Keys and recovery

Ownership, freezes and a recoverable vault · Session 14 · 2 November 2026

Why not pay for coffee in ether?

Reason (session 10)	Where it stands
Speed, cost	layer-2 chains: about a second, a fraction of a cent (later)
Volatility	stablecoins, sessions 10–13
Adoption	few merchants accept it
Technical risk	**today**
Compliance	**today**
Irreversibility	no chargeback
Privacy	the café’s address shows all its sales (later)

Ownership and control

	Ownership	Control
car	registration certificate	the keys
bank account	contract with the bank	credentials
ether, bitcoin	no technical instrument	the private key

- A lost car key or password: ownership is proved, **control is restored**.
- A lost private key: the funds stay visible on the explorer, and **nobody can move them**.

How much is lost?

- Chainalysis, 2020: about **3.7 million bitcoin** likely lost, roughly a **fifth** of the 18.6 million mined at the time.
- "Lost" here means: not moved for at least five years, and judged unlikely to move again.

Leaving the keys to someone else

Not your keys, not your coins: on an exchange you hold a **claim** on the exchange, not coins.

	Year	What happened
Mt. Gox	2014	the largest bitcoin exchange reports 850,000 BTC missing; 200,000 later found in an old wallet
QuadrigaCX	2019	the founder, sole holder of the keys, dies; customers lose at least C$169 million
FTX	2022	customer deposits used by the founder’s trading firm; bankrupt within days

Binance, 29 October 2026

- About **6.3 billion dollars** taken from Binance's hot wallets, on about ten chains, in some twenty minutes.
- The start, according to Binance: an engineer's **compromised laptop**, used to submit withdrawals. No private key leaked, it says.
- Tether, Circle and others **froze** about 3 billion dollars of the stolen stablecoins.
- Binance says it will **cover the losses**: on an exchange, the custodian bears them.

Two ways to hold an asset

	Keep the key yourself	Leave it to a custodian
who can lose it	you	the custodian
who can help	nobody	the custodian, if solvent and honest
who can block it	nobody	the custodian, or an authority through it

A custodian brings back a **trusted intermediary**, often a less supervised one than a bank.

The choice looks sharp. Code lets us design **trade-offs** in between.

Freezing, by omission or in public

	A bank	A token contract
how a freeze works	the bank **stops executing** your orders	someone **calls a function** that blocks your address
who sees it	you, and the bank	everyone, on the chain
what the code allows	whatever the bank decides	only what the contract says

USDT and USDC have such functions. Euro stablecoins issued by banks will probably have them too.

The freeze in USDT's code

```
mapping (address => bool) public isBlackListed;

function addBlackList (address _evilUser) public onlyOwner {
    isBlackListed[_evilUser] = true;
    AddedBlackList(_evilUser);
}
function destroyBlackFunds (address _blackListedUser) public onlyOwner {
    require(isBlackListed[_blackListedUser]);
    uint dirtyFunds = balanceOf(_blackListedUser);
    balances[_blackListedUser] = 0;
    _totalSupply -= dirtyFunds;
}
function transfer(address _to, uint _value) public whenNotPaused {
    require(!isBlackListed[msg.sender]);
    ...
```

Tether can **freeze** an address, and also **destroy** its balance. TetherToken on Ethereum

A power that cannot be stolen

Binance’s stolen funds partly became USDT, and Tether froze them. Good or bad?

- Good this time; but the power can be **abused** by its holder.
- If the owner’s key is **stolen**, the thief can freeze or destroy balances, but cannot **transfer** them to itself.
- The power can be narrowed further: freeze only, or freeze for **at most a month**.

Code lets us state **precisely** what an intervention may do.

The freeze in USDC's code

```
address public blacklister;

modifier notBlacklisted(address _account) {
    require(!_isBlacklisted(_account), "Blacklistable: account is blacklisted");
    _;
}
function blacklist(address _account) external onlyBlacklister {
    _blacklist(_account);
    emit Blacklisted(_account);
}
function updateBlacklister(address _newBlacklister) external onlyOwner { ... }
```

- A separate role, the **blacklister**, appointed by the owner.
- USDC is a **proxy**: the code behind the address can be replaced. FiatTokenProxy · FiatTokenV2_2

A vault, written in class

```
contract RecoverableVault {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    receive() external payable {
        require(msg.sender == owner, "Only owner can deposit.");
    }

    function withdraw() external {
        require(msg.sender == owner, "Only owner can withdraw");
        payable(msg.sender).transfer(address(this).balance);
    }
}
```

If the owner loses the key, the funds are locked.

A safekeeper

A second address, chosen by the owner, that can **name a new owner**.

```
address public safeKeeper;

function setSafeKeeper(address _safeKeeper) external {
    require(msg.sender == owner);
    safeKeeper = _safeKeeper;
}

function changeOwner(address _newOwner) external {
    require(msg.sender == safeKeeper);
    owner = _newOwner;
}
```

The safekeeper can make itself the owner, at any time, and so can anyone who steals its key.

Limiting the safekeeper

Idea from the class	Problem
several safekeepers, who must agree	more people to coordinate, and to trust
a password, written in the contract	the contract is **public**: everyone can read it
the **hash** of a password	lost with the owner; revealed when used
a delay since the owner’s last sign of life	the right delay

A dead man’s switch

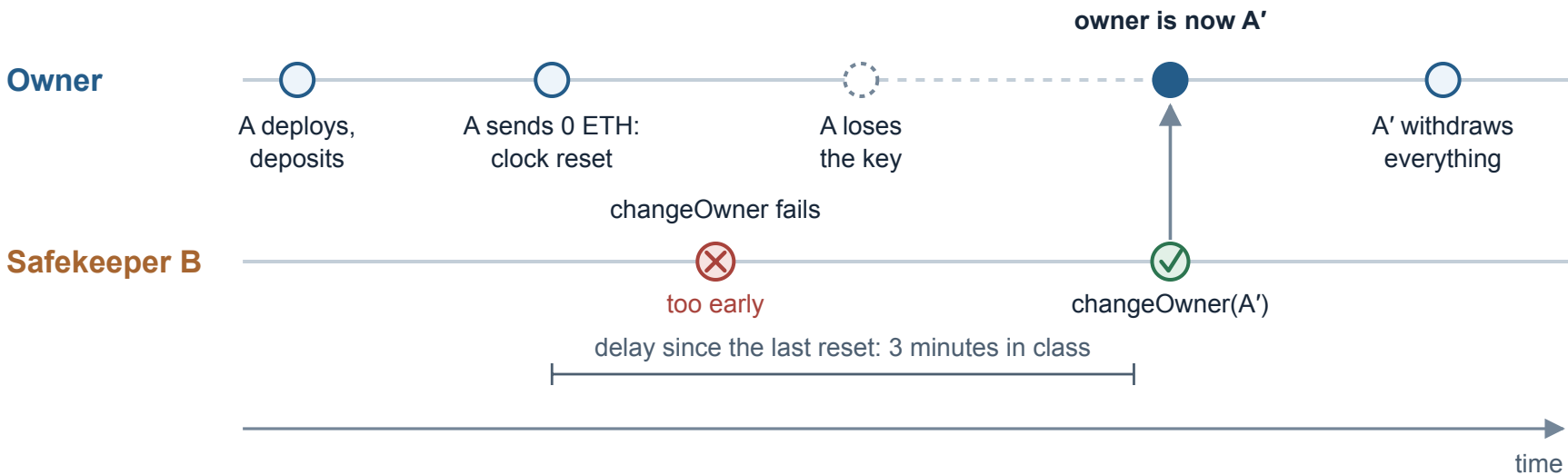
```
uint256 public lastReset;          // set to block.timestamp in the constructor

receive() external payable {
    require(msg.sender == owner, "Only owner can deposit.");
    lastReset = block.timestamp;   // any deposit, even of 0 ether, resets the clock
}

function changeOwner(address _newOwner) external {
    require(msg.sender == safeKeeper);
    require(block.timestamp >= lastReset + 3 minutes);
    owner = _newOwner;
}
```

The safekeeper can act only if the owner has been **silent** for the whole delay.

Key rotation, step by step



From transfer to call

```
payable(msg.sender).transfer(amount);                    // so far

(bool ok, ) = payable(msg.sender).call{value: amount}("");  // from now on
require(ok, "Payment failed");
```

	`transfer`	`call`
gas given to the recipient	2,300: enough to accept, not to record anything	all that is left
if the payment fails	everything reverts	returns `false`: check it
a recipient that is a contract, e.g. a Safe	may fail	works, if it accepts

The vault, tidied up with `call`: /14/recoverable-vault

Challenge: a check-in

During the session the lecturer created a new wallet and asked for test ether.

Send **any amount, even 0 ether**, to

`0x9c8d1cb0C7d6eAD3141CFf8e9B95a610B86345dD`

from the address in the class sheet.

Next session, 9 November

- **Deploy** the vault and rotate a key with a safekeeper.
- From a vault to an **account**: a stable address, keys that can change.
- **Safe**: owners, threshold, modules.
- When several signatures are not enough: **Bybit**.
- And the **chargeback**: can a contract offer one?