SIGPwny

SP2025 Week 12 • 2024-04-17

Blockchain

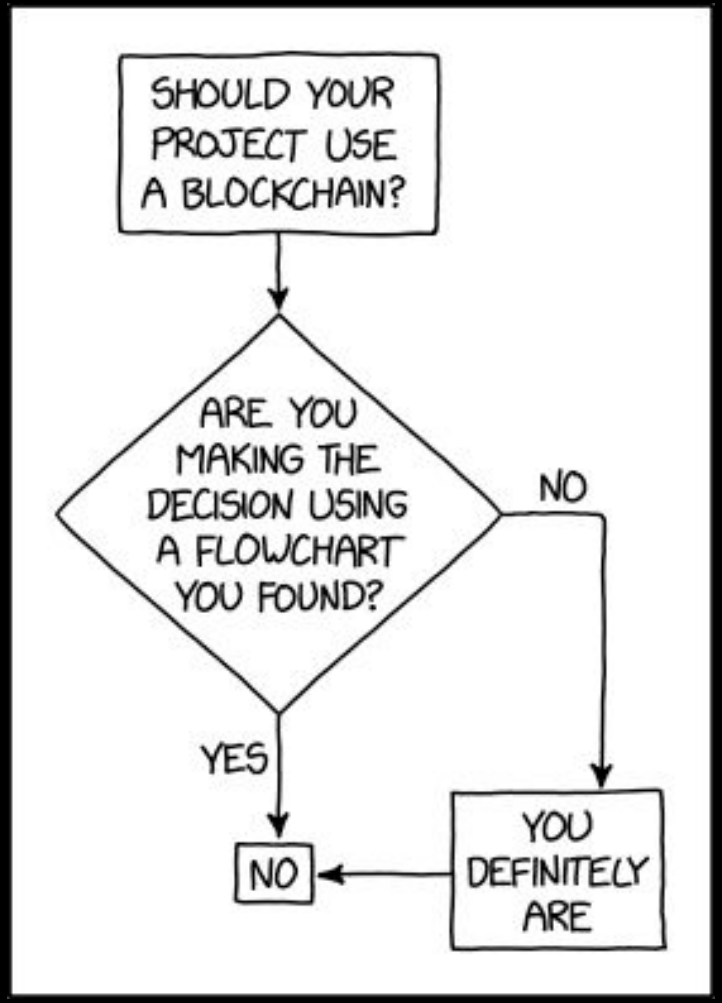
Jake Mayer and Michael Khalaf

Announcements

- **b01lers CTF 2025** tomorrow at **6 PM**!
  - Come to the Siebel CS building to play in our friend/rival team's CTF.
    - Room TBD, probably 2405
  - Free pizza will be provided!

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

- Digital signatures
- Distributed ledger
- Consensus mechanism
- Peer-to-peer network

Bitcoin

- First implementation of decentralized currency
- Allows parties to transact Bitcoin using digital signatures
- 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

Crypto Review: Digital Signatures

- Keypair: corresponding public and private key
  - Public key identifies a party
  - Private key is used to generate signatures
- A signature verifies that the owner of the private key authorizes a message

Crypto Review: 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

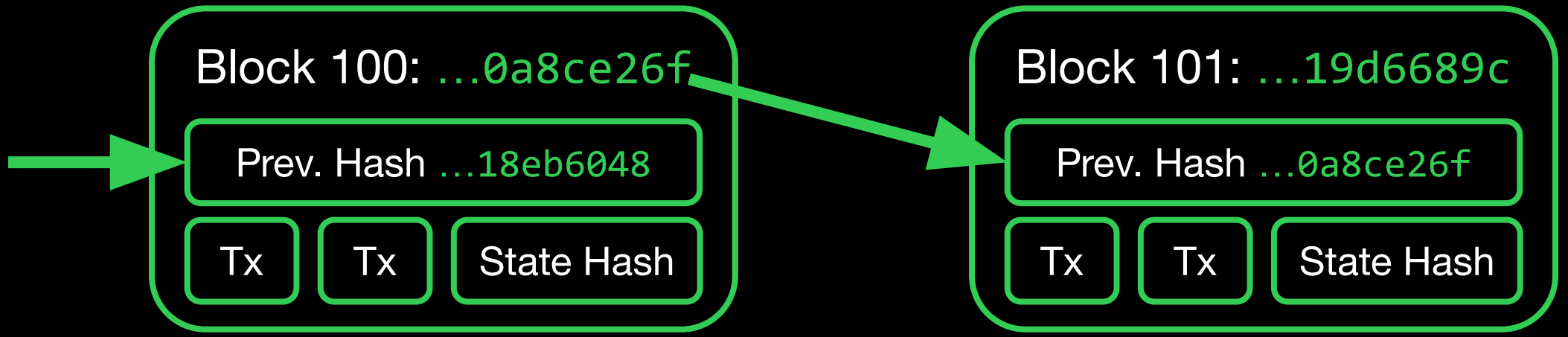
- The basic unit for interacting with a cryptocurrency system
- Signed by the party's private key
- Authorized for the corresponding public key
- Modifies the state of the blockchain

Distributed Ledger

- Records the state of the system (ie. account balances, contract storage, etc.)
- Ex. Bob has $1000, a governance proposal passed, etc.

Blockchain

- How the ledger is stored
- Transactions grouped into blocks
- Blocks are identified by their hash
- Blocks are chained by including the previous block's hash in the next hash calculation

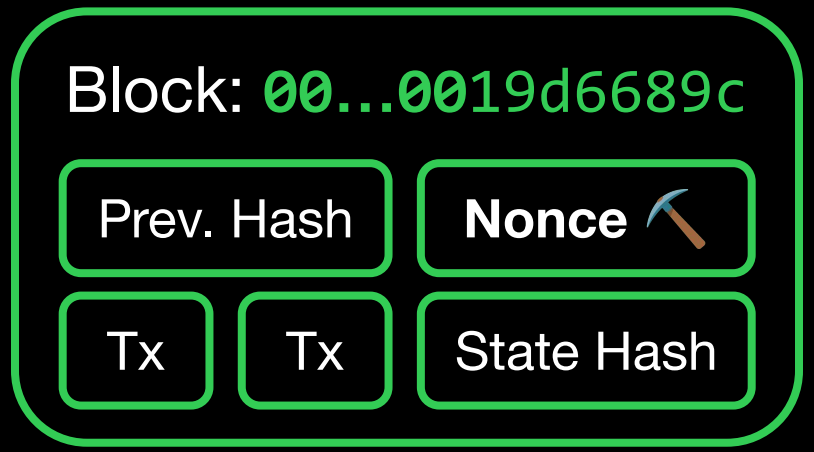


Consensus

- How do we determine the state of the blockchain?
- Need a mechanism for determining the current/head block
  - The rest of the state is a consequence
- Makes it difficult for a single entity to control the network

Proof-of-Work Consensus

- Consensus by demonstrating computational power (work)
- Blocks get their authority by demonstrating sufficient work
  - Miners search for a nonce to create a low enough hash
- Producing valid blocks requires high computational power
- Mining difficulty is adjusted dynamically for pacing



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?

All Together

How can Bob send Alice $10?

Bob 👨

Create & Sign transaction:

Send $10 to Alice 🔑

All Together

How can Bob send Alice $10?

Bob 👨

Create & Sign transaction:

Send $10 to Alice 🔑

Alice 👩

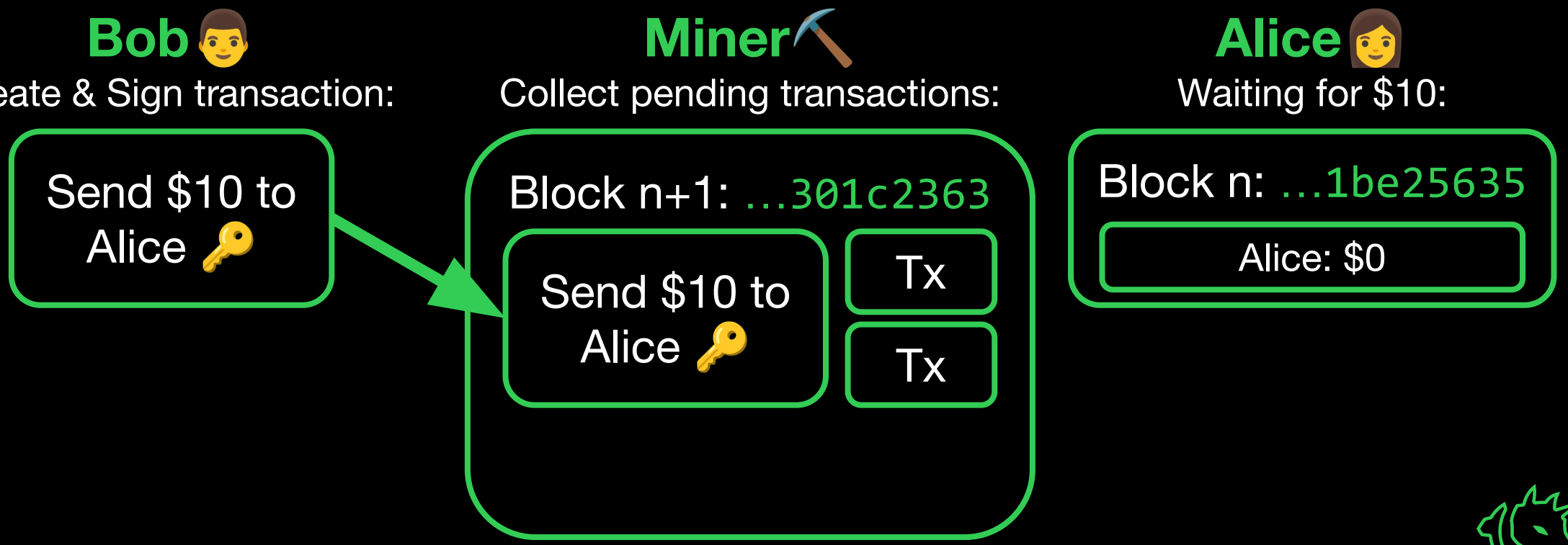
Waiting for $10:

Block n: ...1be25635

Alice: $0

All Together

How can Bob send Alice $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 👩

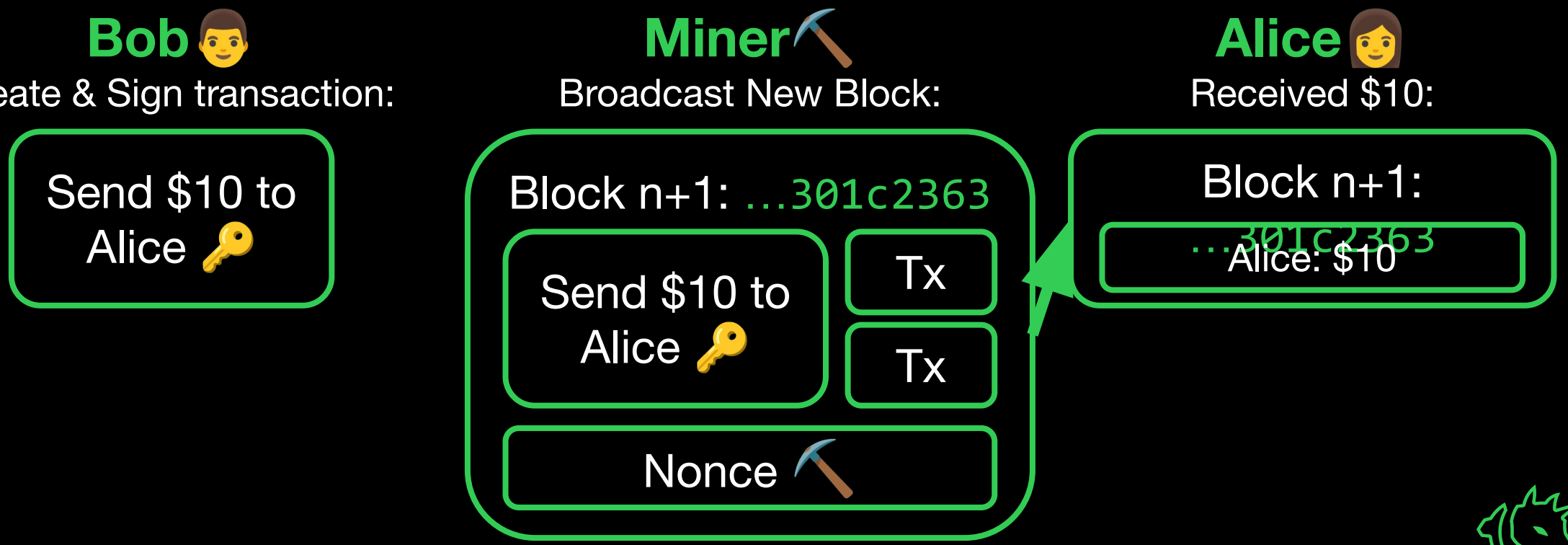
Waiting for $10:

Block n: `...1be25635`

Alice: $0

All Together

How can Bob send Alice $10?



51% Attack

- The attacker has majority of the network's hash power
- Reliably produce blocks faster than the rest of the network
- Attacker can arbitrarily manipulate the network and rewrite history
  - Double spending: Claw back money spent and spend elsewhere
  - Censor transactions
- Typically, regular mining incentives outweighs loss of value if currency becomes untrusted
  - Expensive to execute

51% Attack

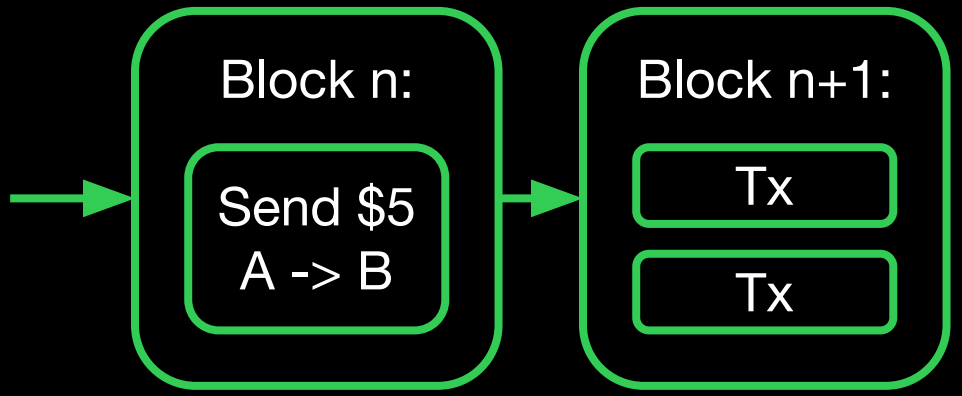
- Several historical attacks have occurred
- In one example, an attacker multiple-spent $17.5-18.6 million worth of BTG by targeting cryptocurrency exchanges
- ETC has suffered numerous attacks including $millions of double-spend due its decreasing popularity with miners
- Hash power can be rented through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

51% Attack

- Goal: A (attacker) wants to claw back money sent to B (victim) to use elsewhere
- Method: Use hash power advantage to create a new chain without the transaction

51% Attack

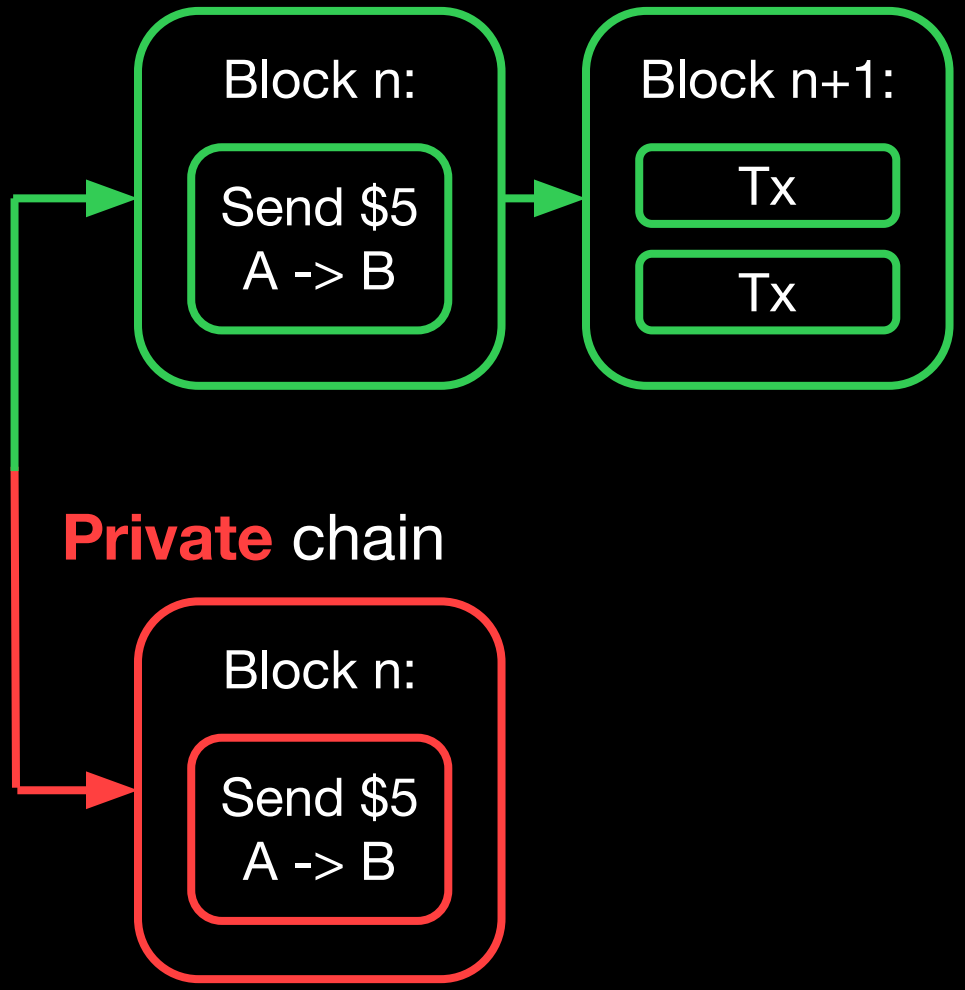
Public chain



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

51% Attack

Public chain

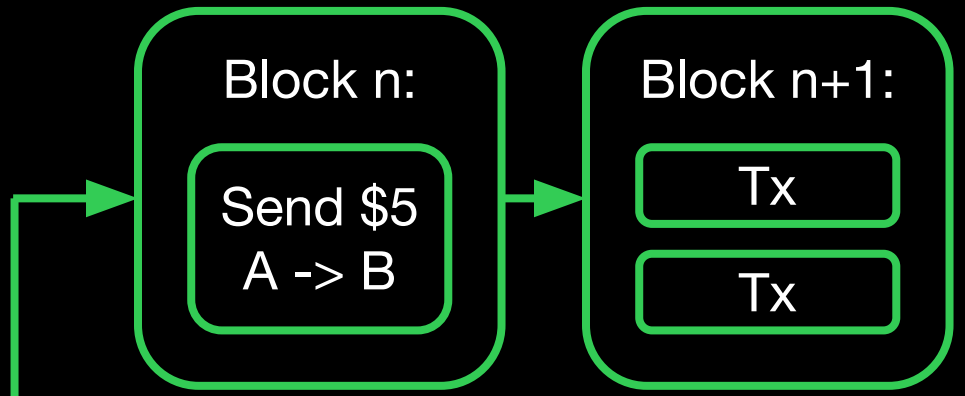


… Eventually, B sees that their $5 has been confirmed for several blocks

Plot twist! We've been mining in private…

51% Attack

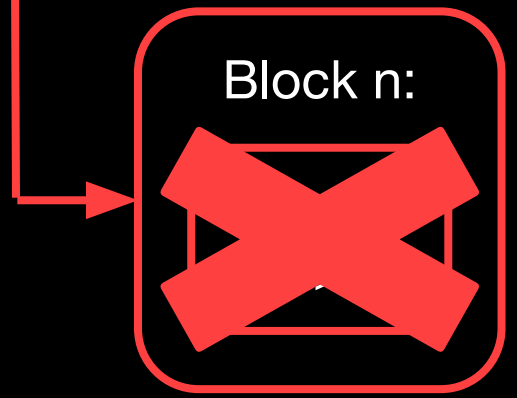
Public chain



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

Plot twist! We've been mining in private…

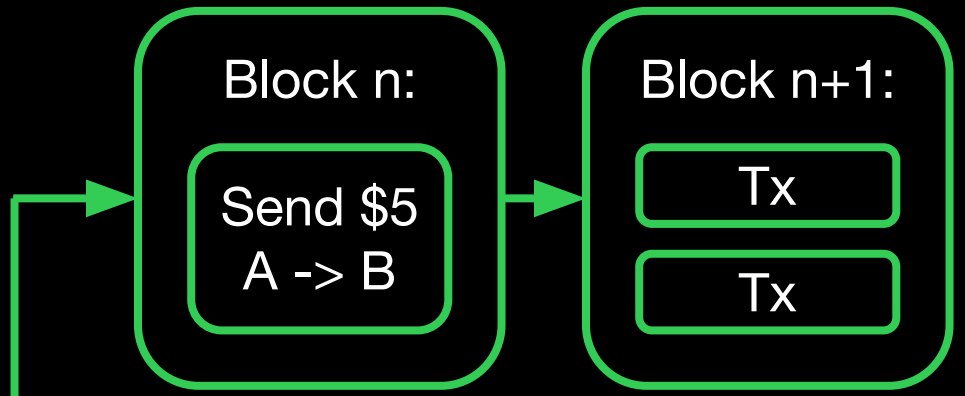
Private chain



Attacker excludes the transaction from the new block

51% Attack

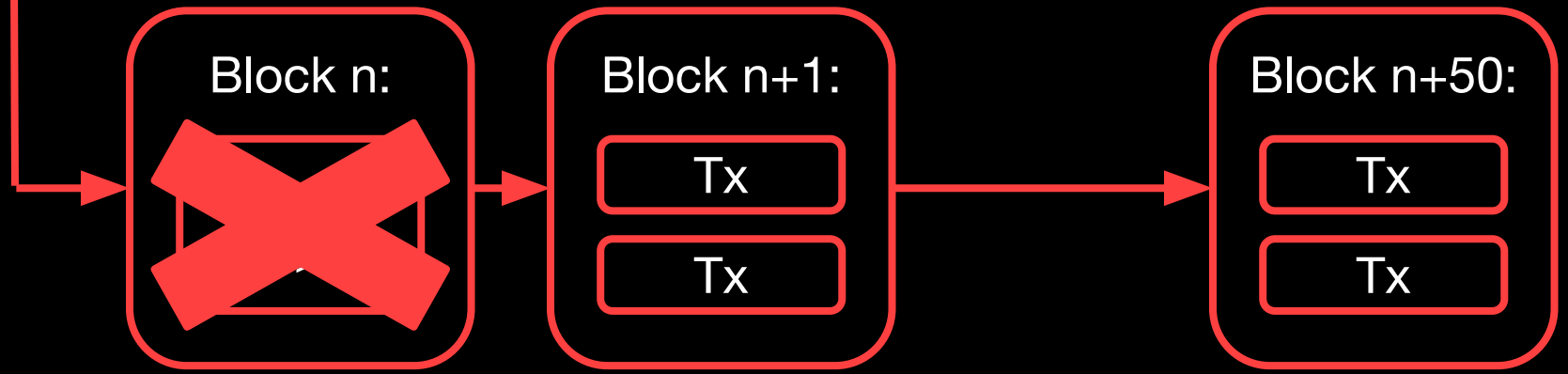
Public chain



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

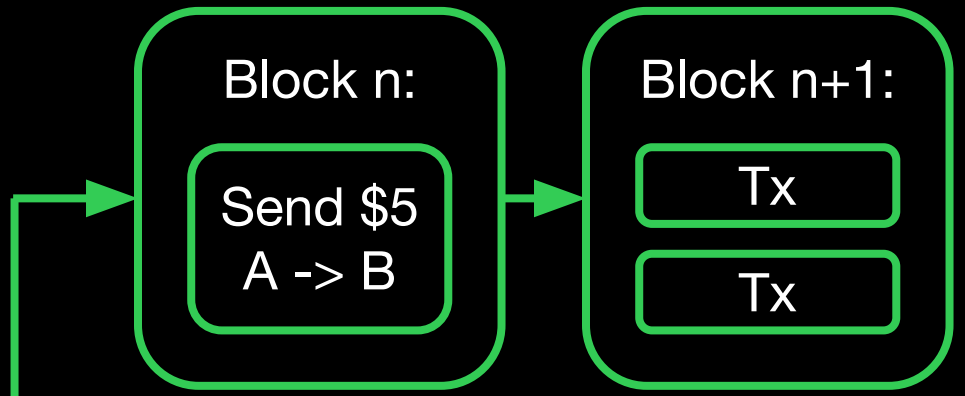
Plot twist! We've been mining in private…

Private chain



51% Attack

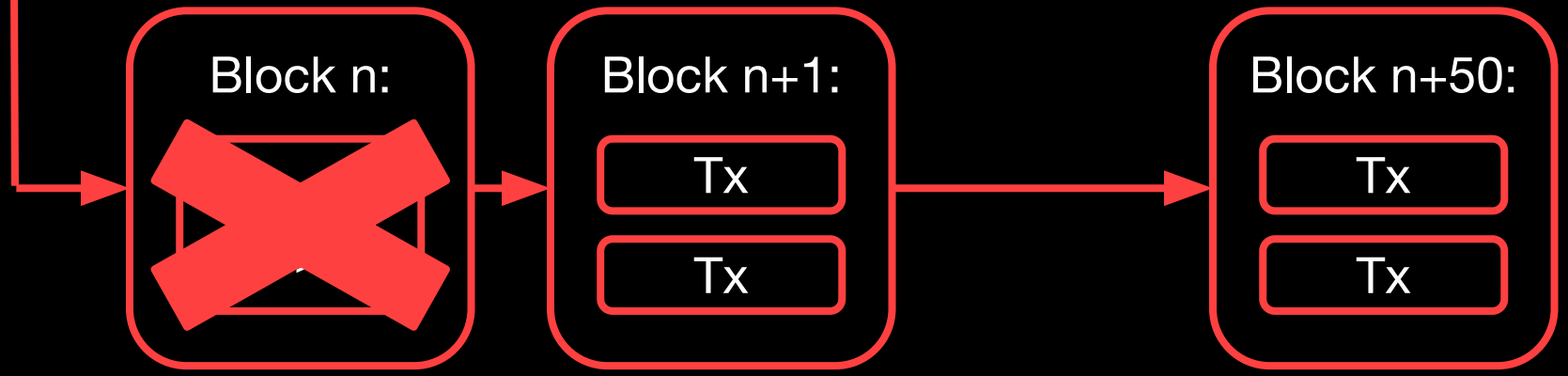
Public chain



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

Plot twist! We've been mining in private…

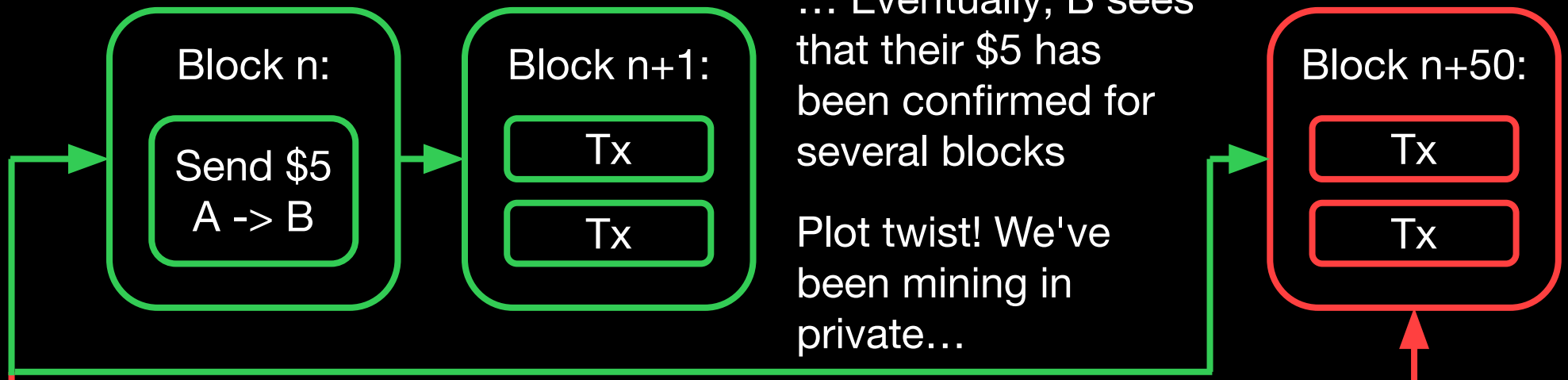
Private chain



Broadcast private blockchain

51% Attack

Public chain

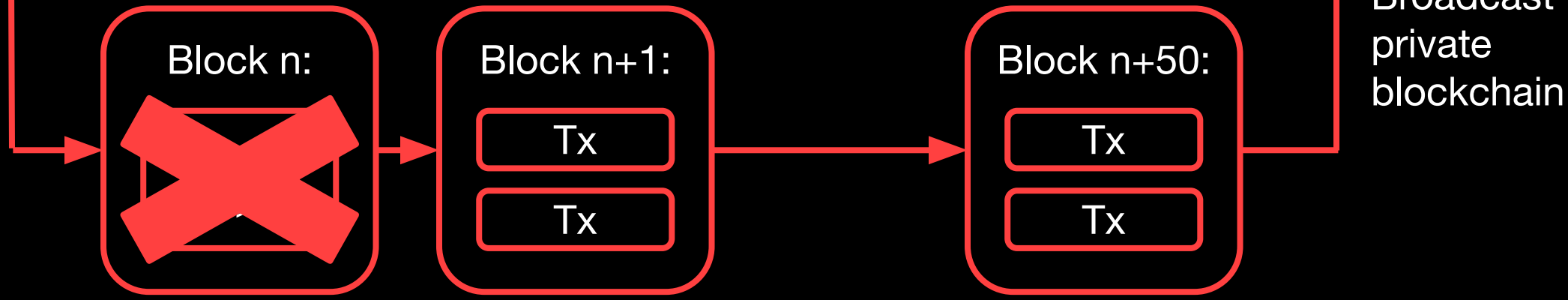


… Eventually, B sees that their $5 has been confirmed for several blocks

Plot twist! We've been mining in private…

Network accepts longer chain excluding the spend

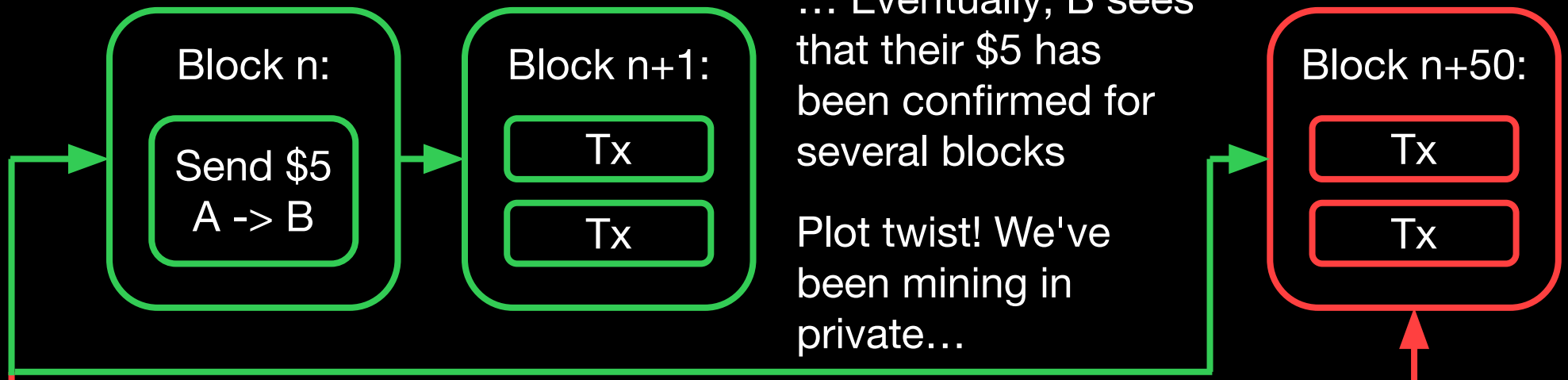
Private chain



Broadcast private blockchain

51% Attack

Public chain

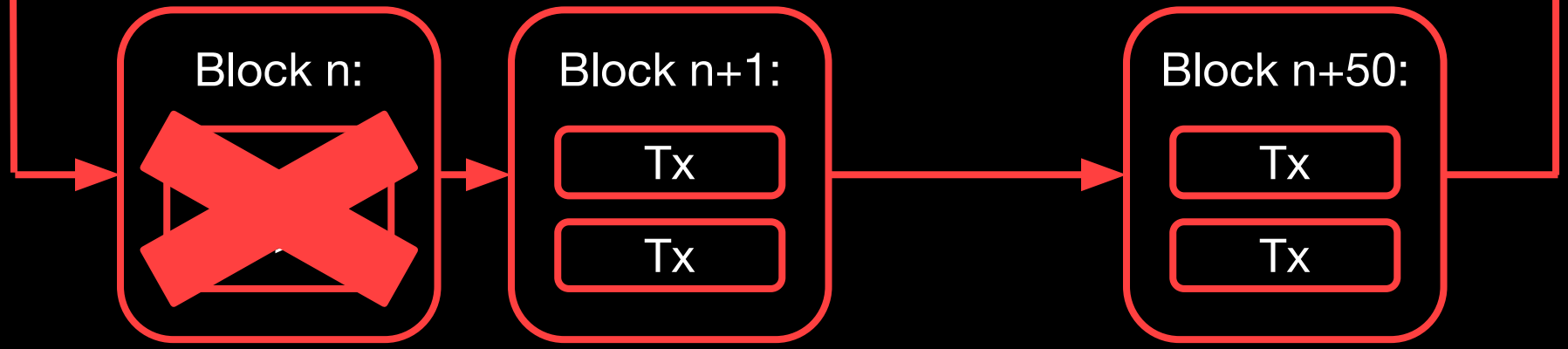


… Eventually, B sees that their $5 has been confirmed for several blocks

Plot twist! We've been mining in private…

Network accepts longer chain excluding the spend
Attacker has clawed back the money victim thought was theirs

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;
    }
}
```

Attacker Contract

Invoke `withdraw` on victim

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;
    }
}
```

Attacker Contract

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;
    }
}
```

Attacker Contract

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;
    }
}
```

Attacker Contract

bal=100

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;
    }
}
```

Attacker Contract

Victim transfers balance to attacker…

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;
    }
}
```

Attacker Contract

Victim transfers balance to attacker… along with execution

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;
    }
}
```

So these aren't executed yet

Attacker Contract

Victim transfers balance to attacker… along with execution

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;
    }
}
```

Attacker Contract

The attacker can invoke `withdraw` again

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;
    }
}
```

Attacker Contract

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;
    }
}
```

Attacker Contract

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;
    }
}
```

Attacker Contract

bal=100

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;
    }
}
```

Attacker Contract

Victim transfers balance to attacker again

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

2025-04-18 • **This Friday**

- b01lers CTF

2025-04-20 • **This Sunday**

- Passkeys

ctf.sigpwny.com

sigpwny{0verflow1ng_wit5_crypt0}

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

SIGPwny