Security, Tokens, Wallets

Janice Ng
Minxing Chen

BLOCKCHAIN
AT BERKELEY

LECTURE OUTLINE

1. SECURITY OVERVIEW
2. ATTACKS/VULNERABILITIES
3. DEFENSES/MITIGATIONS
4. TOKENS & WALLETS

1 SECURITY OVERVIEW

1.1 OVERVIEW

WHY IS PUBLIC BLOCKCHAIN SECURITY HARD?

- ANYONE CAN:
  - Call whatever functions of it they want
  - See the bugs or vulnerabilities

- Incentive Structure:
  - Asks users to find and exploit your code to maximise gain

WHY SHOULD YOU CARE?

WHY DOES SECURITY MATTER?

- Immutable systems make it so that exploits are often extremely difficult to reverse the effects of
- Without a legal system in place, security is the only thing that prevents network adversaries from destroying or bankrupting your protocol
- Has a very strong tie to how much people trust your product

1.2 HISTORY

DAO HACK

What is DAO (Decentralized autonomous organization)

- Virtual venture capital (VC) fund that is governed by the investors of the DAO
- Main Idea:
  - Token holders can become contractors by submitting proposals for funding of their project by using the DAO funds
  - Proposals are approved by a quorum of 20% of all tokens
    - Transfers Ether automatically to the Smart Contract that represents the proposal
  - Raised 12.7 mil. Ether (150 mil USD in June 2016)
  - BUT...they got **H A C K E D**.

DAO HACK

What was the HACK?

“SPLIT PROCEDURE”, THE CATALYST OF THE HACK.

- The creators of DAO implemented the ability of a DAO to be split into 2
- “Minority” = able to retrieve funds when a proposal they objected gets approved
  - At least 48 days before getting it in an account YOU control

THE HACK:

- Coder found a loophole in this procedure
  - Once a split function is called, the code was written in a way to **retrieve ether → update balance**
  - Recursively call the “split procedure” and retrieve their funds **multiple times** before getting to the step where the **code checked the balance**
  -

RESULT OF THE HACK:

- On June 16, 2016, the attacker retrieved **~3.6 million Ether = 150 mil. USD**
- Now known as, the **“recursive call exploit”**

PARITY

WHAT IS PARITY

PARITY?!

- Ethereum client that provides a web interface for underlying Ethereum node software
- Access the basic Ether and token wallet functions
- Integrate seamlessly with all standard tokens & manage Ether transfers

...but what went wrong?

PARITY

CHECK PERMISSIONING

```
// constructor - just pass on the owner array to the multiowned and  // the
limit to daylimit
function initWallet(address[] _owners, uint _required, uint _daylimit) {
  initDaylimit(_daylimit);
  initMultiowned(_owners, _required);
}
```

- Attacker discovered that the default function routed all calls to the Wallet Library, which had an **unpermissioned init Wallet function**
  - Allowed *anyone* to freely change wallet ownership: change owner rights and able to kill it
  - Freeze 573 wallets, with balances of more than 500K Ether (~152 mil USD)

AUTHOR: NISHAAD NAVKAL
EDITED: JANICE NG

2 ATTACKS & VULNERABILITIES

SOURCE1, SOURCE2 FOR THIS SECTION

OVERFLOW

LIKE A METER

An overflow is when a number gets **incremented above** its **maximum** value.

- **Solidity** can handle up to 256 bit numbers (up to 2^{256}-1), incrementing by 1 would lead to an overflow:

```
  0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
+ 0x0000000000000000000000000000000000000001
-------------------------------------------
= 0x0000000000000000000000000000000000000000
```

AUTHOR: JENNY CONG

UNDERFLOW

FROM NONE TO THE MAX POSSIBLE

Likewise, in the inverse case, when the number is **unsigned**, **decrementing** will underflow the number, resulting in the **maximum possible value (255)**

```
  0x0000000000000000000000000000000000000000
- 0x0000000000000000000000000000000000000001
-------------------------------------------
= 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

AUTHOR: JENNY CONG

VISIBILITY OVERVIEW

ACCESS

Public functions can be called by anyone:

- Functions inside the contract, from inherited contracts
- Outside users

External functions can only be accessed externally by other contracts

- cannot be called by other functions of the contract/not visible to the contract itself
- cheaper to use because it uses the calldata opcode while public needs to copy all arguments to memory

Private functions can be called only from inside the contract

Internal functions are a more relaxed version of private functions

- Contracts which inherit from the parent are able to use the the function

Conclusion: Keep your functions private or internal unless there is a need for outside interaction.

-

REENTRANCY ATTACK

SPENDING BEFORE LOGGING

- Consider the following:
  - I have $500 in the bank.
  - I ask the bank to give me my $500.
  - The bank checks and see that my balance is sufficient.
  - The bank hands me $500.
  - The bank deduct $500 from my balance.
  - I walk away happily.

Is there anything wrong?

REENTRANCY ATTACK

SPENDING BEFORE LOGGING

- Now consider the following:
  - I have $500 in the bank.
  - I ask the bank to give me my $500.
  - The bank checks and see that my balance is sufficient.
  - The bank hands me $500.
  - I ask the bank to give me my $500.
  - The bank checks and see that my balance is sufficient.
  - .....repeats 100 times.
  - The bank deduct $500 from my balance.
  - I walk away very happily with $50,000.

REENTRANCY ATTACK

SPENDING BEFORE LOGGING

- Here's the equivalent code in Solidity:

```
function withdraw(uint _amount) public {
 if(balances[msg.sender] >= _amount) {
   if(msg.sender.call.value(_amount)()) {
     _amount;
   }
   balances[msg.sender] -= _amount;
 }
}
```

2.2 SOLIDITY QUIRKS

RANDOMNESS

DIFFICULT TO ACHIEVE

- Everything on the Ethereum Virtual Machine (EVM) is deterministic
- People have tried to use now and block.blockhash for business logic in a contract
  - However, results are predictable and can be manipulated by miners

“Timestamp dependence”

- Block numbers and average block time can be used to estimate time, but this does not act as future proof as block times may change

- How can I securely generate a random number in my smart contract?

PRAGMA STATEMENTS

- Make sure to lock contract solidity versions
- New versions always come out that may introduce some new vulnerabilities if contract isn't locked to one version

```
// dangerous
pragma solidity ^0.5.0;

// good practice
pragma solidity 0.5.0;
```

AUTHOR: Matt Peng

TOOLS TO HELP

A WHOLE NEW WORLD (OF TOOLS)

- Solidity is actually taking steps to introduce semantic checking of code
- Securify.ch is a static analysis tool for Smart Contracts.
- Hydra is a “framework for cryptoeconomic contract security, decentralized security bounties”
- Porosity is a “decompiler for Blockchain-based Ethereum Smart-Contracts”
- Manticore is a dynamic binary analysis tool with EVM support
- Remix performs static analysis to your code

Chris
@ethchris
Follow

#Solidity's new #SMT checker tells me that 23447 is a #prime and 23449 is not (23449 = 131 * 179) #ethereum #formalverification

```
> cat smt.sol
pragma experimental SMTChecker;
pragma solidity ^0.4.0;

contract C {
  function prime(uint a, uint b) public pure {
    require(1 < a && a < 500);
    require(1 < b && b < 25000);
    assert(a * b != 23447);
  }
  function factor(uint a, uint b) public pure {
    require(1 < a && a < 500);
    require(1 < b && b < 25000);
    assert(a * b != 23449);
  }
}

> solc smt.sol
Warning: This is a pre-release compiler version, please do no
smt.sol:1:1: Warning: Experimental features are turned on. Do
pragma experimental SMTChecker;
^-----------------------------^
smt.sol:13:5: Warning: Assertion violation happens here for:
  a = 179
  b = 131

    assert(a * b != 23449);
    ^--------------------^
```

OPENZEPPELIN RECAP

Don’t Forgot Open Source Libraries

What is OpenZeppellin?

- Zeppelin is a library for writing secure smart contracts on Ethereum.

What kind of contracts are available?

- Ownership: declaring, claiming transferring
- Safe Math: takes care of integer overflows and overdrawing
- Push and Pull: deposit and withdraw methods you can trust
- Standard and Basic Tokens: ERC20 implementable and ready out of the box
- GitHub: https://github.com/OpenZeppelin/zeppelin-solidity
- Article: https://blog.zeppelin.solutions/onward-with-ethereum-smart-contract-security-97a827e47702

3 MITIGATIONS/ DEFENSES

WHAT CAN I DO TOWARDS SECURITY

FOR SMART CONTRACTS

- Don’t write fancy code
- Code adversarially, the contract should take before giving
- Use audited and tested code
- Write as many unit tests as possible
- Prepare for failure - at any moment, in any contract or method
- Rollout carefully - bug bounties before ICO
- Keep contracts simple - more complexity = more attack vectors
- Keep updating with software and community
- Beware of blockchain properties: public vs. private, .send() vs. .call()

EXTERNAL CALLS

Takeaway: Avoid external calls when possible

- What is an external call?
  - A call from one contract to another untrusted contract or account.
  - **delegatecall, callcode, call**
- Previous (Expensive) Bugs:
  - The Dao hack
  - The Parity multisignature wallet hack

PULL > PUSH

How do we get data/value to addresses?

- **Push**
  - Contract transfers immediately to addresses after logic
  - **Potentially vulnerable if recipient contract is poisoned**
- **Pull (!!)**
  - “Lazy”, give addresses the ability to get their data or money
  - Like a withdraw function
  - **Use these**

ASSERT AND REQUIRE

Takeaway: Use assert and require properly

- `require(condition)` is meant to be used for input validation
  - Should be done on any user input, and throws if condition is false.
- `assert(condition)` also throws if condition is false
  - Should be used only for internal errors or to check if your contract has reached an invalid state.
- By following this paradigm it would become possible to use a formal analysis tool, that verifies that the invalid opcode can never be reached, for the absence of errors assuming valid inputs.

AUTHOR: COLLIN CHIN
https://github.com/ConsenSys/smart-contract-best-practices

ROUNDING WITH INT DIVISION

Takeaway: All integer division rounds down to the nearest integer.

If you need more precision, consider using a multiplier, or store both the numerator and denominator.

```
// bad
uint x = 5 / 2; // Result is 2, all integer division rounds DOWN to the nearest integer

// good
uint multiplier = 10;
uint x = (5 * multiplier) / 2;

uint numerator = 5;
uint denominator = 2;
```

AUTHOR: COLLIN CHIN
https://github.com/ConsenSys/smart-contract-best-practices

ALL DATA IS PUBLIC

ATTENTION! ATTENTION! ATTENTION!

Takeaway: Remember that on-chain data is public

- Many applications require submitted data to be private up until some point in time in order to work.
- If you are building an application where privacy is an issue, take care to avoid requiring users to publish information too early.

Example: Rock - Paper - Scissors

1. require both players to submit a hash of their intended move first
2. require both players to submit their move;
3. if the submitted move does not match the hash throw it out.

2-PARTY / N-PARTY CONTRACTS

Takeaway: beware of the possibility that some participants may "drop offline" and not return

- Do not make refund or claim processes dependent on a specific party performing a particular action with no other way of getting the funds out.

Example: Rock - Paper - Scissors

- Common Mistake: Not make a payout until both players submit their moves;
- Pitfall: a malicious player can "grief" the other by simply never submitting their move
- Solutions:
    1. Provide a way of circumventing non-participating participants. Ex: time limit
    2. Add an additional economic incentive for participants to submit information in all of the situations in which they are supposed to do so. Ex: deposit stake and slash

ALWAYS REMEMBER TESTING

- As seen from all these possible attacks and vulnerabilities, it is vital to always try and get 100% coverage of testing
- What this means is that all functions and possible interactions within the contract are completely tested for

```
Compiling ConvertLib.sol...
Compiling MetaCoin.sol...
Compiling Migrations.sol...


  Contract: MetaCoin
    ✓ should put 10000 MetaCoin in the first account (107ms)
    ✓ should call a function that depends on a linked library   (109ms)
    ✓ should send coin correctly (153ms)


  3 passing (4s)
```

Author: Matt Peng

SECURITY QUIZ TIME

:)

https://tinyurl.com/vx9he9e

Author: JANICE NG

4 TOKENS

BITCOIN, ETHER, AND COINS

OH MY!

- Tokens are not currency, you don't need to create a blockchain to build a token
- Coins that are currencies include Bitcoin, native to the Bitcoin Network, and Ether native to the Ethereum chain
- Rather, the token is an entity with value associated with it
- Most companies, when they perform ICOs, associate tokens as objects of shares of stock of their company
- The beauty of tokens is that they can easily be built and have very standard smart contract logic behind them

AUTHOR: MATT PENG

BUILDING A TOKEN

HARD CAN IT BE?

- The most important distinction is knowing the true purpose of tokens and why it might be great to build them
- Tokens can act as symbols or values for smart contracts or dApps, playing a vital role in the logic of these applications
- The cycle of tokens is as follows:
  - Creation
  - Purchase of token
  - Transactions with tokens

AUTHOR: MATT PENG

4.1 FUNGIBLE TOKENS

STANDARD ERC20 TOKEN

A coin for a coin...

- We will be building tokens on the Ethereum platform
- ERC20 tokens are a type of tokens standardized by Ethereum and can easily be built
- They are considered fungible since one token will always be worth the same as any other token
- All that is necessary to build this token is to implement the ERC20 interface

ERC20 INTERFACE

WHY DIDN'T THEY THINK OF THIS EARLIER

The ERC-20 defines a common list of rules for all Ethereum tokens to follow, meaning that this particular token empowers developers of all types to accurately predict how new tokens will function within the larger Ethereum system.

```
contract ERC20Interface {
  function totalSupply() constant returns (uint256 totalSupply); function balanceOf(address
  _owner) constant returns (uint256 balance); function transfer(address _to, uint256 _value)
  returns (bool success);

  function transferFrom(address _from, address _to, uint256 _value) returns(bool success);
  function approve(address _spender, uint256 _value) returns (bool success);
  function allowance(address _owner, address _spender) constant returns (uint256 remaining);
  event Transfer(address indexed _from, address indexed _to, uint256 _value);
  event Approval(address indexed _owner, address indexed _spender, uint256 _value);
}
```

AUTHOR: NICK ZOGHB

```
contract ERC20Interface{
  // Get the total token supply
  function totalSupply() constant returns (uint256 totalSupply);

  // Get the account balance of another account with address _owner function
  balanceOf(address _owner) constant returns (uint256 balance);

  // Send _value amount of tokens to address _to
  function transfer(address _to, uint256 _value) returns (bool success);

  // Send _value amount of tokens from address _from to address _to
  function transferFrom(address _from, address _to, uint256 _value) returns (bool success);

  // Allow _spender to withdraw from your account, multiple times, up to the _value amount.
  // If this function is called again it overwrites the current allowance with _value.
  // this function is required for some DEX functionality
  function approve(address _spender, uint256 _value) returns (bool success);

  // Returns the amount which _spender is still allowed to withdraw from _owner
  function allowance(address _owner, address _spender) constant returns (uint256 remaining);

  // Triggered when tokens are transferred.
  event Transfer(address indexed _from, address indexed _to, uint256 _value);

  // Triggered whenever approve(address _spender, uint256 _value) is called.
  event Approval(address indexed _owner, address indexed _spender, uint256 _value);
}
```

ERC20 EXTRA FRAGMENTS

NECESSARY ADDITIONS

```
contract TokenContractFragment {
  // Constructor
  function TokenContractFragment(_uint amount) {
    // critical code to execute on deployment
  }
  // Balances foreach account
  mapping(address => uint256) balances;
  // Owner of account approves the transfer of an amount to another account
  mapping(address => mapping (address => uint256)) allowed;
}
```

AUTHOR: NICK ZOGHB

ERC20 HUMAN STANDARD

HUMAN READABILITY IS IMPORTANT

```
contract TokenContractFragment {
  // Human standard token modification
  string public constant name = "Token Name";
  string public constant symbol = "SYM";
  uint8 public constant decimals = 18;  // 18 is the most common number of decimal places
}
```

AUTHOR: NICK ZOGHB

TOKEN TOOLS

SIMPLE BUILDING

- Even with all the functions reqreuied outlined, building a token can be made even easier with online open source resources
- The community has come together to create commnity-backed smart contracts in the Open Zeppelin tool
- Here, there is code implementing many different ERC tokens that can be reused in anyone's personal projects
- With Open Zeppelin, tokens can be built in minutes without having to sweat out too much of the coding part

OpenZeppelin

4.2 NONFUNGIBLE TOKENS

STANDARD ERC721 TOKEN

CRYPTOKITTIES

- Another type of token is the ERC721 token that Ethereum supports as a standardized token
- This token is nonfungible in that one token is not the same value
- Some tokens are worth more than others, think trading cards etc.
- The best example here is Cryptokitties
- Some kitties are worth more than others even though each kitty is still one token
- Implementing ERC721 tokens are very simple and just like before, require implementing an interface

ERC721

MORE LIKE ASSETS THAN TOKENS

Created: 2017-09-20, the “non-fungible” token model.

[Original ERC721 issue](), [implementation example]()

AUTHOR: NICK ZOGHB

```
contract ERC721 {

  // ERC20 compatible functions

  function name() constant returns (string name);
  function symbol() constant returns (string symbol);
  function totalSupply() constant returns (uint256 totalSupply);
  function balanceOf(address _owner) constant returns (uint balance);

  // Functions that define ownership

  function ownerOf(uint256 _tokenId) constant returns (address owner);
  function approve(address _to, uint256 _tokenId);
  function takeOwnership(uint256 _tokenId);
  function transfer(address _to, uint256 _tokenId);
  function tokenOfOwnerByIndex(address _owner, uint256 _index) constant returns (uint tokenId);

  // Token metadata

  function tokenMetadata(uint256 _tokenId) constant returns (string infoUrl);

  // Events

  event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
  event Approval(address indexed _owner, address indexed _approved, uint256 _tokenId);

}
```

5 WALLETS

5.1 WALLET BASICS

WHAT IS A WALLET?

AT A HIGH LEVEL

Definition: A wallet is a user interface for users to manage tokens or cryptocurrencies

Usages:

A wallet allows a user to:

- control access to a user's token
- manages keys and addresses
- tracks the balance
- creates and signs transactions

WHAT IS A WALLET?

AT A LOW LEVEL

Definition: A wallet is a data structure that stores and manages a user's public/private keys

Usages:

A wallet allows a user to:

- control access to a user's token
- manages keys and addresses
- tracks the balance
- creates and signs transactions

How:

- Private key allows **sending** money out of the wallet
- Public key (wallet address) allows **receiving** money to the wallet
- Transactions can be viewed on the blockchain using the public key

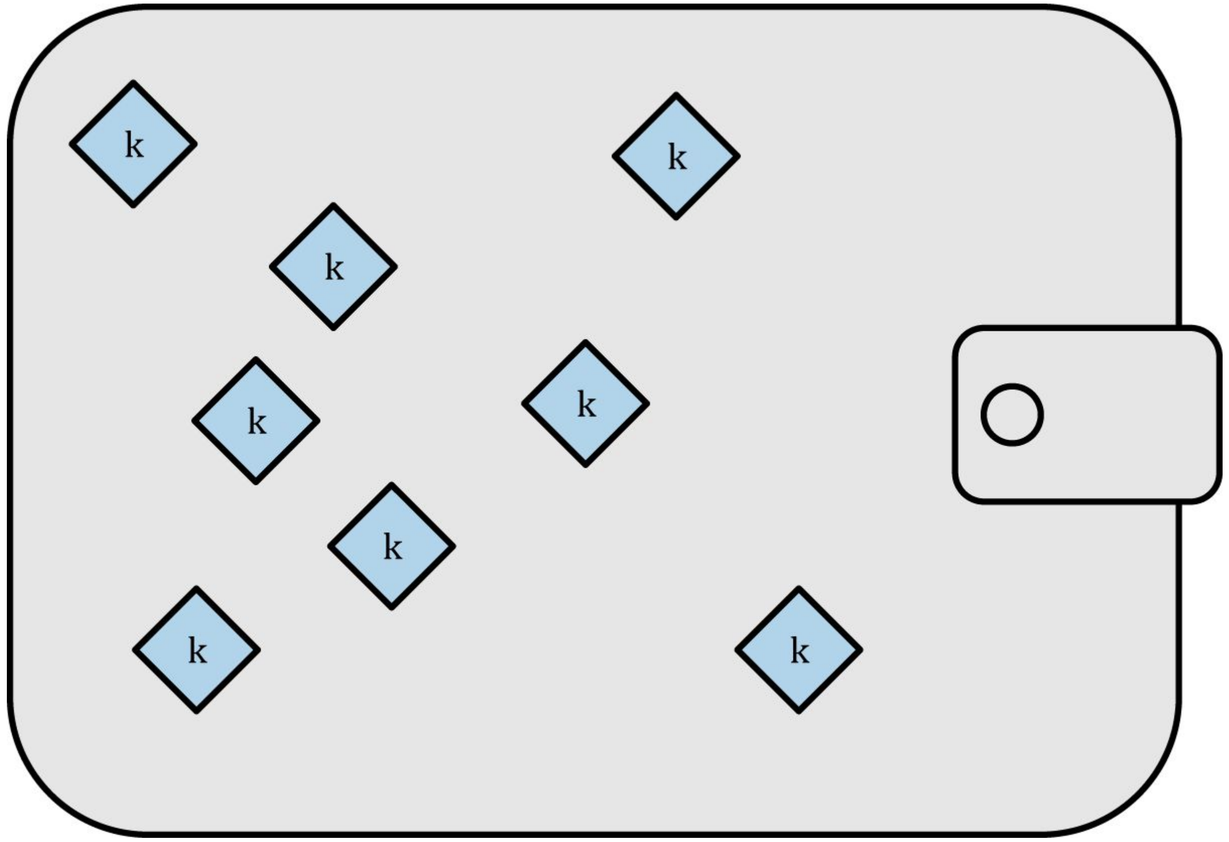
AUTHOR: JENNY CONG

5.2 TYPES OF WALLETS

SOURCE FOR THIS SECTION

NONDETERMINISTIC WALLETS

RANDOM



Nondeterministic wallet:

A collection of independently **randomly generated** keys

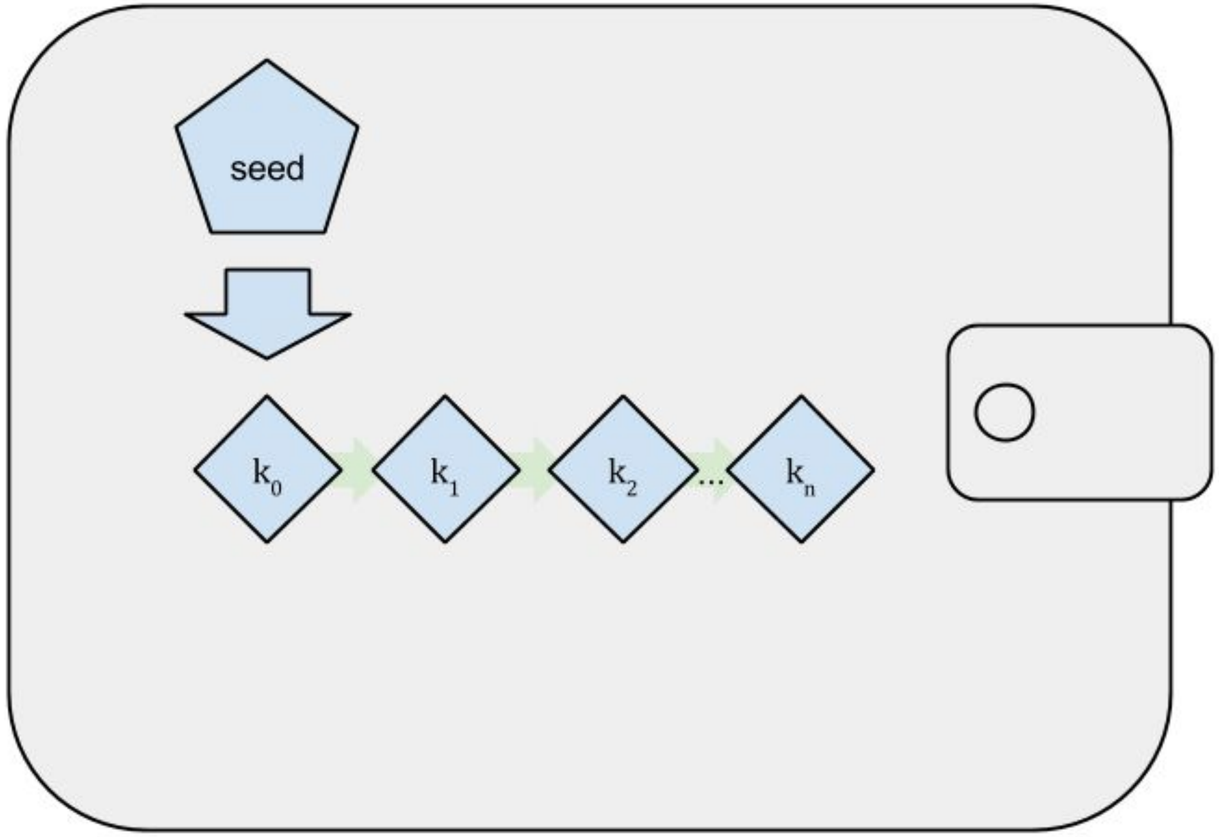
Disadvantages:

- cumbersome to manage, back up, and import
- Reduces privacy due to address reuse

AUTHOR: JENNY CONG

DETERMINISTIC WALLETS

SEEDED



Deterministic wallet:

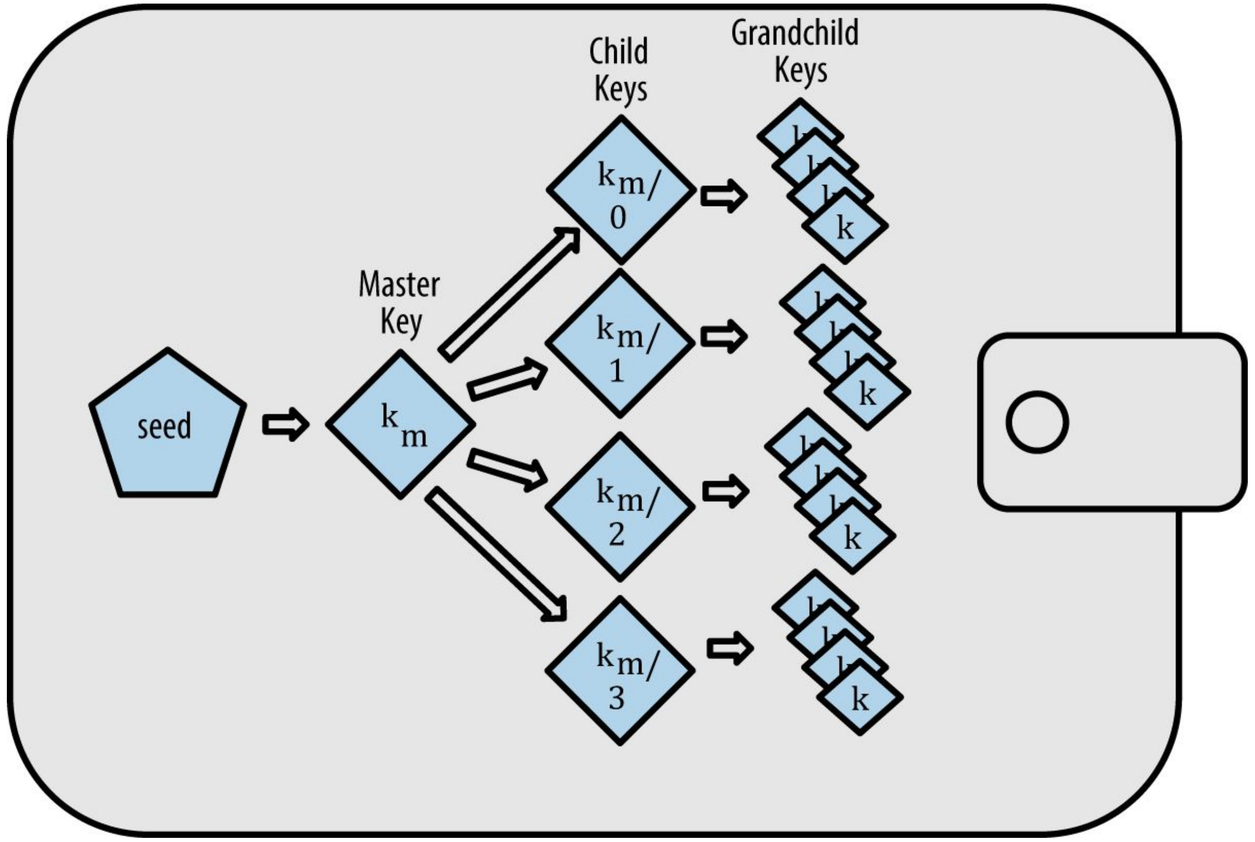
A deterministic sequence of keys **derived from a seed** using a hash function

Advantage:

- the seed is sufficient to recover all the derived keys -> efficient backup

DETERMINISTIC WALLETS

HD WALLETS



HD wallet:

A collection of keys derived in a **tree** structure

Advantages:

- Additional structural management: giving access to specific branch of keys
- users can create a sequence of public keys without having access to the corresponding private keys

AUTHOR: JENNY CONG

5.3 MULTISIG WALLETS

MOTIVATION

WHY MULTI-SIG?

- Contract Wallets can be setup as Multisig Wallets.
- Simple Wallets and Multisig Wallets are both examples of Contract Wallets.
- A Simple Wallet:
  - only one Account both creates and owns the wallet
- Multisig Wallet:
  - has several owner Accounts one of which will also be the creator Account.
  - M-of-N type wallets

THE BASICS

WHAT A MULTI-SIG WALLET NEEDS

- A list of **people** who are allowed to do things
- Rules on **how many** of those people **have to agree** before it happens
- A way to **receive** ether
- A way to **submit** a request
- A way to **agree** to a request (and submit it if you are last)
- A way to **resubmit** the request if it fails

USE CASES

WHY IS A MULTI-SIG WALLET NEEDED

- **Use case 1:** You own some ether and want to store it securely but are concerned that just relying on a single private key may not be safe. So you create a Multisig Wallet, a 2-of-3 wallet, but all the owner accounts are under your control, that is, you control all the private keys.

- **Use case 2:** You set up a Multisig Account jointly owned by say two people, you and Alice. So you have one private key and Alice the other.

Attendance

https://tinyurl.com/sp20-dev-decal-5