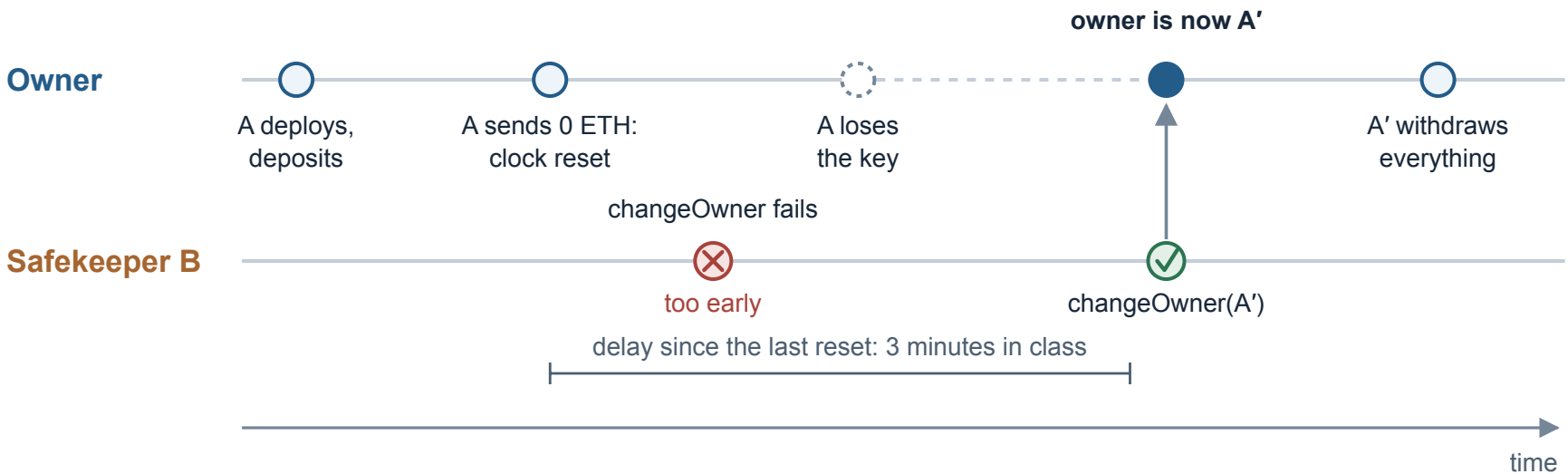
# Smart accounts

A vault in use, and Safe · Session 15 · 9 November 2026

# Where we left off



# The class vault

```
uint256 public constant DELAY = 3 minutes;
address public owner;  address public safeKeeper;  uint256 public lastReset;

constructor(address _safeKeeper) {
    owner = msg.sender;  safeKeeper = _safeKeeper;  lastReset = block.timestamp;
}
receive() external payable {                  // even 0 ether resets the clock
    require(msg.sender == owner, "Only owner can deposit");
    lastReset = block.timestamp;
}
function withdraw() external { ... }          // owner only; also resets the clock
function changeOwner(address _newOwner) external {
    require(msg.sender == safeKeeper, "Only the safekeeper can change owner");
    require(block.timestamp >= lastReset + DELAY, "Owner still active");
    owner = _newOwner;  lastReset = block.timestamp;
}
```

Full source, with `call` in place of `transfer`: /15/recoverable-vault

# A rotation, step by step

| Step | Call | Result |
|---|---|---|
| A deploys, naming B as safekeeper | constructor, no value | the vault starts its clock |
| A deposits | `receive`, with value | the clock restarts |
| anyone else tries | `changeOwner`, `withdraw` | ✕ "Only the safekeeper…", "Only owner…" |
| B acts too soon | `changeOwner(A′)` | ✕ "Owner still active" |
| B acts after the delay | `changeOwner(A′)` | ✓ A′ is the owner |
| A′ withdraws | `withdraw` | ✓ needs ether for gas |

Example, from class: vault 0x170B…1650. Every failed call is on the chain too.

# Verifying a contract

- On the chain there is only **bytecode**, what the machine runs. The source is not there.
- To verify, the explorer **compiles** the source we give it, with the same compiler and settings, and checks that the result matches the bytecode.
- Any explorer can do it, and they are independent: Blockscout, Etherscan, Routescan, or one we run ourselves.

Verification lets everyone read what a contract does. It does not say the code is correct.

# Who pays the gas?

A′ had no ether: its first `withdraw` could not be sent.

- In class, A sent it some ether from another account.
- A contract can pay the gas for someone else: a **paymaster** (ERC-4337, account abstraction).
- The paymaster sets its own conditions: pay for a recovery, then take a fee from the funds recovered.

# Challenge: rotate a key

In pairs, A and B, on Sepolia, in Remix. Source: /15/recoverable-vault

1. A deploys `RecoverableVault` with `_safeKeeper` = B's address, **with no value**.
2. A sends the vault some ether. B calls `changeOwner` at once: it fails.
3. A creates a second account in MetaMask, A′, and gives it a little ether for gas.
4. Three minutes after A's last deposit, B calls `changeOwner(A′)`.
5. From A′, call `withdraw`. Then swap roles.

A constructor accepts ether only if it is `payable`, like any function: this one is not, so a deploy that sends ether fails.

# Other rules

The delay is one choice among many.

| Rule | Effect |
|---|---|
| request and veto | the safekeeper asks; the owner can cancel during a waiting period |
| two safekeepers of three | no single safekeeper can take over |
| an identity document | a safekeeper without a wallet signs with an ID card |

"If I lose my key I lose everything" depends on **how the contract is written**.

# From a vault to an account

- With the vault, only what I put in is recoverable; what I keep for spending is at risk.
- Add a function to **pay** from the vault, and it works like an account.
- Add `execute(target, value, data)`, and it can call any contract: hold tokens, swap, vote.
- If I lose or change my key, the owner changes and **the address stays**.

A contract that does what an account does, with rules we choose: a **smart account**.

# Smart accounts on Ethereum

| | Since | Idea |
|---|---|---|
| ERC-4337 | 2023 | smart accounts as a standard, with fees paid by a paymaster |
| EIP-7702 | 2025 | an ordinary address delegates its behaviour to a contract |

With EIP-7702 an existing address, with its history, can become recoverable.

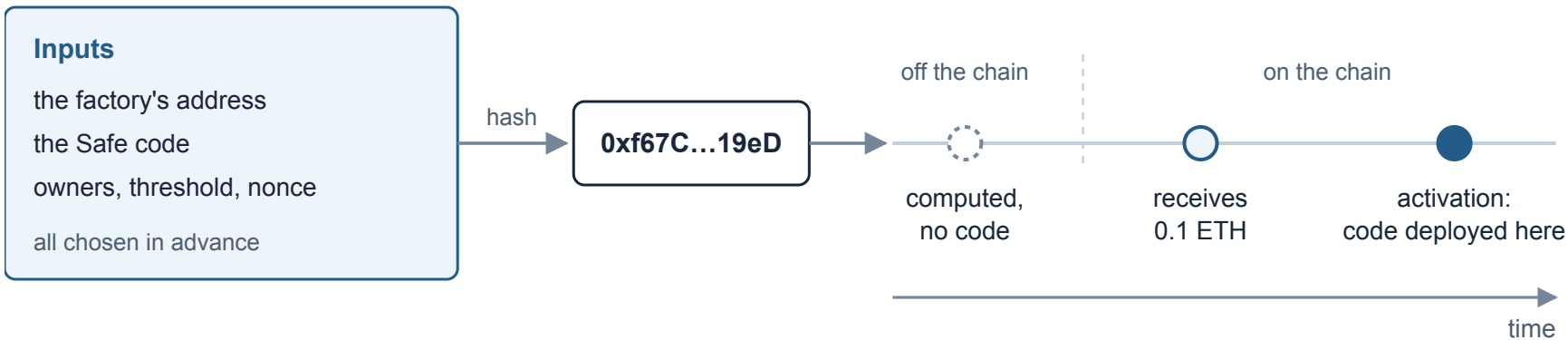
# Safe

The most widely used smart account, formerly **Gnosis Safe**: a contract that acts as an account.

| Setting | Example |
| --- | --- |
| **owners**: who may sign | A, B, C |
| **threshold**: how many signatures per transaction | 2 of 3 |
| **modules**: extra rules | a recovery delay, a check on risky transactions |

1 of 2: either owner spends, the other is effectively a safekeeper with full power. 2 of 2: nothing moves without both.

# An address before a contract




Same inputs, same address: the factory can only deploy the Safe that was promised.

A Safe can be paid before it exists, and deployed only when it is first needed.

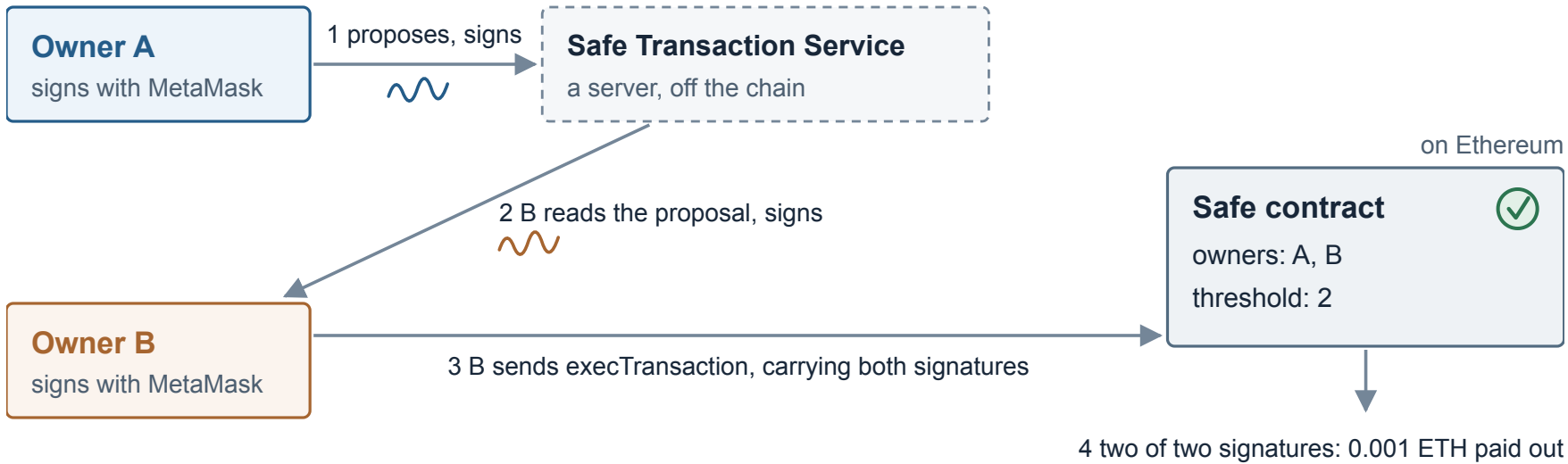
# Modules: rules added to an account

A Safe can be extended with **modules**, contracts allowed to act on it under their own rules.

| Module | Rule |
| --- | --- |
| recovery | a recoverer can replace an owner after a review window, during which the owners can cancel |
| guard | an external service can block transactions it judges risky |
| spending limit | a chosen address can spend up to an amount per period, without the others |

The safekeeper of our vault, as a module: the owners choose who, and with which limits.

# A payment from a Safe



# Challenge: a Safe for two

In pairs, on app.safe.global, network **Sepolia**.

1. Create a Safe with **both** your addresses as owners, threshold **2**.
2. Send it 0.01 ETH.
3. Propose a payment of 0.001 ETH to `0x9c8d1cb0C7d6eAD3141CFf8e9B95a610B86345dD`.
4. The second owner signs; the last signer executes.

On Blockscout: one `execTransaction`, two signatures.

# Bybit, 21 February 2025

- About **1.5 billion dollars** in ether taken from a Safe that needed several signatures.
- The contract worked as written. The attackers had compromised a Safe developer's machine and altered the **website**.
- The signers saw a routine transfer and signed a change of the Safe's own code.

Several keys protect against one lost key, not against **signing what you cannot read**. The FBI attributed the theft to North Korea.

# A bank as safekeeper

- Today a bank controls the **infrastructure** and has the **trust** of its customers.
- On a blockchain the infrastructure is no longer the bank's; the trust remains.
- A supervised institution can be the **recovery safekeeper** of a smart account.

  Not needed to move the funds; needed when something goes wrong.

# Rules for an authority

A smart account can also carry **compliance** rules:

- a module that lets a judicial authority freeze or seize funds, under stated conditions;
- an asset that can be held only in accounts with such a module.

Whether this is desirable is a political choice. The technology makes it possible, with **public rules** enforced by the network.

# And the chargeback?

Irreversibility was another reason on the list. With a card, a disputed payment can be reversed:

| Ingredient | With a card | In a contract |
|---|---|---|
| a time window | usually some months | a delay, as in the vault |
| an arbiter | the bank that issued the card | a third party, as the safekeeper |
| where the money waits | already with the merchant | in the contract |

# Next session, 16 November

- **Privacy**: everything we did today is public. Can it be otherwise?
- **Who pays the gas**: a paymaster.
- The **escrow**, if there is time.