current node with the neighboring nodes from the proof
  - Verify that the result is the root.
- **Depending on the hashing function, at each level you might need to know if the neighbor is left or right.**

Proof length is logarithmic relative to the number of leafs

Verifying a merkle-proof



Verifying a merkle-proof

```
import "@openzeppelin/contracts/utils/cryptography/MerkleProof.sol";

function _verify(bytes32 root, bytes32 leaf, bytes32[] memory proof)
internal view returns (bool)
{
     return MerkleProof.verify(proof, root, leaf);
}
```

Generating a merkle-tree & merkle-proof

```
const { MerkleTree } = require('merkletreejs');
const keccak256      = require('keccak256');

function hashToken(tokenId, account) {
     return Buffer.from(ethers.utils.solidityKeccak256(
          ['uint256', 'address'],
          [tokenId, account],
     ).slice(2), 'hex');
}

const tokens = {
     '056665177': '0xa111C225A0aFd5aD64221B1bc1D5d817e5D3Ca15',
     '364166988': '0x8de806462823aD25056eE8104101F9367E208C14',
     '777704111': '0x801EfbcFfc2Cf572D4C30De9CEE2a0AFeBfa1Ce1',
};
const leaf       = Object.entries(tokens).map(token => hashToken(...token));
const merkletree = new MerkleTree(leaf, keccak256, { sortPairs: true });
const proof      = merkleTree.getHexProof(hashToken(...Object.entries(tokens)[0]));
```

Demo Time

Hands-on with the code

zpl.in/contracts-workshop

Conclusion: Cryptographic tooling

- EDCSA recovery for PersonalSign
- EIP712 helper for SignTypedData
- Signature verifier for ERC1271
- Merkle-proof verification

@openzeppelin/contracts

docs.openzeppelin.com

forum.openzeppelin.com

defender.openzeppelin.com

Thank you!

Learn more

openzeppelin.com/**contracts**
forum.openzeppelin.com
docs.openzeppelin.com

Contact

@amxx
hadrien@openzeppelin.com