Blockchain Databases

Dov Kruger

Department of Electrical and Computer Engineering
Rutgers University

March 20, 2024

Definition

- Blockchain is a shared, immutable ledger
- There are two kinds: public and private
- In a public blockchain
  - All transactions are viewable by everyone
  - Multiple copies of the ledger exist all over
  - All transactions are verifiable

Problem: If two ledgers are different who defines reality?

Key Technical Hurdle: Trust

- How to build a trustworthy database without trusting parties?
- Based on Asymmetric Cryptography

Misconceptions about Public Blockchains

- Good way to hide activity
- Good way to cheat on taxes
- More secure than a bank account

Risk in Banking

- Bank goes bankrupt
- Employee steals money from account
- Hacker steals money from account
- Bank ledger is hacked and records destroyed
- Government seizes assets
- Inflation of currency

Safeguards

- Rule of law
- Government guarantees: SBIC
- Insurance

Risk of Blockchain

- Hacker steals private key
- Send money to wrong public key (even nonexistent)
- Private key is lost
- Cryptocurrency drops in value
- Lack of liquidity in exchanges
- Arbitrary transaction costs
- Exchange lack of trustworthiness (like a bank?)
- Code is rewritten: whose law governs cryptocurrency?

Summary: Everything has risk. Cryptocurrencies are not currently safer than banks. The question is, do they have a future?

Trustless and Decentralized Databases

- Blockchain enables trustless and decentralized databases.
- Trustless means that participants do not need to trust each other or a central authority.
- Decentralized means that the control and ownership of data are distributed across a network.

Cryptography

- Symmetric Cryptography
  - Both parties use the same key
  - Both parties use the same algorithm
  - Cannot distinguish between parties
- Asymmetric Cryptography
  - Relies on one-way functions
  - May be insecure
  - if secure, many operations possible

Simple Symmetric Cryptography Example

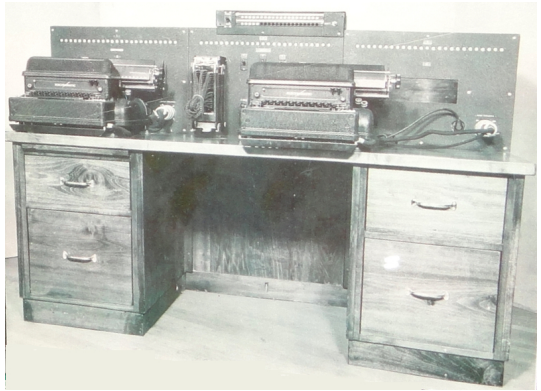
- $key = \text{"}A\text{"} = 01000001$
- $message = \text{"}hello\text{"} = 68656c6c6f$
- $encrypted = message \oplus key$
- $68656c6c6f \oplus 4141414141 = 29242d2d2e$
- Anyone with the key can do this
- XOR is its own inverse, to recover, do it again
- $29242d2d2e \oplus 4141414141 = 68656c6c6f$
- Same is possible with addition and subtraction instead

Plaintext Attack

- If the key can be determined if a message is known
- Major weakness in a cryptosystem
- Example: using the XOR method just used
- Given knowledge: Message = "hello"
- key = 'A'

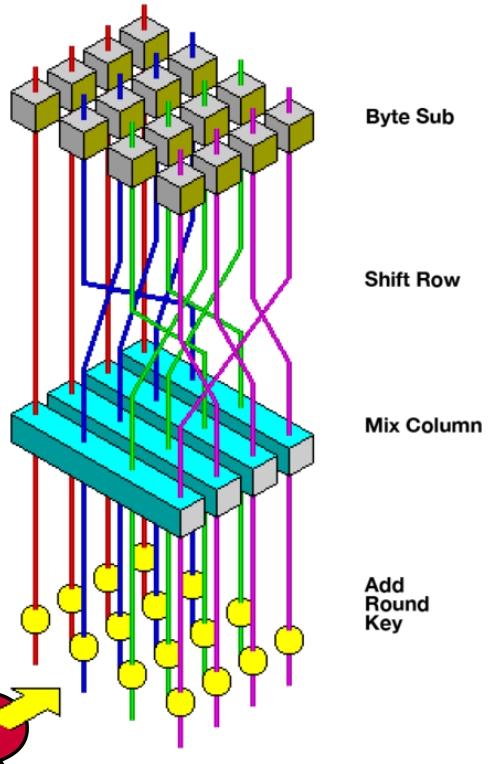
Plaintext Attack History: Enigma

- In World War II, both Germany and Japan relied on electromechanical machines to encrypt messages
- Germany: The Enigma machine
- Japan: Type B (US called it Purple)
- UK attacked Enigma messages by using
  - weather messages which were rigidly formatted
  - messages enciphered both in enigma and another breakable method



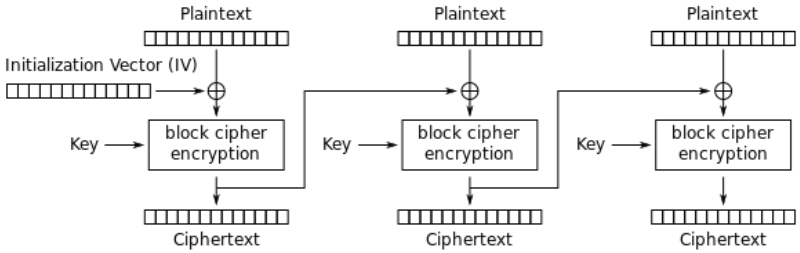
Current Symmetric Cryptography: AES-256

- 256-bit key
- 128-bit block
- 14 rounds of encryption
- substitution, transposition, mixing
- `https://www.cryptool.org/en/cto/aes-animation`



Increasing Difficulty: CBC

- Even with a complicated function, a key is not safe if it is reused a lot
- Methods of increasing difficulty: Cypher Block Chaining (CBC)
- Even CBC is subject to plaintext attack



Cipher Block Chaining (CBC) mode encryption

Asymmetric Cryptography

- Public and Private key
- Everyone knows the public keys, can be distributed
- Source of public keys must be trustworthy
- Generally slower than symmetric cryptography operations
- Relies on one-way functions
- $m' = E(m, k_{pub})$
- $m = D(m', k_{priv})$
- E and D are inverse operators
- $E(D(m, k_{pub}), k_{priv}) = m$
- $D(E(m, k_{pub}), k_{priv}) = m$

Digital Signature

- A digital signature proves that a message was sent by the owner of the private key
- Assuming
  - The private key is kept secret
  - The correct public key is distributed
  - The cryptosystem is not broken
  - New messages cannot be constructed with the same hash value
- Compute a hash of the message: $h = H(m)$
- Sending the hash and the message would not work
- Given m, h the attacker could just change the message and rehash it
- Instead, send $[m, D(h, k_{priv})]$
- only the author knows the private key, this proves the author sent it

Signature Validate

- Given $[m, h' = D(h, k_{priv})]$, encrypt the hash
- $h = E(h', k_{pub})$
- Compute the message hash: $h = H(m)$
- If the two match, the signature is valid

Proof of Identity

- Sender picks a random number (nonce)
- Sender encrypts using receiver's public key $E(n, k_{pub})$ and sends
- Receiver decrypts: $n = D(m, k_{priv})$ and send back to sender, proving identity
- in ssh key authentication, both sides do this. First we verify that the server really is who it says it is

Signature Validation Security

- Over time, the security of digital signature fades
- Computers get faster, and better algorithms start to threaten the hash function
- My idea: retain a "secret" hash on file in a secure repository
- If an attack in the future claims the message is not correct and offers an alternative
- Prove by unlocking the original secret and rehashing using the secret

Diffie-Hellman Key Exchange

- Given RSA is vulnerable to plaintext attacks, never send known text
- Pick a random number (nonce)
- Party A computes $E(g^x \mod p, k_{pub,B})$ and sends to B
- Party B computes $E(g^y \mod p, k_{pub,A})$ and sends to A
- Both parties compute a shared key
- Use a symmetric algorithm (AES-256) to encrypt messages
- Change keys regularly (ie every 30 minutes)

Secure Hash Algorithm

A cryptographic hash algorithm must:

- Any change in input should cause a large (most or all bit) change in hash
- Difficult to reverse engineer the message
- Difficult to construct a different message with the same value.
    - Example
    - hash("X sells house to Y for 1000BTC")
    - hash("X gives house to Y for $1")

Given m is large, there must be multiple m for the same $hash(m)$

Certificates and Authorities

- Asymmetric cryptography is convenient because it simplifies establishing secret communication, but..
- It does require getting the right public key
- Problem: Who do you trust?
- Solution: A Certificate Authority
- One place that stores all public keys and gives them out

RSA

- RSA is an asymmetric cryptosystem using prime factorization as the one-way function
- Relies on the fact that
  - It is easy to multiply two numbers, even big ones
  - Very hard to factor a large number into two primes
  - The problem is, just because we don't know how to do it does not mean it can't be done

RSA Details

- Pick two large primes $p = 17, q = 41$
- Compute the product $n = pq$
- Compute $\phi(n) = (p-1)(q-1) = 16 * 40 = 640$
- Pick an integer $1 < e < \phi(n)$ and $e, \phi(n)$ are coprime
- $e = 11$
- Compute multiplicative inverse $de \mod \phi(n) = 1$
- $d = 7$
- public key $= (n, e)$
- private key $= (n, d)$
- Security of RSA depends on: factoring n is slow

Extended Euclid

```
extended_gcd(a, b)
    oldr ← a
    r ← b
    olds ← 1
    s ← 0
    oldt ← 0
    t ← 1
    while r ≠ 0 do
        quotient ← oldr ÷ r
        oldr ← r
        r ← oldr − quotient * r
        olds ← oldt
        oldt ← t
        t ← olds − quotient * t
    end
    return olds
```

Extended Euclid

Example: extendedGCD(240,46)

step	quotient	remainder	s_i	t_i
0		240	1	0
1		46	0	1
2	$240/46 = 5$	$240 - 5 * 46 = 10$	$1 - 5 * 0 = 1$	$0 - 5 * 1 = -5$
3	$46/10 = 4$	$46 - 4 * 10 = 6$	$0 - 4 * 1 = -4$	$1 - 4 * (-5) = 21$
4	$10/6 = 1$	$10 - 1 * 6 = 4$	$1 - 1 * -4 = 5$	$-5 - 1 * 21 = -26$
5	$6/4 = 1$	$6 - 1 * 4 = 2$	$-4 - 1 * 5 = -9$	$21 - 1 * -26 = 47$
6	$4/2 = 2$	$4 - 2 * 2 = 0$	$5 - 2 * -9 = 23$	$-26 - 2 * 47 = -120$

$gcd = as_i + bt_i = 240 * -9 + 46 * 47 = 2$

Asymmetric Operations in RSA

- Encryption $c = E[m] = m^e \mod n$
- Decryption $m = D[c] = c^d \mod n$
- Secure Hash Algorithms $hash(m) = H[m]$
  `https://sha256algorithm.com/`
- Signing $m, s = D[hash(m)]$
- Verifying $hash(m) = E[s]$

Risks to RSA

- Factoring $n = pq$ may not be slow
- One-way functions may not exist in general
  `https://en.wikipedia.org/wiki/One-way_function`
- Quantum computers of sufficient size could break RSA in polynomial time
- RSA can be broken with a plaintext attack

Elliptic Curve Cryptography: ECC

- Elliptic Curves
- key generation: random 256-bit integer
- public key: pairs of integer coordinates on the curve (x,y)
- compressed public key: x coordinate + 1 bit (odd or even)
- Requires fewer bits than RSA
- Not all curves are secure! Use curves identified by NIST
- ECC is also not quantum secure!

ECC Algorithm Overview

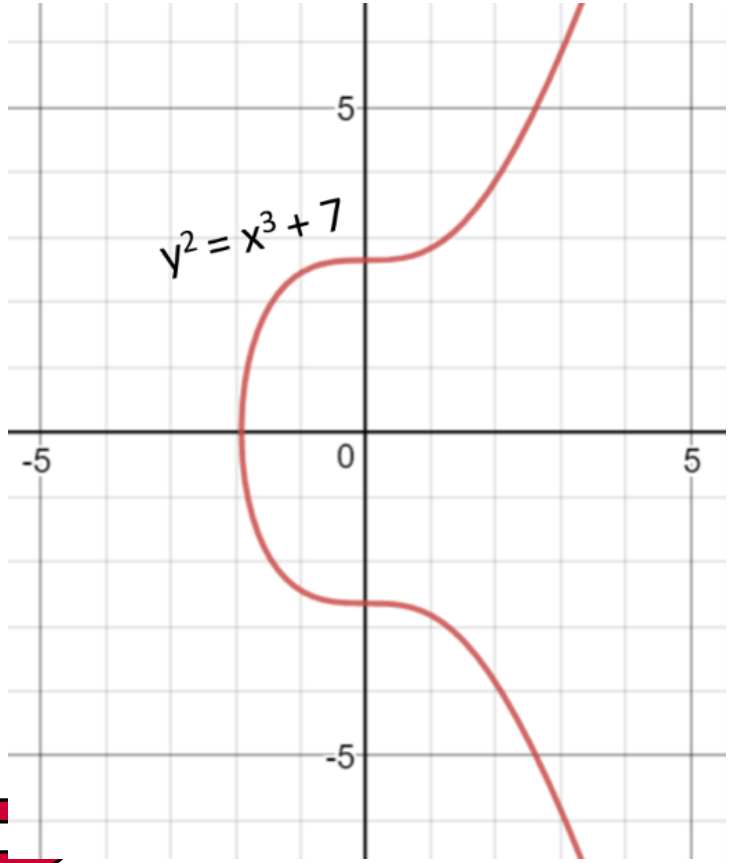
- ECC on an integer field means adding points is easy
- $y^2 = x^3 + 7 \mod 2^{256}$
- $P_2 = P_a + P_b$
- Doubling means calculating powers of P are easy
  $2P_1 = P_1 + P_1$
- Any multiple can be achieved by adding the powers:
  $19P_a = 16P_a + 2P_a + 1P_a$
- The reverse is hard P_1/k

ECC Visualized

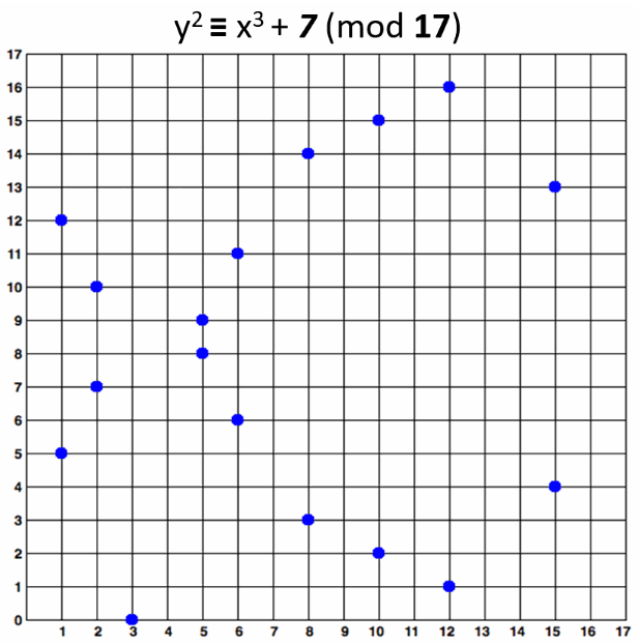
- NIST secp256k1 (used in bitcoin) over $GF(2^{256})$
- $y^2 = x^3 + 7 \mod 2^{256}$
- $p = 2^{256} - 2^{32} - 977$
- $G =$
  $(0x6B17D1F2E12C4247F8BCE6E563A440F277037D812DEB3$
  $0x4FE342E2FE1A7F9B8EE7EB4A7C0F9E162BCE33576B31$

ECC Discrete Logarithm Visualization

- https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc
- https://www.desmos.com/calculator/ialhd71we3



ECC Discrete Logarithm Points



ECC Animation

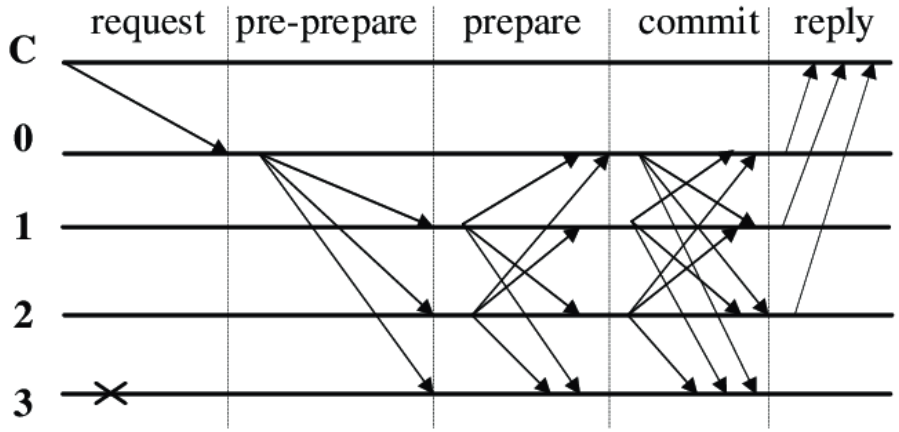
https://curves.xargs.org/

Byzantine Fault Tolerance (BFT)

- Byzantine generals problem
- How to come to consensus when some of the parties may be traitors
- Majority Rule: Blockchain ledger requires majority of nodes to agree
- If an attacker can gain $> 51\%$ of nodes they can change reality

Practical Byzantine Fault Tolerance (PBFT)

- Castro and Liskov 1990
- $2/3$ majority rule



Consensus Mechanisms in Blockchain

A consensus mechanism in blockchain is a way for distributed actors to agree on a ledger

- Mechanism must be able to survive an attacker trying to inject bad data
- Multiple parties must agree and synchronize their efforts
- Current Ideas include
  - Proof of Work (Nodes compete to compute something)
  - Proof of Stake (Nodes get chances to register transactions based on their holdings)

Proof of Work (PoW) Algorithm Explained

- People are rewarded for recording transactions in the chain
- Profit is amortized cost of hardware and electricity to run
- All nodes compete to compute a hard computational problem
- First to complete publishes the result and links a new block onto the chain
- Parties willing to validate a transaction are paid (mining)
- Over time, computation is expected to get faster, so reward drops over time
- PoW is a huge waste of power
- The problem is that without PoW, difficult to stop one party from taking control of the chain

Pseudocode: Proof of Work in Bitcoin

Repeatedly hash a value based on the block until we reach a value with the right number of leading zeros

```
block hash = sha256(TX_root, timestamp, previous_ha
nonce = 0
repeat
   nonce++
until sha256(TX_root, timestamp, previous_hash, non
```

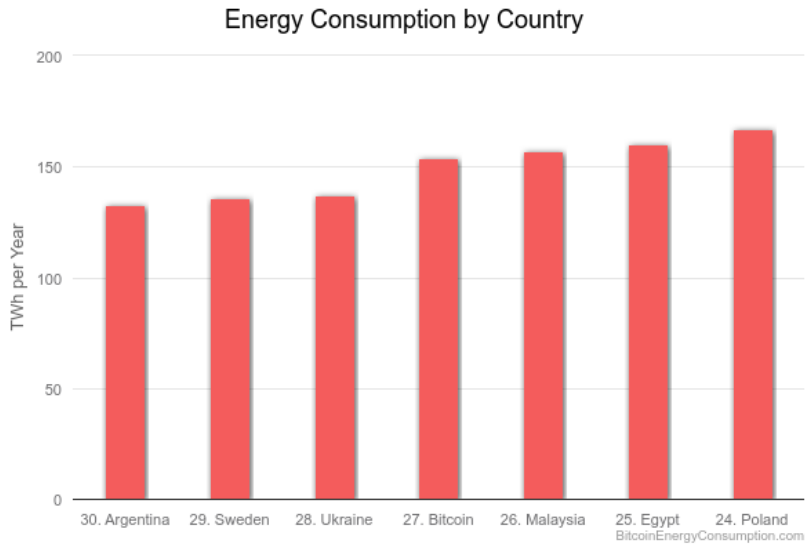
Electricity use: Bitcoin

- Bitcoin started using CPUs in 2009
- For greater efficiency, ASIC (custom) mining hardware 2013
- Siberian shacks can be heated free because the cost of electricity is lower than the earned bitcoin

Electricity use: Bitcoin

This graph shows bitcoin electricity usage over time

- 2017: Bitcoin uses more electricity than Belgium
- 2021: Bitcoin uses more electricity than Argentina
- 2024: Bitcoin uses more electricity than Ukraine



Proof of Stake (PoS) Algorithm Explained

- Nodes register transactions based on their stake
- Problem that big stakeholders can hijack the chain
- Fairness: the rich get richer

Replacements for PoW and PoS

- Somehow make everyone register, and give everyone equal access
- Base on posting a bond or cell phone identity, ie require money and physical presence
- Problem to prevent people from registering multiple devices to try to take over

Delegated Proof of Stake (DPoS) Consensus

- Overview of DPoS consensus mechanism
- Role of elected delegates and voting
- Scalability benefits and potential issues

Federated Byzantine Agreement (FBA) Protocol

- Explanation of FBA protocol
- Usage in permissioned blockchain networks
- Comparison with other consensus algorithms

Sharding in Blockchain Networks

- Definition and purpose of sharding
- How sharding improves blockchain scalability
- Challenges and potential solutions

Scalability Challenges and Solutions in Blockchain

- Overview of scalability issues in blockchain
- Layer 1 and Layer 2 scaling solutions
- Examples of implemented solutions

Layer 2 Solutions: Lightning Network and Plasma

- Introduction to Lightning Network and Plasma
- Off-chain payment channels and sidechains
- Benefits and limitations of Layer 2 solutions

Smart Contracts: Introduction and Use Cases

- Definition and characteristics of smart contracts
- Examples of smart contract applications
- Potential impact on various industries

Security Risks

- **51% Attack:** If a single entity or a coalition of entities controls more than 50% of the network's computational power in a proof-of-work blockchain, they can manipulate transaction history, double-spend coins, and disrupt the network's operations.
- **Data Modification:** Once data is recorded on the blockchain, it is immutable. However, if incorrect data is recorded, it cannot be easily corrected or removed, leading to data integrity issues.
- If the private wallet id is stolen, it can be emptied
- If the encryption system becomes breakable, then nothing in the blockchain is safe

Security Risks (Continued)

- **Smart Contract Vulnerabilities:** Smart contracts are susceptible to bugs and vulnerabilities in the code, which can lead to exploits and financial losses.
- **Sybil Attacks:** In a Sybil attack, a malicious actor creates multiple fake identities or nodes to gain control or influence over the network.
- **Regulatory and Compliance Risks:** Blockchain projects may face regulatory challenges and legal uncertainties, especially regarding compliance with existing laws and regulations related to securities, privacy, and financial transactions.
- **Privacy Risks:** While blockchain offers pseudonymity, transactions and data stored on the blockchain can still be traced back to individuals or entities, posing privacy risks.
- **Scalability and Performance:** Blockchain networks face scalability and performance challenges, especially as the number of transactions and users increase. Slow transaction speeds and high fees can impact user experience and adoption.

Privacy and Anonymity in Blockchain Transactions

- Challenges of privacy in public blockchains
- Techniques for enhancing privacy such as ring signatures and zero-knowledge proofs
- Privacy-focused blockchain projects
  -

Challenges to Privacy in Public Blockchains (Part 1)

1. **Transparent Transactions**:
   - All data is visible, exposing transaction details such as sender and recipient addresses, transaction amounts, and timestamps.
   - Lack of privacy can compromise user anonymity and expose sensitive financial information to anyone with access to the blockchain.
2. **Address Reuse**:
   - Reusing addresses in public blockchains can lead to privacy leaks as adversaries can link multiple transactions to the same user.
   - Blockchain transactions are permanent and immutable, allowing adversaries to trace spending habits and income sources.
3. **Blockchain Analysis Tools**:
   - Enable third parties to trace and analyze blockchain transactions
   - Uncovering patterns and relationships between addresses.
   - Used by adversaries, regulators, and law enforcement agencies to deanonymize users and track illicit activities.

Challenges to Privacy in Public Blockchains (Part 2)

4. **Network Analysis**:
   - IP address correlation
   - Transaction propagation analysis
   - Can deanonymize users by linking their blockchain transactions to their internet activities.
5. **Data Leakage from Smart Contracts**:
   - Smart contracts may expose sensitive data or metadata, compromising user privacy.
   - Improper data handling
   - Unintended information disclosure can lead to data leakage and privacy breaches.
6. **Regulatory Compliance**:
   - Know Your Customer (KYC) and Anti-Money Laundering (AML) requireents
   - May conflict with user privacy expectations on public blockchains.
   - Compliance with these regulations often involves identity verification and transaction monitoring, undermining user anonymity and privacy.

Privacy-Focused Blockchain Projects (Part 1)

1. **Monero (XMR)**:
   - Privacy-focused cryptocurrency that aims to provide secure, private, and untraceable transactions.
   - It utilizes features such as
     - Ring signatures
     - stealth addresses
     - confidential transactions
2. **Zcash (ZEC)**:
   - Zcash is a privacy-centric cryptocurrency that employs zero-knowledge proofs (zk-SNARKs) to enable shielded transactions.
   - Transaction details are encrypted and only accessible to the parties involved, providing enhanced privacy while still allowing for selective disclosure when necessary.

Privacy-Focused Blockchain Projects (Part 2)

3. **Dash (formerly Darkcoin)**:
   - Dash is a privacy-focused cryptocurrency that offers optional privacy features through its PrivateSend functionality.
   - It utilizes a decentralized mixing mechanism to obfuscate the origin and destination of transactions, enhancing privacy for users.
4. **Grin (GRIN)**:
   - Grin is an open-source privacy-focused cryptocurrency that emphasizes privacy, scalability, and fungibility.
   - It implements the Mimblewimble protocol, which achieves privacy by default through features such as confidential transactions and CoinJoin.
5. **Beam (BEAM)**:
   - Beam is a privacy-focused cryptocurrency based on the Mimblewimble protocol, similar to Grin.
   - It offers features such as confidential transactions, opt-in auditability, and atomic swaps to provide privacy, scalability, and interoperability.

Zero-Knowledge Proofs

Zero-knowledge proofs (ZKPs) allow one party, the prover, to convince another party, the verifier, that a statement is true without revealing any additional information beyond the validity of the statement itself. This process involves three main steps:

1. **Setup**: Agreement on parameters and cryptographic primitives.
2. **Proof Generation**: Prover constructs a proof without revealing the witness.
3. **Proof Verification**: Verifier verifies the proof's validity.

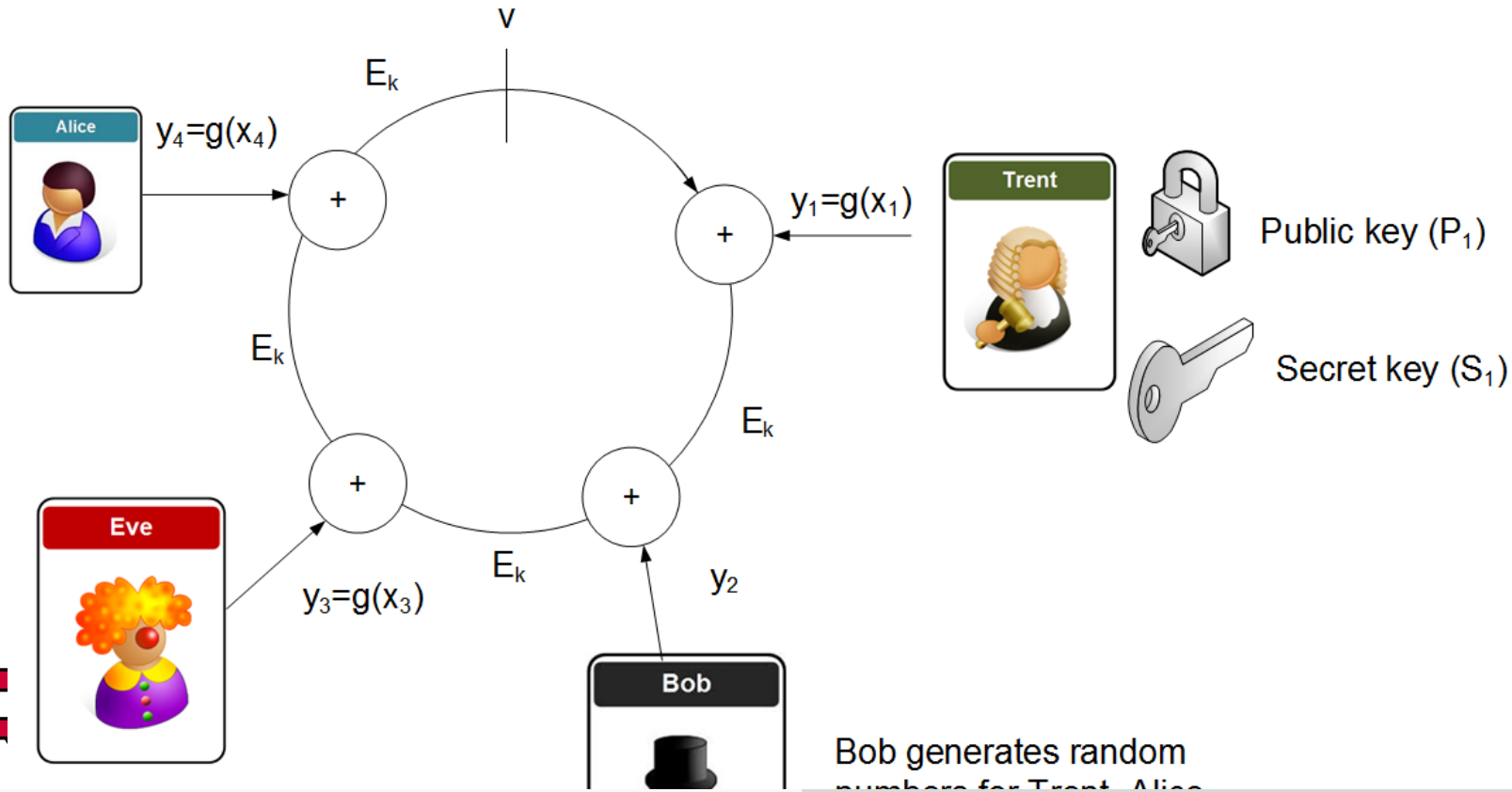
Key properties of ZKPs:

- **Completeness**: Convincing the verifier of a true statement.
- **Soundness**: Ensuring false statements cannot be proven.
- **Zero-knowledge**: Revealing no additional information beyond statement validity.

ZKPs find applications in privacy-preserving transactions, authentication protocols, identity management, and secure multiparty computation.

Ring Signatures and Confidential Transactions

- Ring signatures are a cryptographic protocol that enables users to sign transactions without revealing their private keys.
- User combines their public key with multiple other users making it difficult to tell which user signed.
- Ring signatures are used to provide additional security for transactions and prevent tampering.



Ring Signature Privacy

- **Anonymity**: Ring signatures obscure the true identity of the signer by including a group of potential signers' public keys in the signature.
- **Increased Privacy**: With a larger anonymity set, it becomes difficult for adversaries to determine the true signer of a message, enhancing user privacy.
- **Plausible Deniability**: Signers benefit from plausible deniability as any member of the ring could have produced the signature, making it challenging to attribute the signature to a specific individual.
- **Privacy-Preserving Transactions**: Ring signatures are commonly used in privacy-focused cryptocurrencies like Monero to facilitate confidential and untraceable transactions, protecting user financial privacy.

Ring Signature Security

- **Resistance to Compromise**: Ring signatures distribute the signing authority across a group of potential signers, reducing the risk of compromise associated with individual signatures.
- **Forgery Resistance**: Ring signatures are computationally binding, making it infeasible for adversaries to produce a valid signature without access to the legitimate signer's private key or collusion with other ring members.
- **Anonymity Sets**: Larger anonymity sets provided by ring signatures make it more challenging for adversaries to conduct targeted attacks or surveillance, enhancing the security of the signer.
- **Enhanced Security Measures**: Besides privacy benefits, ring signatures also offer additional security advantages, making them a valuable tool for ensuring the security and integrity of digital transactions and communications.

Blockchain Governance Models

The mechanisms and processes by which decisions are made

- Development
- Maintenance
- Evolution

Several types of blockchain governance models

- **Decentralized Governance**: In a decentralized governance model, decision-making power is distributed among network participants, typically through a consensus mechanism. Examples include proof of work (PoW) and proof of stake (PoS) protocols, where stakeholders have a say in network decisions based on their stake or computational power.
- **On-chain Governance**: On-chain governance involves decision-making processes that occur directly on the blockchain through smart contracts or protocol-level voting mechanisms. Participants can vote on proposals to enact changes to the protocol, such as software upgrades or parameter adjustments.
- **Off-chain Governance**: Off-chain governance refers to decision-making processes that occur outside the blockchain network, typically through traditional governance structures such as boards of directors, committees, or foundations. These entities may oversee the development and maintenance of the blockchain protocol, allocate resources, and make strategic decisions.

Decentralized Governance Example: Bitcoin

- In the Bitcoin network, decision-making power is decentralized among miners who contribute computational power to secure the network.
- Changes to the protocol require broad consensus among miners, developers, and users.
- For example, proposed changes to the Bitcoin protocol undergo extensive community discussion and are implemented through software updates known as Bitcoin Improvement Proposals (BIPs).

On-chain Governance Example: Tezos

- Tezos employs an on-chain governance mechanism where stakeholders can vote on protocol upgrades and amendments through a formalized voting process.
- Participants can submit proposals, vote on them, and delegate their voting power to others.
- Once a proposal is approved through the voting process, it is automatically implemented into the protocol without the need for manual intervention.

Off-chain Governance Example: Ethereum

- Ethereum's governance model combines on-chain decision-making with off-chain discussions and coordination.
- While major protocol changes are proposed and implemented through on-chain mechanisms (e.g., Ethereum Improvement Proposals), community feedback and discussions often occur off-chain through forums, social media, and developer meetings.
- Off-chain governance allows for broader community participation and informal discussions but may lack the transparency and enforceability of on-chain processes.

Hybrid Governance Example: Cardano

- Cardano utilizes a hybrid governance model that combines on-chain voting with off-chain community engagement.
- Protocol changes and funding proposals are voted on through on-chain mechanisms, ensuring transparency and immutability.
- However, community discussions, research, and development often occur off-chain, allowing for flexibility and collaboration outside the strict confines of the blockchain.

Regulatory Challenges and Compliance in Blockchain

- Overview of regulatory landscape for blockchain and cryptocurrencies
- Compliance requirements for blockchain-based businesses
- Regulatory trends and challenges

Energy Consumption and Environmental Impact of Blockchain

- Analysis of energy consumption in proof-of-work blockchains
- Environmental concerns and criticisms of blockchain technology
- Efforts to address sustainability issues

Quantum Computing Threats to Blockchain Security

- Potential impact of quantum computing on blockchain security
- Vulnerabilities of current cryptographic algorithms to quantum attacks
- Research and development efforts for quantum-resistant cryptography

Immutable Ledger: Data Integrity and Auditing

- Importance of immutability in blockchain
- Use cases of blockchain for data integrity and auditing
- How blockchain ensures tamper-proof records

Use Cases of Blockchain Beyond Cryptocurrencies

- Diverse applications of blockchain technology in various industries
- Examples of supply chain management, healthcare, voting systems, and more
- Potential benefits and challenges of blockchain adoption

Future Trends and Research Directions in Blockchain Technology

- Emerging trends in blockchain technology
- Areas of ongoing research and development
- Predictions for the future of blockchain and its impact on society

Blockchain Primitives

- Cryptographic Hash Functions
- Secure Hash Algorithms (SHA)
- Digital Signatures in Blockchain
- Elliptic Curve Cryptography (ECC)
- Merkle Trees and Merkle Proof
- Public Key Infrastructure (PKI)
- Consensus Algorithms in Blockchain
- Byzantine Fault Tolerance (BFT)
- Smart Contracts and Solidity
- Decentralized Applications (DApps)

Cryptocurrencies

- Cryptocurrencies are digital assets that use cryptography to secure transactions and verify transactions.
- Geeky appeal that known algorithms prevent inflation
- This ignores basic problems
  - Many risks, and no Safeguards
  - Each cryptosystem may have rules, but the community writing the code can change those rules
  - Intrinsic value is zero
  - Governments may outlaw use of cryptocurrencies
  - While inflation control may be designed into a currency, there is no limit to how many currencies can be created
- Bitcoin: `https://github.com/bitcoin/bitcoin`
- Ethereum: `https://github.com/ethereum`

Cryptographic Hash Functions

- Cryptographic hash functions are algorithms that take an input (or message) and produce a fixed-size string of bytes.
- Properties of cryptographic hash functions include determinism, pre-image resistance, second pre-image resistance, and collision resistance.
- In blockchain, hash functions are used to create digital fingerprints of data, ensuring data integrity and immutability.

Secure Hash Algorithms (SHA)

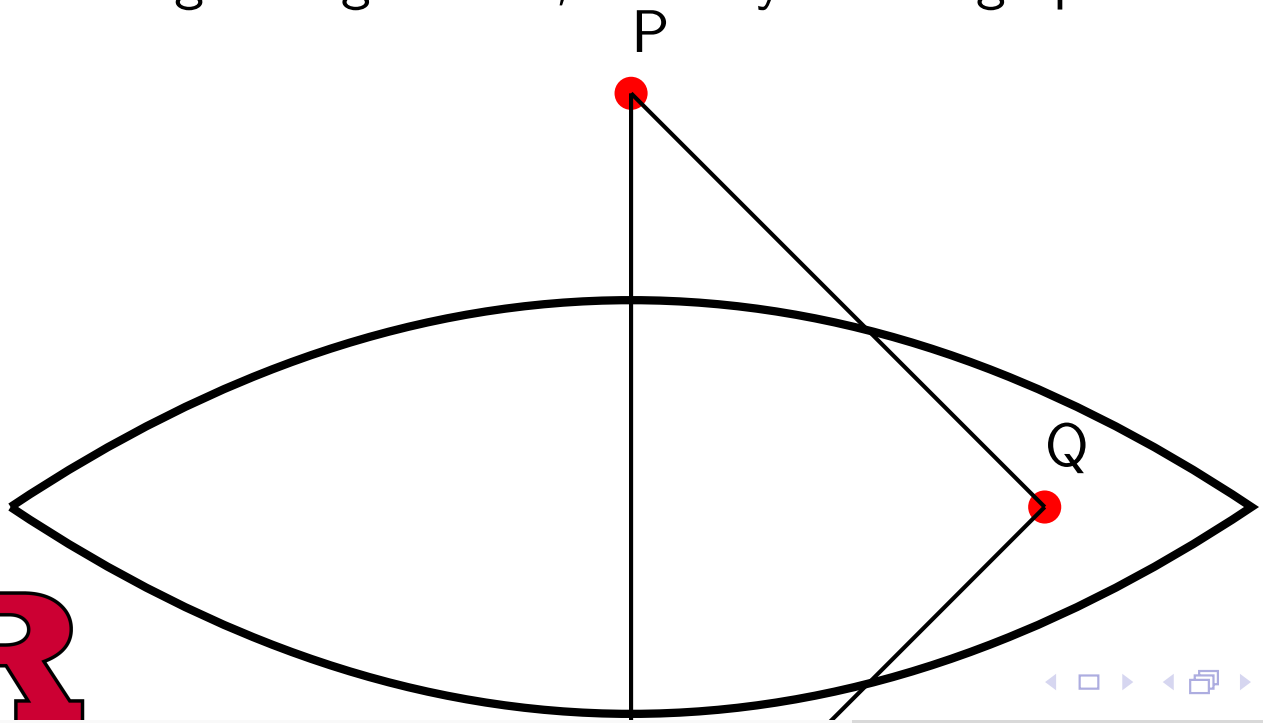
- The Secure Hash Algorithms (SHA) are a family of cryptographic hash functions designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST).
- SHA-256 is commonly used in blockchain to create unique hash values for blocks and transactions.
- SHA algorithms provide strong collision resistance and are widely adopted for securing data in blockchain networks.

Digital Signatures in Blockchain

- Digital signatures are cryptographic mechanisms that provide authentication, integrity, and non-repudiation for digital messages or documents.
- In blockchain, digital signatures are used to verify the authenticity of transactions and ensure that they have been authorized by the rightful owner of the associated private key.
- Digital signatures rely on asymmetric cryptography, where a private key is used to sign the message and a corresponding public key is used to verify the signature.

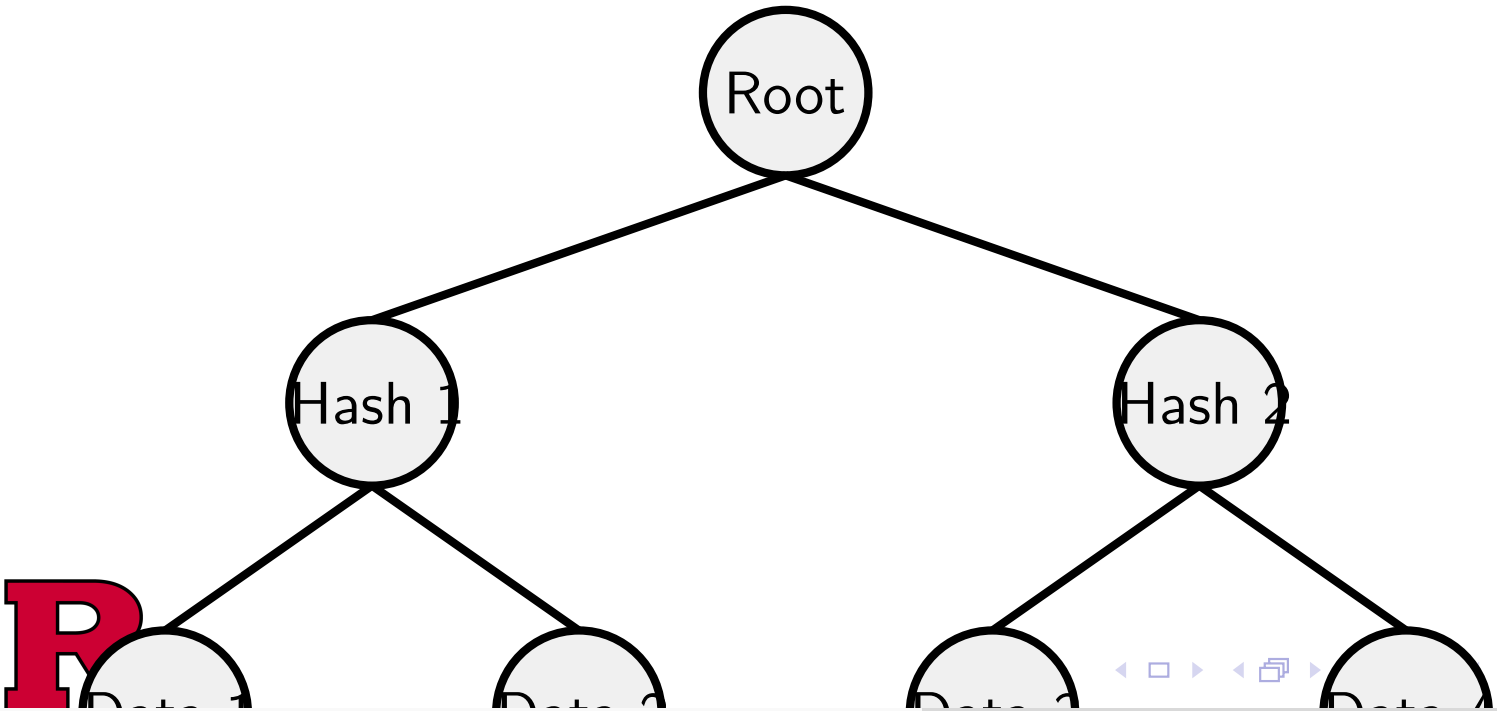
Elliptic Curve Cryptography (ECC)

- Elliptic Curve Cryptography (ECC) is a form of public key cryptography based on the algebraic structure of elliptic curves over finite fields.
- ECC offers equivalent security to RSA and other cryptographic systems but with smaller key sizes, making it more efficient for constrained environments like blockchain.
- ECC is widely used in blockchain for generating key pairs, digital signatures, and key exchange protocols.



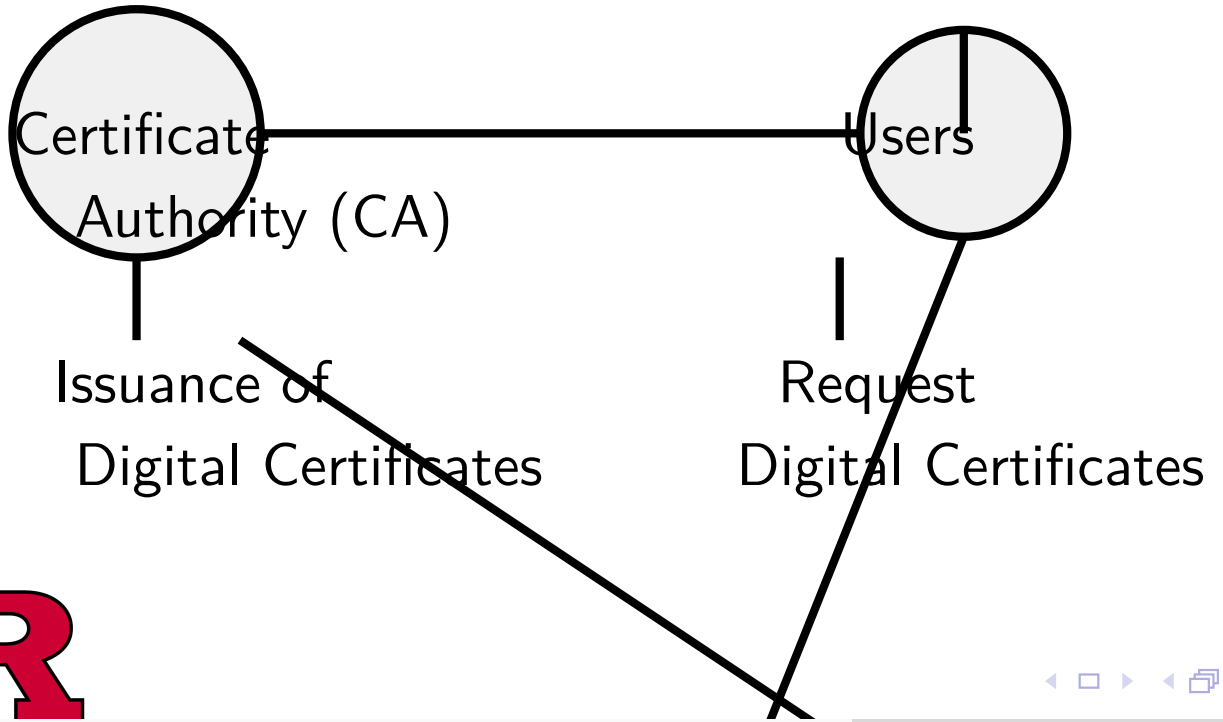
Merkle Trees and Merkle Proof

- Merkle trees are data structures that enable efficient and secure verification of large datasets.
- In blockchain, Merkle trees are used to summarize the transaction history within a block, allowing for quick verification of individual transactions.
- Merkle proofs provide a compact way to prove the inclusion or absence of a particular transaction in a block without revealing the entire block's contents.



Public Key Infrastructure (PKI)

- Public Key Infrastructure (PKI) is a set of hardware, software, policies, and standards used to manage digital certificates and public-private key pairs.
- In blockchain, PKI enables the secure issuance, distribution, and revocation of digital certificates, facilitating secure communication and transaction validation.
- PKI plays a crucial role in establishing trust and authenticity within blockchain networks.



Consensus Algorithms in Blockchain

- Consensus algorithms are protocols used to achieve agreement among distributed nodes in a blockchain network.
- Common consensus algorithms include
  - Proof of Work (PoW)
  - Proof of Stake (PoS)
  - Delegated Proof of Stake (DPoS)
  - Practical Byzantine Fault Tolerance (PBFT)
- Consensus algorithms ensure the integrity and security of the blockchain by enabling decentralized decision-making and preventing double-spending attacks.

Byzantine Fault Tolerance (BFT)

- BFT is a property of distributed systems that can tolerate the failure of a certain number of nodes or malicious actors.
- In blockchain, BFT consensus algorithms ensure that the network remains operational and secure even in the presence of faulty or malicious nodes.
- BFT algorithms use redundancy and cryptographic mechanisms to achieve consensus and prevent Byzantine failures.

Smart Contracts and Solidity

- Smart contracts are self-executing contracts with the terms of the agreement directly written into code.
- Solidity is a high-level programming language used to write smart contracts on blockchain platforms like Ethereum.
- Smart contracts enable decentralized applications (DApps) to automate and enforce the execution of agreements, transactions, and other processes without the need for intermediaries.
- Solidity, part of Ethereum is the most widely used programming language for smart contracts.

Solidity

- Solidity is a contract-oriented, high-level programming language for implementing smart contracts
- Based on C++, Python and JavaScript
- Targets the Ethereum Virtual Machine (EVM)
- EVM is designed to run untrusted code by computers all over the world
- docs: `https://solidity.readthedocs.io/`
- Tutorial: `https://www.tutorialspoint.com/solidity/index.htm`

Solidity Example

Smart Contract Attacks (Solidity)

- The code is your contract
- Re-entrancy Attack Possibility to call a function multiple times before it has completed
- Default Visibilities Functions used internally should not be visible (and callable) to the outside world
- Arithmetic under/overflows
- Entropy Illusion
- Race Conditions
- Denial of Service

`https://hacken.io/discover/most-common-smart-contract-attacks/`

Famous Smart Contract Failures

- The reentrancy attack led to hundreds of millions of dollars in losses
- Ethereum fork in 2016 to change the code and rewrite history
- DAO Hack 2016 $60 million

ucination! OpenZeppelin 2019 $100 million

- Grim Finance 2021 $30 million
- DFORECE Apr 2020 $24 million

DAO Hack 2016

- Seized 5.2 percent of ETH
- Required a fork of Ethereum to "fix"
- While the outcome may be good, effectively Ethereum is governed by an oligopoly of programmers?
- No court of law decides what will happen

Example of Reentrancy Attack

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

  function deposit() public payable{
    require(msg.value > 0, "funds needed to set bal
    balances[msg.sender] += msg.value;
  }

  function withdraw() public {
    require(balances[msg.sender] > 0, "insufficient
    (bool success, ) = msg.sender.call{value: balan
    require(success, "transfer failed");
    balances[msg.sender] = 0;
  }
}
```

Reentrancy Attack

https://www.infuy.com/blog/preventing-re-entrancy-attacks-in-solidity/ contract BankAttack Bank bank;
function setBankContract(address bankContract) public bank = Bank(bankContract);
fallback () external payable if (address(bank).balance >= 1 ether) bank.withdraw();
function attack() external payable require(msg.value == 1 ether); bank.depositvalue: 1 ether(); bank.withdraw();

Solution

```
contract Bank {
  bool private _entered;

  modifier nonReentrant {
    require(!_entered, "re-entrant call");
    _entered = true;
    _;
    _entered = false;
  }

  function withdraw() public nonReentrant {
    require(balances[msg.sender] > 0, "insufficient
    (bool success, ) = msg.sender.call{value: balan
    require(success, "transfer failed");
    balances[msg.sender] = 0;
  }
}
```

Decentralized Applications (DApps)

- Decentralized Applications (DApps) are software applications that run on a distributed computing system like blockchain.
- DApps leverage the decentralized nature of blockchain to provide transparency, security, and censorship resistance.
- Examples of DApps include decentralized finance (DeFi) platforms, decentralized exchanges (DEXs), and blockchain-based games.

Summary: Usefulness of Blockchain

- Public blockchain does not appear to be useful except to criminals
- Being able to have multiple parties validate a blockchain so records are immutable seems to be useful
- Problems stem from changing technology: How do we verify the integrity of a blockchain when the cryptosystem can be broken?
- Cryptocurrency as an investment is speculation based on zero value