



SP2024 Week 05 • 2024-02-22

Blockchain

Jake Mayer

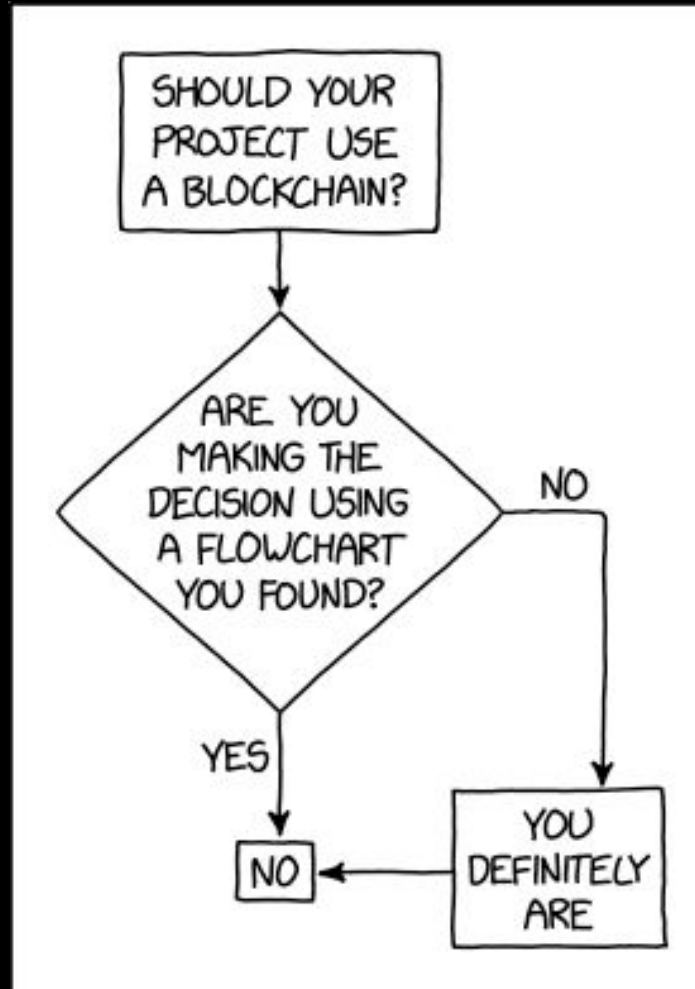
Announcements

- Social on the quad
 - This Sunday, February 25 at 3:00 PM
 - Playing frisbee and other games
- No meeting this Sunday
 - Come play frisbee instead



ctf.sigpwny.com

sigpwny{0verflow1ng_wit5_crypt0}



What is Blockchain?

- Traditionally, finance relies on trusted institutions
- What if we didn't need to trust anyone?
- Goals:
 - Decentralized
 - Cryptographic authorization
 - Prevent double-spend
 - Maintain state integrity (immutable and irreversible)
- Means:
 - Public/private key encryption
 - Distributed ledger
 - Consensus mechanism
 - Peer-to-peer network



What is Blockchain?

- Bitcoin:
 - First implementation of decentralized currency
 - Allows parties to transact Bitcoin using public key encryption
 - The state of the network is validated through cryptographic means
 - No more trusted parties
 - Introduced Proof-of-Work (PoW) consensus mechanism
 - Ex. I want to send money to pay for something
- Ethereum:
 - What if we could extend Bitcoin to turing-complete applications?
 - Allows parties to interact with smart contracts: NFTs, trading, governance
 - Ex. I want to vote on a governance proposal



Cryptography Review

- Public key cryptography:
 - Generate a keypair: public and private key
 - The public key is the party's "identity"
 - The private key is used to generate signatures
 - A signature verifies that the owner of the private key authorizes a message
- Hashing:
 - One-way function (arbitrary length input -> fixed length output)
 - Extremely difficult to find multiple inputs with the same output
 - Ex. SHA-256 (Bitcoin), KECCAK-256 (Ethereum)



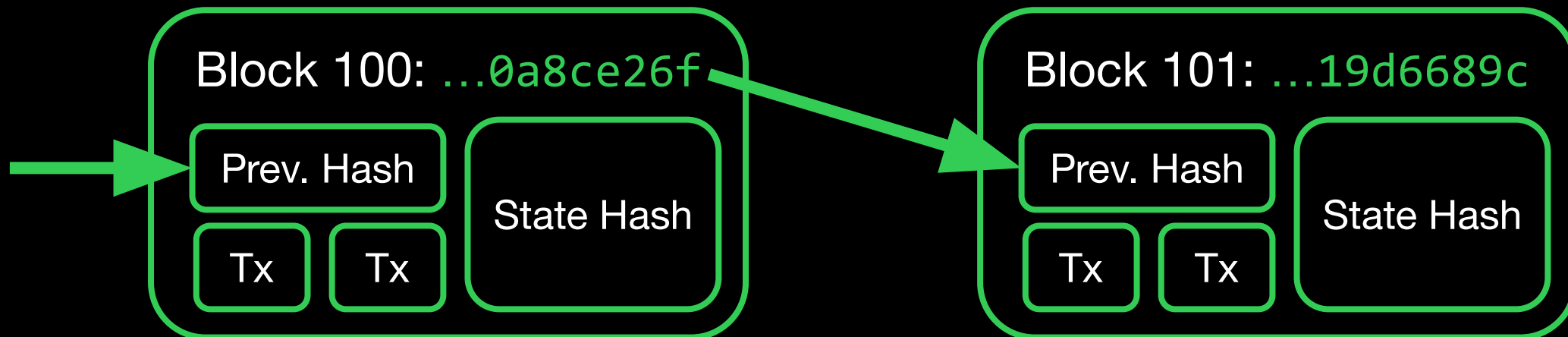
Transactions

- The basic unit for interacting with a cryptocurrency system
- Signed with the party's private key
- Authorized for the corresponding public key
- Modifies the state of the blockchain



Distributed Ledger

- Records the state of the system (ie. account balances, contract storage, etc.)
- Grouped into blocks of transactions
- Blocks are identified by their hash
- Blocks are chained by including the previous block's hash in the next hash calculation



Proof-of-Work Consensus

- How do we determine the state of the blockchain?
- Need a mechanism for determining the current/head block
- Rather than relying on trusted authority, we rely on computational power ("work")
- Remember the nonce?
- "Miners" brute-force a nonce value
- An attacker would need a majority of compute in the network to repeatedly insert malicious blocks
- Mining difficulty is adjusted dynamically



Peer-to-Peer Network

- This is great, but how can we distribute it?
- Participants find each other through discovery protocols
- Participants:
 - Share pending transactions
 - Pool pending transactions
 - Mine proof-of-work verification
 - On successfully finding proof-of-work, broadcast the block
 - Accept valid blocks as new state of blockchain



All Together

How can Bob send Alice \$10?

Bob 🧑

Create & Sign transaction:

Send \$10 to
Alice 🔑

Miner 🪓

Collect & Verify Transactions:

Tx

Tx

Alice 🧑

Waiting for \$10:

Block n: ...1be25635

Alice: \$0



All Together

How can Bob send Alice \$10?

Bob 🧑

Create & Sign transaction:



Miner 🪓

Collect & Verify Transactions:



Alice 🧑

Waiting for \$10:



All Together

How can Bob send Alice \$10?

Bob 🧑

Create & Sign transaction:

Send \$10 to
Alice 🔑

Miner 🪓

Mine proof-of-work nonce:

Block n+1: ...301c2363

Send \$10 to
Alice 🔑

Tx

Tx

Nonce 🪓

Alice 🧑

Waiting for \$10:

Block n: ...1be25635

Alice: \$0



All Together

How can Bob send Alice \$10?

Bob 🧑

Create & Sign transaction:

Send \$10 to
Alice 🔑

Miner 🪓

Broadcast New Block:

Block n+1: ...301c2363

Send \$10 to
Alice 🔑

Tx

Tx

Nonce 🪓

Alice 🧑

Receive \$10:

Block n: ...1be25635

Alice: \$0

Block n+1:

...301c2363
Alice: \$10



51% Attack

- The attacker has majority of the network's hash power
- Allows the attacker to outmine the network and control the blockchain
 - Double spending: Send transactions to one destination, then revert and spend elsewhere
 - Censor transactions
- Typically, regular mining incentives outweighs loss of value if currency becomes untrusted
- Expensive to execute



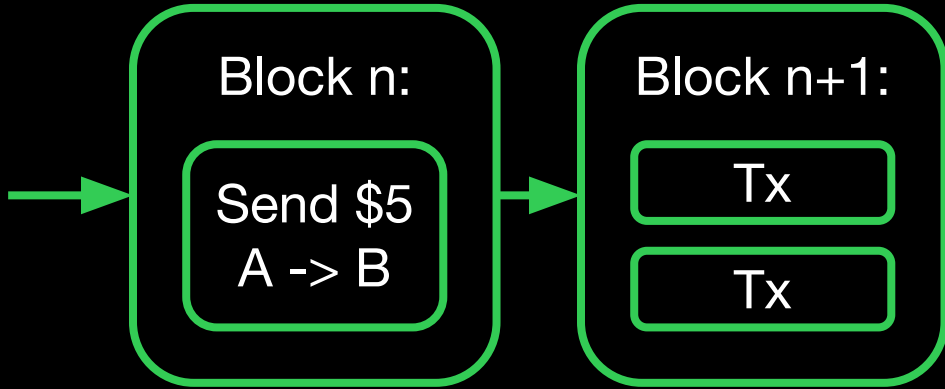
51% Attack

- Several historical attacks have occurred
- In one example, an attacker multiple-spent **\$17.5-18.6 million** worth of BTG by targeting cryptocurrency exchanges
- ETC has suffered numerous attacks including **\$millions** of double-spend due its decreasing popularity with miners
- Hash power can be rented through services like **NiceHash**
- Accessibility of hash-power-for-hire underscores the need for honest mining incentives (ie. block reward, transaction fees)



51% Attack

Public chain

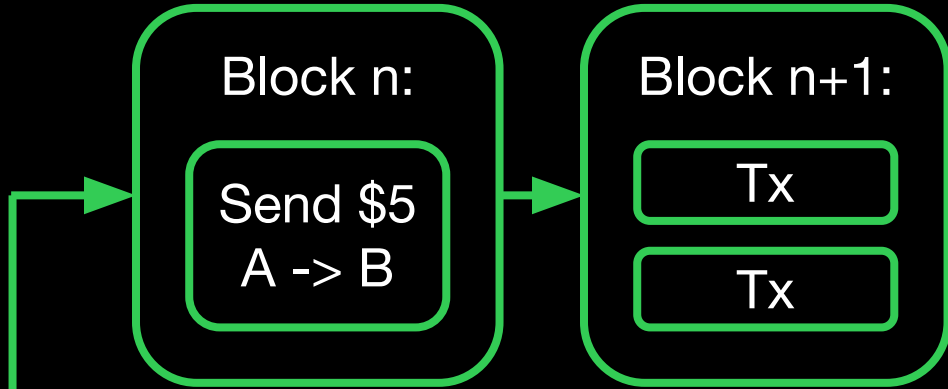


... Eventually, B sees that their \$5 has been confirmed for several blocks



51% Attack

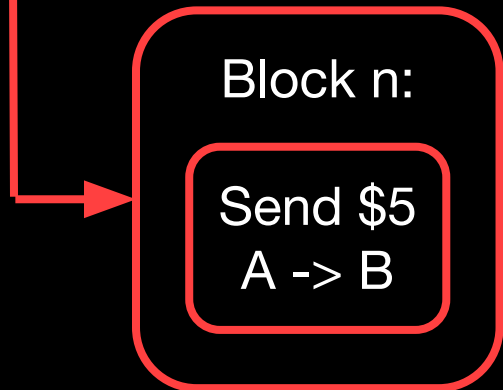
Public chain



... Eventually, B sees that their \$5 has been confirmed for several blocks

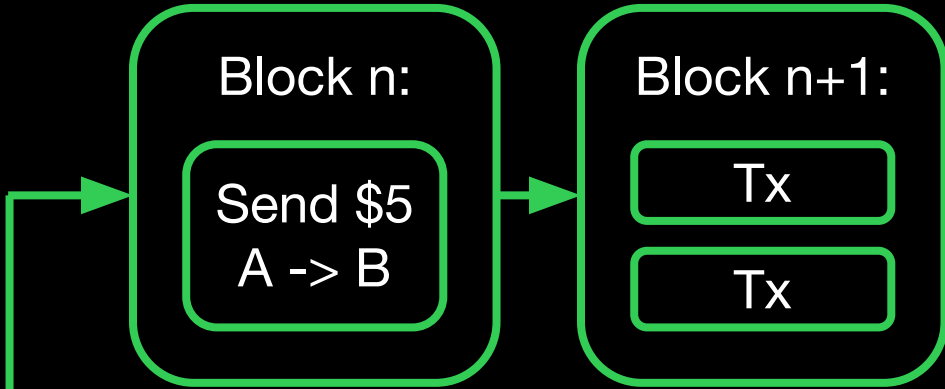
Plot twist! We've been mining in private...

Private chain



51% Attack

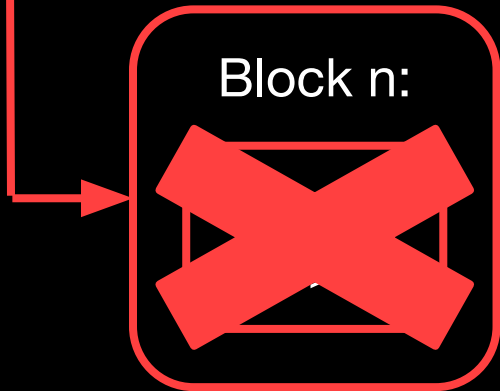
Public chain



... Eventually, B sees that their \$5 has been confirmed for several blocks

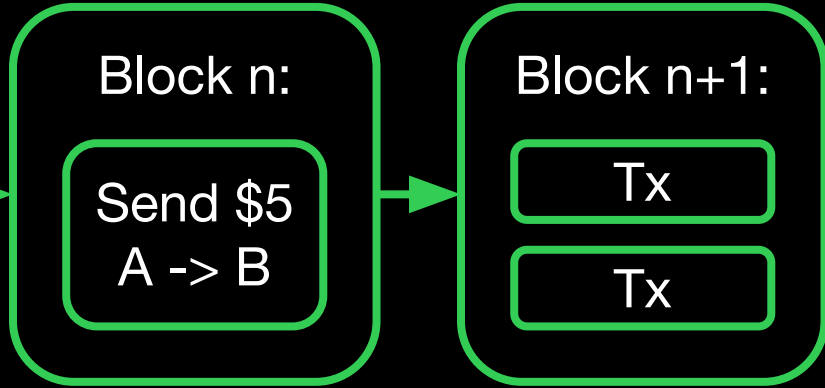
Plot twist! We've been mining in private...

Private chain



51% Attack

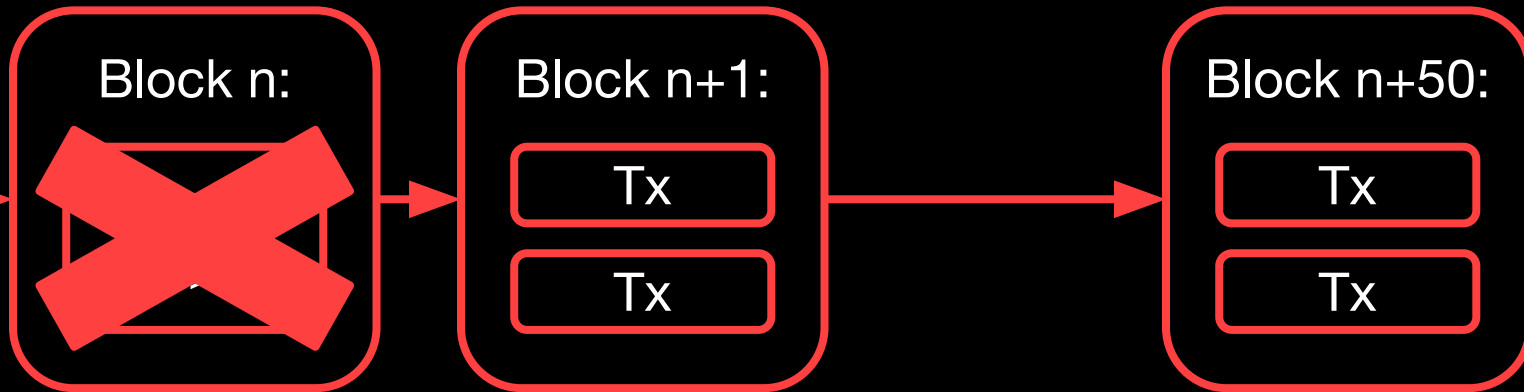
Public chain



... Eventually, B sees that their \$5 has been confirmed for several blocks

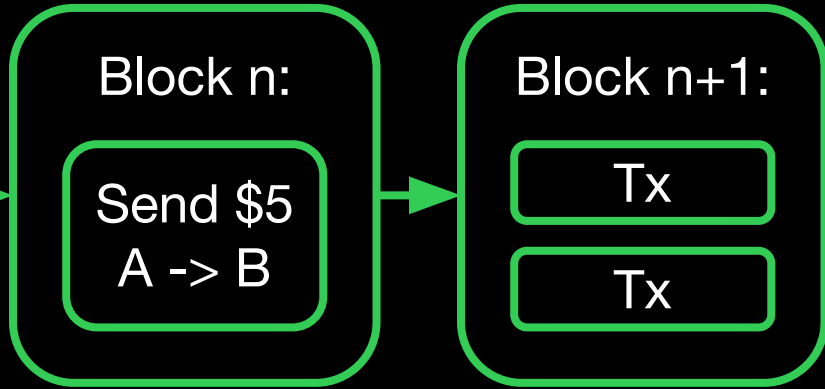
Plot twist! We've been mining in private...

Private chain



51% Attack

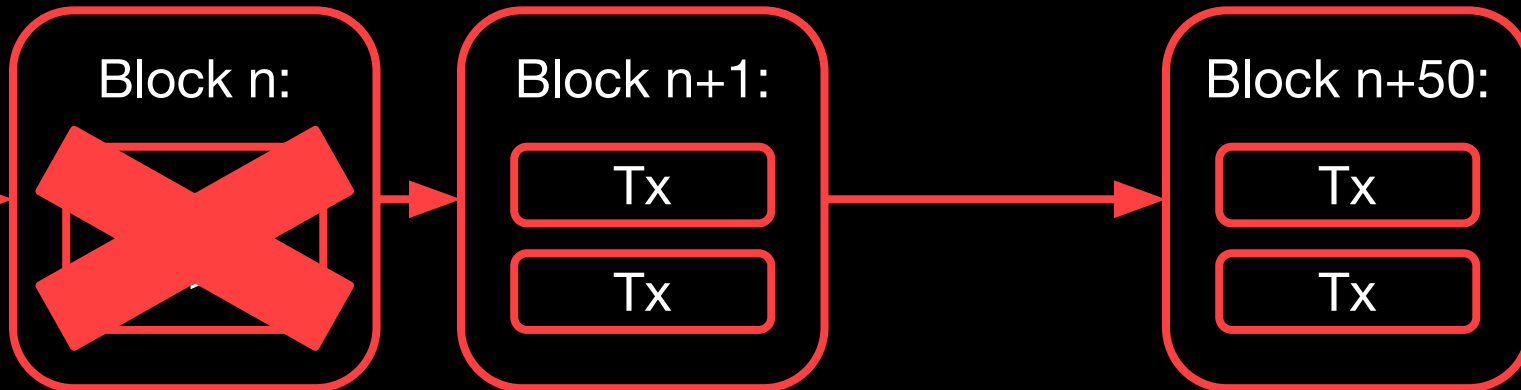
Public chain



... Eventually, B sees that their \$5 has been confirmed for several blocks

Plot twist! We've been mining in private...

Private chain

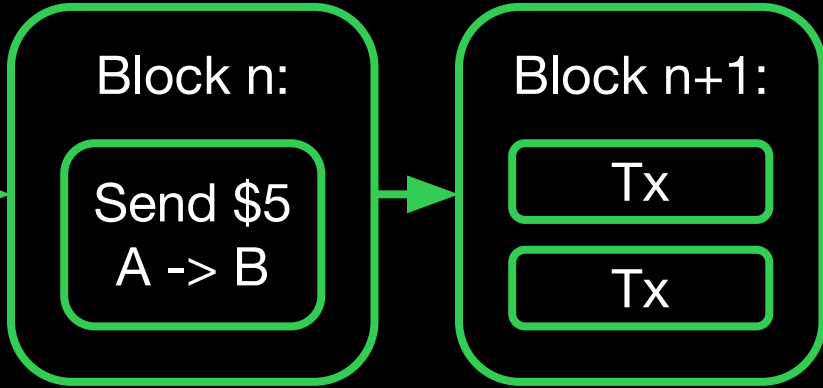


Broadcast private blockchain



51% Attack

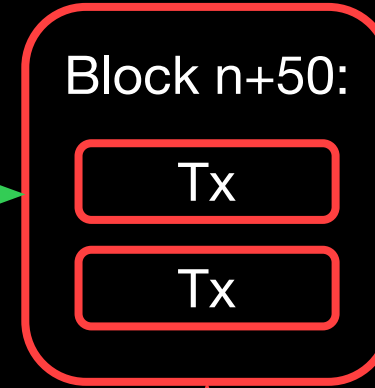
Public chain



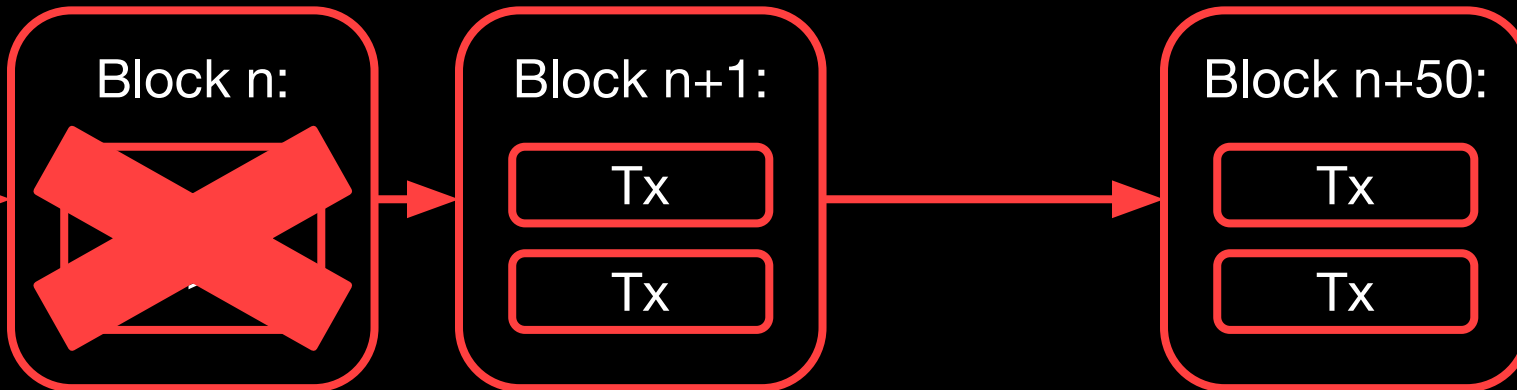
... Eventually, B sees that their \$5 has been confirmed for several blocks

Plot twist! We've been mining in private...

Network accepts longer chain excluding the spend



Private chain

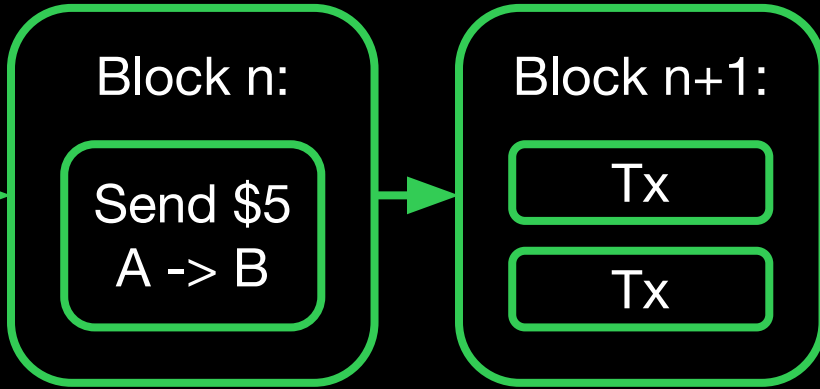


Broadcast private blockchain



51% Attack

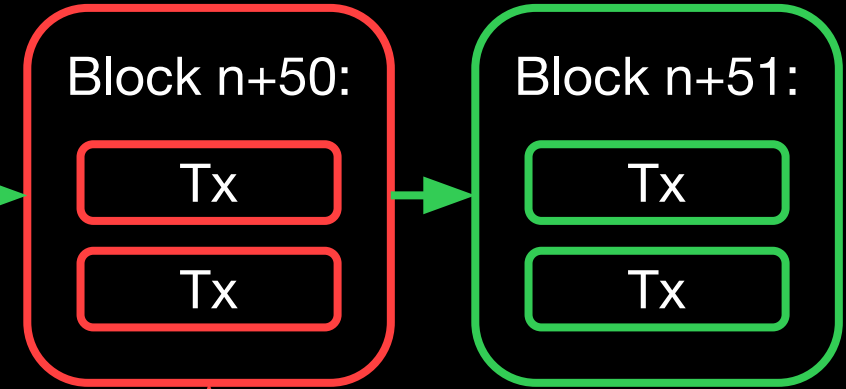
Public chain



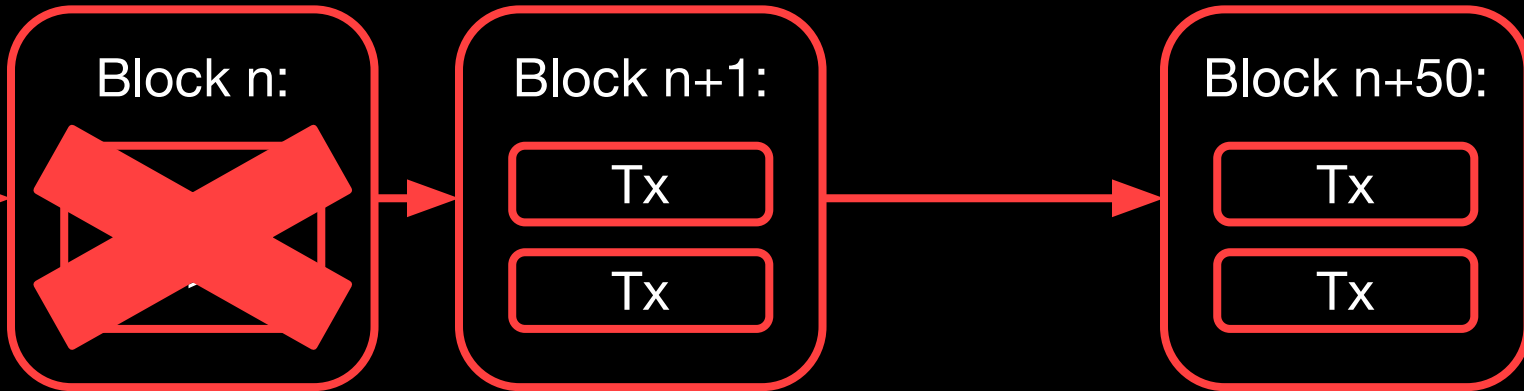
... Eventually, B sees that their \$5 has been confirmed for several blocks

Plot twist! We've been mining in private...

Network accepts longer chain excluding the spend



Private chain



Broadcast private blockchain



Sybil Attack

- Exploit the peer-to-peer network by creating many identities
- Allows an attacker to gain disproportionate control over the network
- Could allow Denial-of-Service (DoS) attacks
- Eclipse attack: Isolate nodes from the network by displacing legitimate nodes



Smart Contracts

- Programs stored and executed on the blockchain
- Interacted with through other smart contracts or directly with a transaction
- Can execute "contracts" without another party to oversee the transaction
- For Ethereum, these are often written in **Solidity**



Smart Contracts

```
contract Counter {  
    uint private count;  
  
    function get() public view returns (uint) {  
        return count;  
    }  
    function inc() public {  
        count += 1;  
    }  
    function dec() public {  
        count -= 1;  
    }  
}
```



Smart Contract Attacks

- Just like usual programs, smart contracts have typical software vulnerabilities:
 - Integer underflow
 - Logical bugs
 - Improper access control
- Due to the interactive nature of smart contracts, reentrancy vulnerabilities are common
- An attacking contract unexpectedly re-enters the victim contract after regaining execution from the victim contract
- Once your contract is exploited, you cannot rollback (the blockchain is immutable)



Reentrancy Attack

```
contract Bank {  
    mapping(address => uint) public balances;  
  
    function deposit() public payable {  
        balances[msg.sender] += msg.value;  
    }  
    function withdraw() public {  
        uint bal = balances[msg.sender];  
        require(bal > 0);  
        (bool sent, ) = msg.sender.call{value: bal}("");  
        require(sent, "Failed to send Ether");  
        balances[msg.sender] = 0;  
    }  
}
```



Reentrancy Attack

```
contract Bank {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }
    function withdraw() public {
        → uint bal = balances[msg.sender]; 100
        require(bal > 0);
        (bool sent, ) = msg.sender.call{value: bal}("");
        require(sent, "Failed to send Ether");
        balances[msg.sender] = 0;
    }
}
```



Reentrancy Attack

```
contract Bank {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }
    function withdraw() public {
        uint bal = balances[msg.sender];
        → require(bal > 0); 100 > 0
        (bool sent, ) = msg.sender.call{value: bal}("");
        require(sent, "Failed to send Ether");
        balances[msg.sender] = 0;
    }
}
```



Reentrancy Attack

```
contract Bank {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }
    function withdraw() public {
        uint bal = balances[msg.sender];
        require(bal > 0);
        → (bool sent, ) = msg.sender.call{value: bal}(""); bal=100
        require(sent, "Failed to send Ether");
        balances[msg.sender] = 0;
    }
}
```



Reentrancy Attack

```
contract Bank {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }
    function withdraw() public {
        uint bal = balances[msg.sender];
        require(bal > 0);
        → (bool sent, ) = msg.sender.call{value: bal}("");
        require(sent, "Failed to send Ether");
        balances[msg.sender] = 0;
    }
}
```



Reentrancy Attack

```
contract Bank {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }
    function withdraw() public {
        uint bal = balances[msg.sender];
        require(bal > 0);
        (bool sent, ) = msg.sender.call{value: bal}("");
        → require(sent, "Failed to send Ether");
        → balances[msg.sender] = 0;
    }
}
```



Reentrancy Attack

```
contract Bank {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }
    function withdraw() public {
        uint bal = balances[msg.sender];
        require(bal > 0);
        (bool sent, ) = msg.sender.call{value: bal}("");
        require(sent, "Failed to send Ether");
        balances[msg.sender] = 0;
    }
}
```



Reentrancy Attack

```
contract Bank {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }
    function withdraw() public {
        → uint bal = balances[msg.sender]; 100
        require(bal > 0);
        (bool sent, ) = msg.sender.call{value: bal}("");
        require(sent, "Failed to send Ether");
        balances[msg.sender] = 0;
    }
}
```



Reentrancy Attack

```
contract Bank {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }
    function withdraw() public {
        uint bal = balances[msg.sender];
        → require(bal > 0); 100 > 0
        (bool sent, ) = msg.sender.call{value: bal}("");
        require(sent, "Failed to send Ether");
        balances[msg.sender] = 0;
    }
}
```



Reentrancy Attack

```
contract Bank {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }
    function withdraw() public {
        uint bal = balances[msg.sender];
        require(bal > 0);
        → (bool sent, ) = msg.sender.call{value: bal}(""); bal=100
        require(sent, "Failed to send Ether");
        balances[msg.sender] = 0;
    }
}
```



Reentrancy Attack

```
contract Bank {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }
    function withdraw() public {
        uint bal = balances[msg.sender];
        require(bal > 0);
        → (bool sent, ) = msg.sender.call{value: bal}("");
        require(sent, "Failed to send Ether");
        balances[msg.sender] = 0;
    }
}
```



Reentrancy Attack

- In 2016, an ethereum smart contract was exploited for **3.6 million** ETH through a reentrancy attack
- Controversially, the Ethereum community created a "hard fork" to revert the losses



Exploring Ethereum

- Tools like [Etherscan](#) allow you to explore the state of the blockchain from your browser
 - Useful for OSINT
- [Remix](#) is a web IDE that allows you to develop and test smart contracts
- [Metamask](#) manage your crypto wallets from the browser
 - Integrates with Remix, allowing you to send transactions to live blockchains or testnets



Learning Resources

- Ethereum [docs](#) and [whitepaper](#)
- Learn about [EVM](#) (how ethereum contracts are executed)
- Solidity [docs](#)



Next Meetings

2024-02-25 • This Sunday

- Frisbee!
- On the quad at 3:00 PM

2024-02-29 • Next Thursday

- Password Cracking
- Learn how to crack different types of password hashes using tools like hashcat and John the Ripper!



ctf.sigpwny.com

`sigpwny{0verflow1ng_wit5_crypt0}`

Meeting content can be found at
sigpwny.com/meetings.

