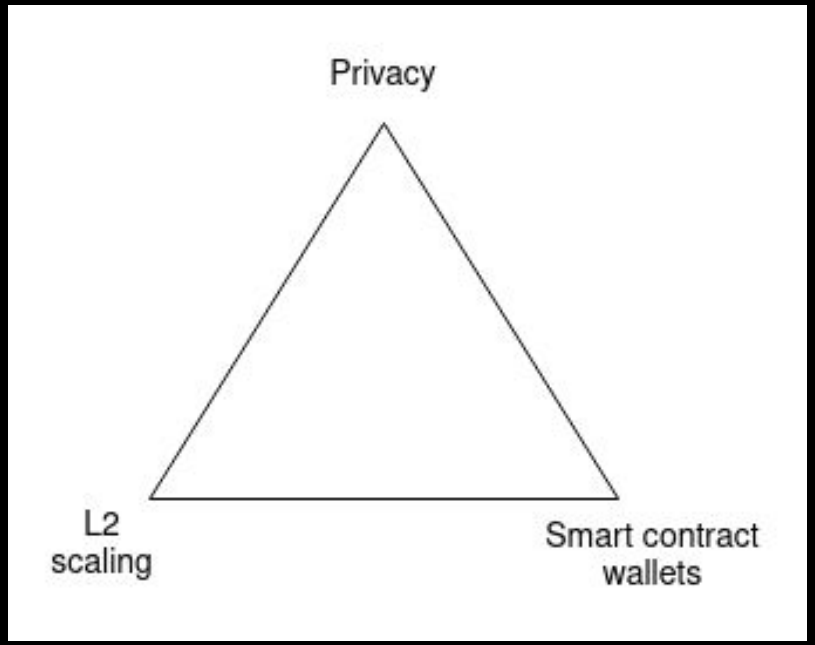
The Three Transitions

Cross-Chain Smart Wallets with Privacy

Niran Babalola, Base

VITALIK'S VISION

The Three Transitions



"Without [L2 scaling], Ethereum fails because each transaction costs $3.75 ($82.48 if we have another bull run)..."

"Without [smart contract wallets], Ethereum fails because users are uncomfortable storing their funds..."

"Without [privacy], Ethereum fails because having all transactions (and POAPs, etc) available publicly for literally anyone to see is far too high a privacy sacrifice for many users..."

SMART WALLETS ARE HERE

Seed phrase wallets are hard to use.

Install a wallet. Here are 400+ wallets to choose from.

Write down your seed phrase. Don’t lose it. Copy it to every device you want to use. You can’t change it, so if someone else gets it, you’re screwed.

Make sure you fund your account to pay for gas. Want to use multiple accounts? Fund them all.

Sign this approval transaction to send some tokens. Now sign this transfer transaction to send some tokens. You can’t batch them together.

SMART WALLETS ARE HERE

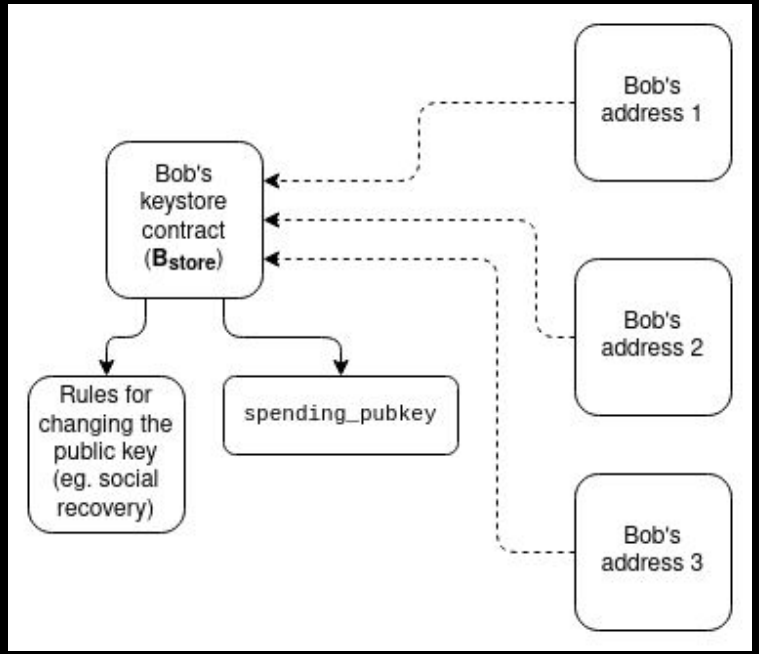
Smart Wallet

Add and remove signers as needed.

- When you change signers, the message you sign can be replayed on any chain
- There is **no eventual consistency** between your wallets on different chains
- Configuration for new features (like session keys) needs to be synced separately
- Every wallet vendor has to build all this themselves

VITALIK'S VISION

Keystore Wallets



Keystore wallets keep their configuration in sync by referencing their *keystore*, the single source of truth for the wallet's latest configuration. The keystore lives on one chain and is read by wallets on all chains.

- The Three Transitions, June 9, 2023
- Deeper dive on cross-L2 reading for wallets and other use cases, June 20, 2023
- Possible futures of the Ethereum protocol, part 2: The Surge, October 17, 2024

Safe Wallet, Scroll, Stackr, and other teams have been pushing these ideas forward.

WHAT IS KEYSPACE?

Keyspace is a keystore for cross-chain smart wallets that keeps their configuration in sync.

KEYSPACE 101

Give your wallet a keystore

Inheriting from Keystore builds the ability to sync configuration across chains directly into your wallet contract.

- One chain is the **master chain**, the rest are **replica chains**
- Signer changes are written directly to the master chain, so it never needs to sync
- Cheaper and easier than a dedicated keystore L2, but finalizes more slowly and involves more moving parts

```
abstract contract Keystore {
```

```
abstract contract OPStackKeystore is Keystore {
```

```
contract MultiOwnableWallet is OPStackKeystore, UUPSUpgradeab
```

```
constructor(uint256 masterChainId)
                        OPStackKeystore(masterChainId) {}
```

KEYSPACE 101

Choose your master chain

Many wallet teams run their own L2 chains, and we expect them to choose their chain as the master. OP Stack chains are supported out of the box, and we want to help you with your PRs for other stacks.

- Coinbase Wallet + Base
- MetaMask + Linea
- Uniswap + Unichain
- Worldcoin + World Chain
- Zora App + Zora Chain
- Abstract Global Wallet + Abstract
- Kraken Wallet + Ink
- ... and more

KEYSPACE 101

Update your configuration

Sign a hash of the new configuration for your wallet and set it on the master chain.

- Use update hooks to write the new configuration and validate that the new configuration can verify signatures.
- For multisig wallets, skip the update hooks to allow for easy signer additions and threshold increases.
- Your configuration is usable on the master chain without syncing.

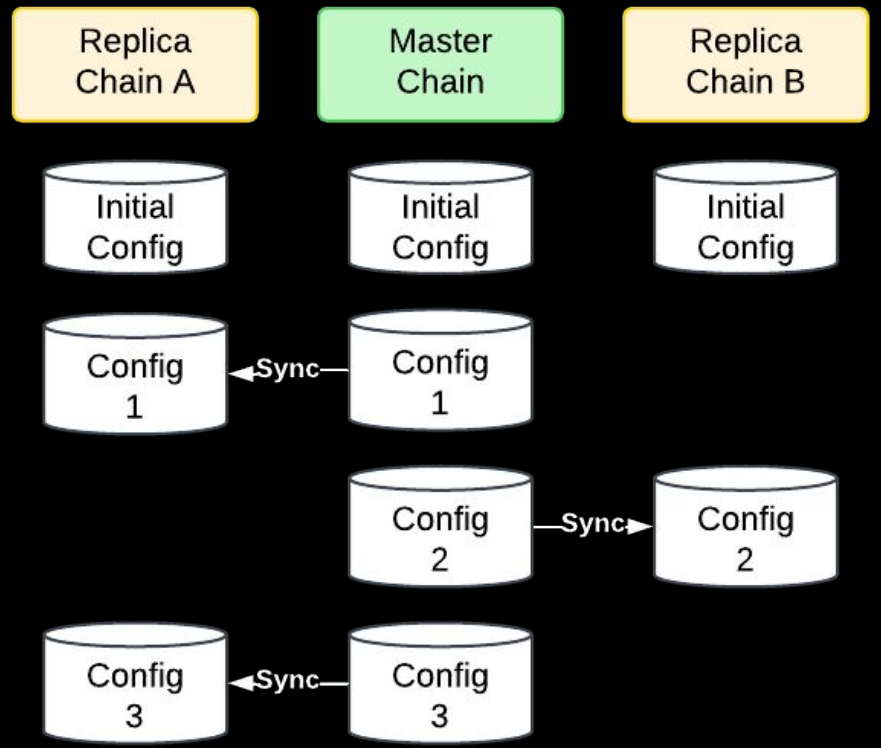
```
// Hook before (to authorize the config update).
_authorizeConfigUpdateHook({newConfig: newConfig, authorizationProof:
authorizationProof});

// Apply the update to the internal Keystore storage.
newConfigHash = applyConfigInternal(newConfig);

// Hook between (to apply the update).
bool triggeredUpgrade = _applyConfigHook({config: newConfig});

// Hook after (to validate the update).
if (triggeredUpgrade) {
    this.validateConfigUpdateHook({newConfig: newConfig,
    authorizationProof: authorizationProof});
} else {
    validateConfigUpdateHook({newConfig: newConfig,
    authorizationProof: authorizationProof});
}
```

KEYSPACE 101

Sync configuration to replica chains

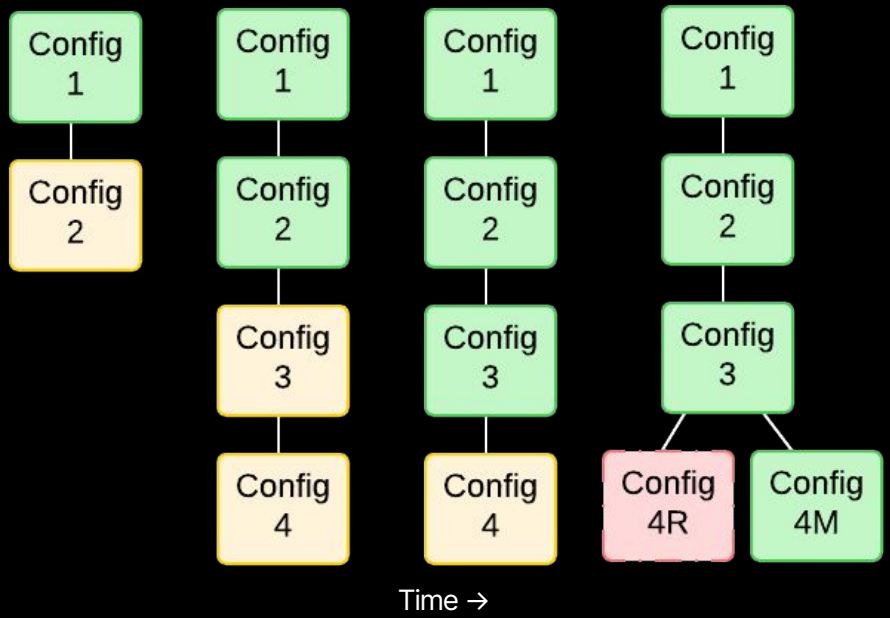
Keyspace is eventually consistent, and you get to configure how often syncing is required. Our TypeScript keyspace-client gives you everything you need to generate cross-chain Merkle proofs to stay in sync.

- Optimistic rollups take 3.5 to 10 days to finalize, so synced state lags by this duration.



KEYSPACE 101

Preconfirm changes on replica chains



Since most master chains have slow finalization, wallet clients preconfirm changes on replica chains as needed. The new configuration can be used in two seconds.

- The same signature used on the master chain is reused on replicas to update the config locally.
- Revoking a signer has security implications, so wallet clients preconfirm revocations on all active chains, not on demand.
- During syncing, any newer preconfirmations are preserved.

BASE

Key Moments in the Lifespan of a Wallet

01 Creating the Wallet

02 Updating the Wallet's Configuration

03 Upgrading the Wallet's Functionality

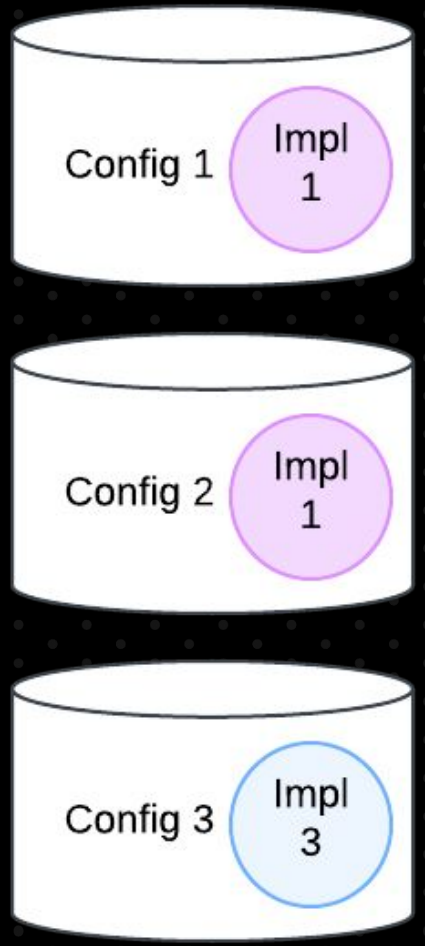
04 Initializing the Wallet on New Chains

WALLET LIFESPAN

Wallet configurations and implementations are intertwined

Your new wallet contract implementation needs to be compatible with the current configuration, so Keyspace helps you make atomic configuration + implementation upgrades.

- Store your implementation address within the keystore that Keyspace syncs across chains, and perform the upgrade any time a new implementation is available.
- For safety, verify that the upgraded configuration + implementation can still verify signatures as expected, and revert if the upgrade is inconsistent.



WALLET LIFESPAN

Wallet factories are forever

```
function createAccount(ConfigLib.Config calldata config, uint256 salt)
    external
    payable
    virtual
    returns (MultiOwnableWallet account)
{
    bytes32 configHash = ConfigLib.hash(config);
    (bool alreadyDeployed, address accountAddress) = LibClone.
    createDeterministicERC1967(
        msg.value, implementation, _getSalt({configHash: configHash,
        salt: salt})
    );

    account = MultiOwnableWallet(payable(accountAddress));

    // If the account is newly deployed, initialize it with the
    provided Keystore config.
    if (!alreadyDeployed) {
        account.initialize(config);
    }
}
```

There will be many new chains. When you initialize your wallet on a new chain, the only way to maintain the same address is for the wallet to be generated from the same factory. But each L1 and L2 hard fork puts cross-chain syncing at risk.

- L1 can break syncing by changing the state root. If the Merkle-Patricia tree root becomes a Verkle root, syncing breaks.
- L2 can break syncing with new system contracts, like the DisputeGameFactory or a replacement for L1Block.
- Eventual consistency requirements are waived to allow wallets to update themselves to the latest configuration from scratch, then sync from there.

USE KEYSPACE

With Keyspace, wallet vendors can deliver an account for every chain that's as easy to use as offchain accounts.

Privacy

BASE

Types of Privacy

Pseudonymity
Senders and recipients use many fresh identities
(e.g. Bitcoin HD wallets)

Anonymity
Senders and recipients are unknown, but amounts are known
(e.g. Tornado Cash)

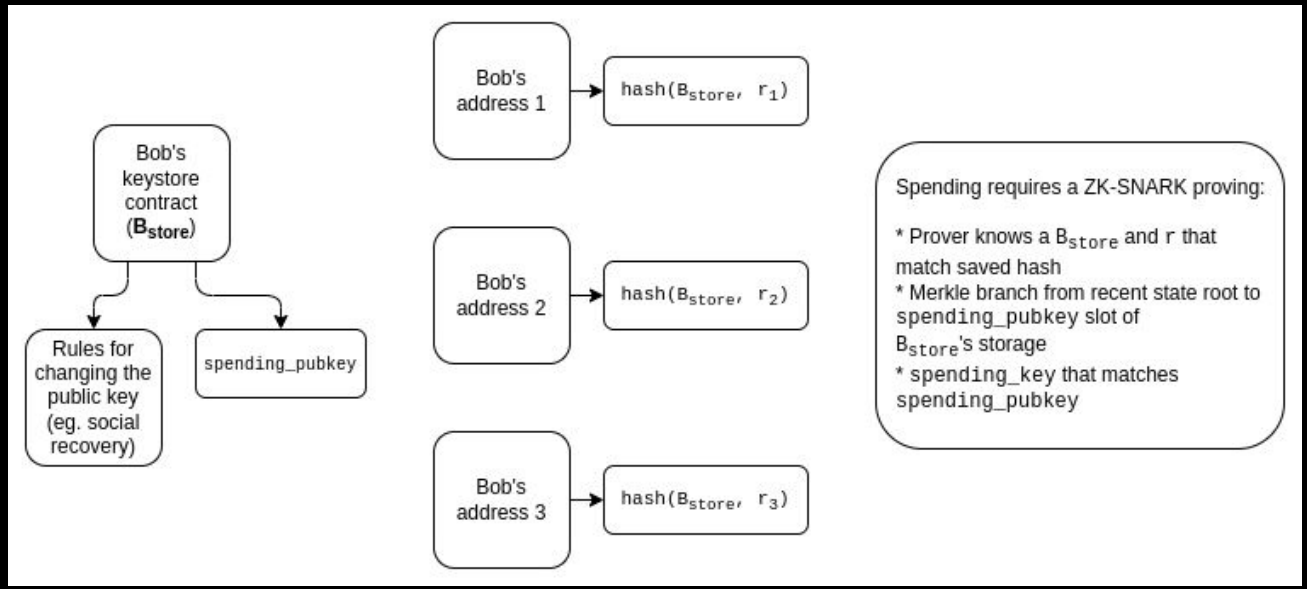
Confidentiality
Transaction amounts are unknown, but senders and recipients are known
(e.g. Confidential ERC-20)

PRIVACY

Keyspace brings pseudonymity to Ethereum

Pseudonyms can now be more usable than ever when you give smart wallets a keystore that keeps their configuration in sync.

- Ethereum EOAs have always had to fund their own gas, but smart wallets can use paymasters for their many pseudonyms.
- To get pseudonyms, create many wallets that reference the same keystore.
- To make the keystore private, use a SNARK to prove a blinded reference to the keystore.

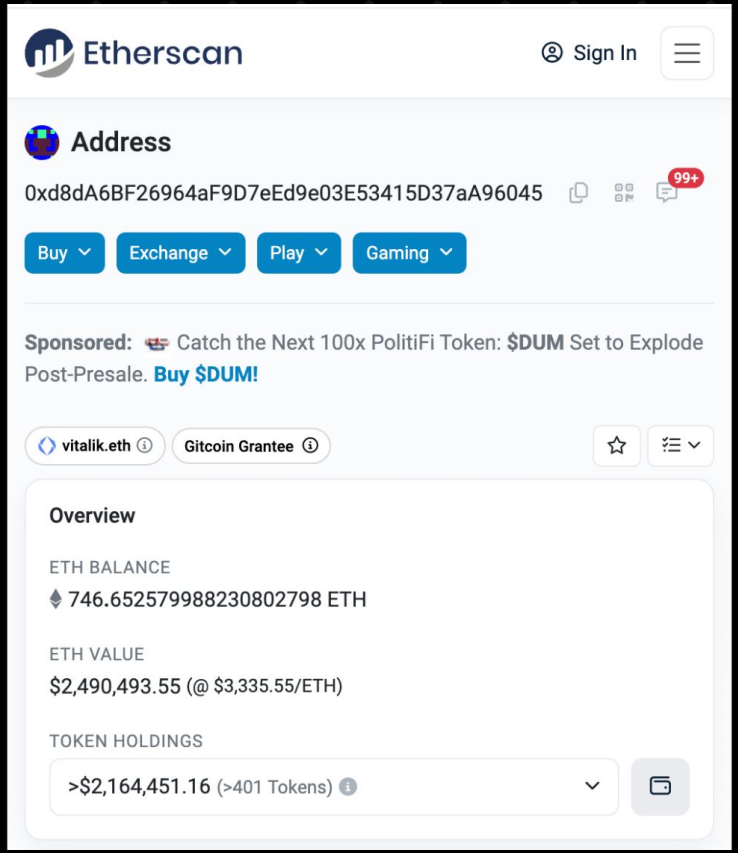


PRIVACY

Pseudonymity + Confidentiality is a sweet spot

You shouldn't be able to look up someone's wallet on Etherscan and learn their financial history.

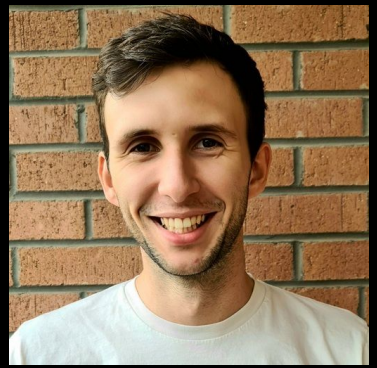
- Keyspace's **pseudonymity** makes it hard to learn who other people are transacting with.
- Confidential ERC-20 uses fully homomorphic encryption for **confidentiality**: it hides the amounts people are sending to each other.
- Combined, Ethereum smart wallets will be private enough to use as your primary financial account.



KEYSPACE

Team

NIRAN BABALOLA
Github: @niran
Twitter: @niran



BAPTISTE OUERIAGLI
Github: @xenoliss
Twitter: @xenoliss

MICHAEL DE HOOG
Github: @mdehoog
Twitter: @michaeldehoog

https://docs.key.space

THANK YOU

Questions?
Let’s Build

Appendix

SMART WALLETS ARE HERE

Smart Wallet

Simplify onboarding in seconds.

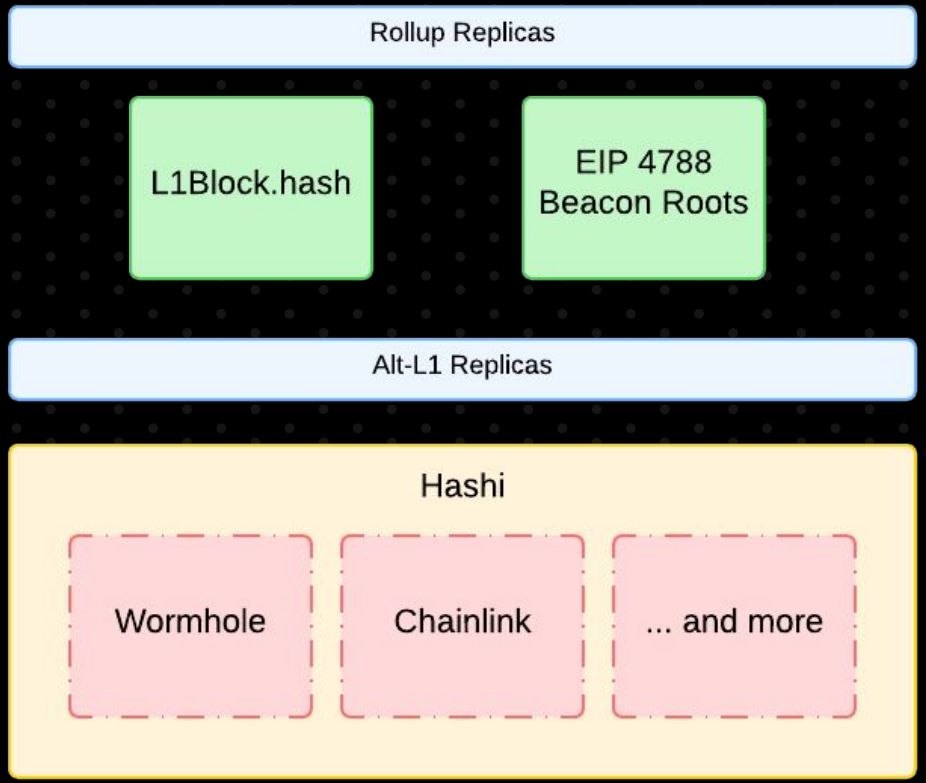
- Spin up a new wallet in seconds — accessible from any browser, no seed phrase required
- Easy to pay onchain with a self-custody wallet account or Coinbase account
- Portable to thousands of onchain apps
- Gasless with Paymaster integration

KEYSPACE 101

Rollups can sync trustlessly

On rollups, Keyspace syncs using the L1 block hash from wherever it can get it. On alt-L1 chains, Keyspace uses oracles.

- The main sources of trustless L1 block hashes are rollup system contracts like OP Stack's L1Block, and via EIP 4788's beacon chain roots.
- We use Hashi to make L1 block hash oracles more secure on alt-L1 chains, but it's always risky for wallets to rely on trust rather than cryptography.
- Some wallets will decide to rely on oracles on alt-L1s, others will only sync between rollups and allow configuration to diverge on alt-L1s.



BASE

User-Friendly Security

What if I lose my phone?

01 Time-Delayed Guardians

02 ZK Email Guardian

USER-FRIENDLY SECURITY

Authorize configuration changes using L1 block headers

With L1 block headers, your wallet's authorization logic for configuration updates can use the block timestamp or any onchain data.

- Time-delayed guardians can initiate a configuration change on an L2, then prove that initiation and the elapsed time using an L1 block header.
- ZK Email guardians can prove that a required email was signed by a specific DKIM signature, and use an onchain DKIM Registry to tie it to a sender's identity.
- Guardian recoveries work well on active chains. But if you lost your keys, you won't be able to use the wallet on new chains without syncing, which can break.