Cryptographic Foundations

Lesson 3: The Math Behind Blockchain Security

Prof. Joerg Osterrieder

Spring 2026

Learning Objectives

After this lesson, you will be able to:

- Explain how cryptographic hash functions work and their key properties
- Describe public/private key cryptography and its role in blockchain
- Understand how digital signatures prove ownership
- Explain Merkle trees and their efficiency benefits

Prerequisites: Lesson 2 (Blockchain Fundamentals)

Crypt

is the mathematical foundation of blockchain security

Lesson Outline

1. Hash Functions
2. Public Key Cryptography
3. Digital Signatures
4. Merkle Trees
5. Putting It Together

Hash Functions

What is a Hash Function?

Definition: A hash function takes any input and produces a fixed-size output (the "hash" or "digest").

Mathematical Notation:

$$H : \{0,1\}^* \to \{0,1\}^n$$

Example (SHA-256):

- Input: Any data (text, file, transaction)
- Output: 256-bit string (64 hexadecimal characters)
- "Hello" $\to$ `185f8db32271...` (64 chars)

Hash

functions create unique "fingerprints" of data

Key Properties of Cryptographic Hash Functions

Five Essential Properties:

1. **Deterministic**: Same input always produces same output
2. **Fast**: Computing the hash is computationally efficient
3. **One-way**: Cannot reverse-engineer input from output
4. **Collision-resistant**: Extremely hard to find two inputs with same hash
5. **Avalanche effect**: Small input change = completely different output

These properties make hash functions ideal for blockchain integrity

The Avalanche Effect

Chan

one character produces a completely different hash

SHA-256: The Bitcoin Standard

SHA-256 (Secure Hash Algorithm 256-bit):

- Developed by NSA, published in 2001
- Produces 256-bit (32-byte) output
- Used in Bitcoin for block hashing and addresses

Security:

- 2^{256} possible outputs ($\approx 10^{77}$)
- Nearly as many combinations as atoms in the observable universe (10^{77} vs 10^{80})—astronomically large
- No known practical attacks

SHA-256 has remained secure since Bitcoin's creation in 2009

Hash Functions in Blockchain

Three Primary Uses:

1. **Block linking**: Each block contains hash of previous block
2. **Transaction fingerprints**: Identify transactions uniquely
3. **Mining puzzle**: Find nonce such that $H(\text{block}) < \text{target}$

Why These Properties Matter:

- One-way: Cannot forge transactions
- Collision-resistant: Cannot create fake blocks
- Avalanche: Cannot predict valid nonces

Hash functions provide data integrity without revealing the data

Public Key Cryptography

Symmetric vs. Asymmetric Cryptography

Symmetric (Shared Key):

- Same key for encryption and decryption
- Problem: How to share the key securely?
- Example: AES

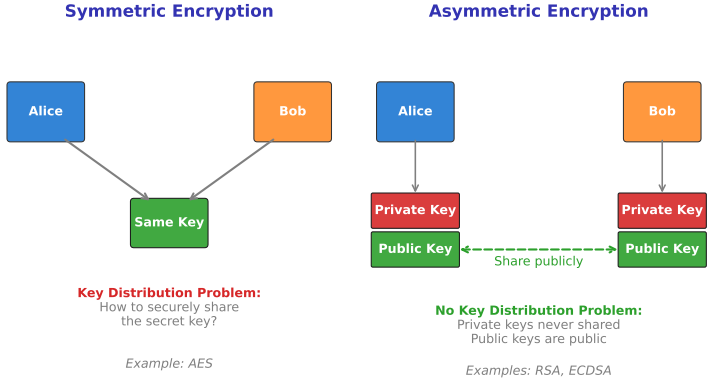
Asymmetric (Public/Private Key):

- Two mathematically related keys
- Public key: Share with everyone
- Private key: Keep secret
- Example: RSA, ECDSA

Block

uses asymmetric cryptography for ownership proof

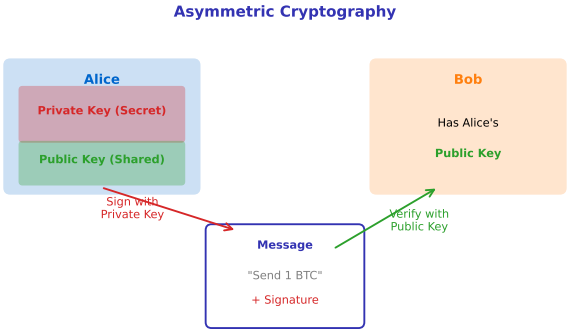
Symmetric vs. Asymmetric: Visual Comparison



Asym

encryption solves the key distribution problem of symmetric encryption

Public and Private Keys



Priva

key signs, public key verifies—cannot derive private from public

Key Generation in Bitcoin

Elliptic Curve Cryptography (secp256k1):

1. Generate random 256-bit number = **Private Key**
2. Multiply by generator point G: Public Key = $k \times G$
3. Hash public key = **Bitcoin Address**

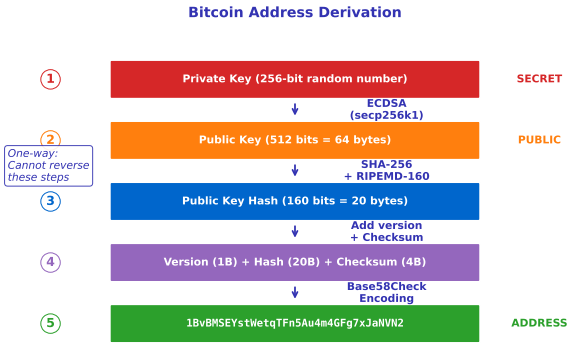
Security Properties:

- Easy: Calculate public from private (multiplication)
- Hard: Calculate private from public (discrete log problem)
- Computationally infeasible to reverse

2^{256}

possible private keys ensures collision is practically impossible

From Private Key to Bitcoin Address



The derivation is one-way: you cannot reverse-engineer a private key from an address

Why Elliptic Curves?

Advantages over RSA:

- Smaller key sizes for equivalent security
- 256-bit ECC $\approx$ 3072-bit RSA
- Faster computation
- Less storage and bandwidth

The secp256k1 Curve:

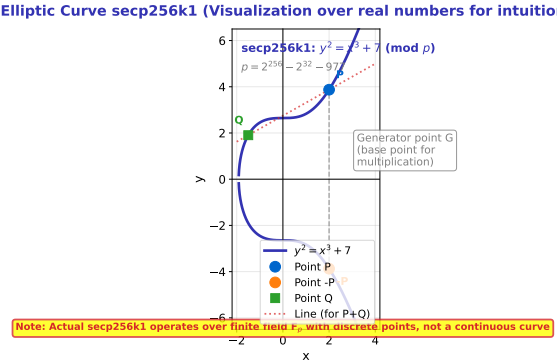
$$y^2 = x^3 + 7 \pmod{p}$$

where p is a large prime (256 bits)

Ether

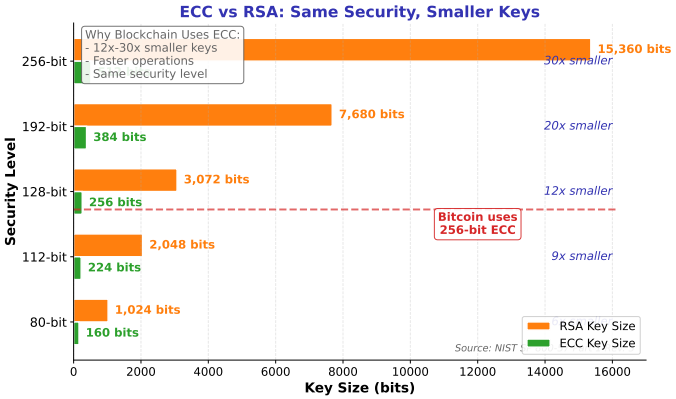
and Bitcoin both use secp256k1 for key generation

The secp256k1 Curve Visualized



Point addition and scalar multiplication are easy; finding the scalar from points is computationally infeasible

ECC vs RSA: Key Size Comparison



ECC provides equivalent security with 12-30x smaller keys—critical for blockchain efficiency

Digital Signatures

What is a Digital Signature?

Definition: Cryptographic proof that a message was created by a specific person.

Properties:

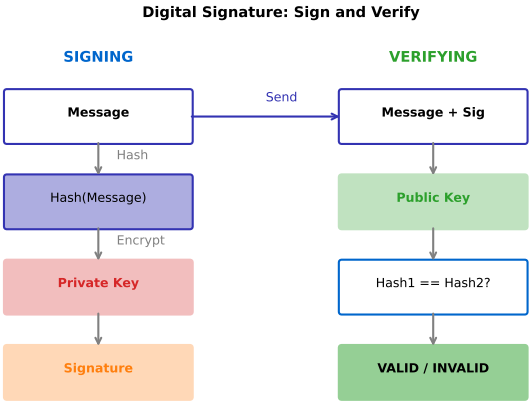
- **Authentication**: Proves who signed
- **Non-repudiation**: Signer cannot deny signing
- **Integrity**: Detects if message was altered

In Blockchain:

- Every transaction is signed by sender
- Proves ownership of funds
- Cannot be forged without private key

signatures are the proof of ownership in blockchain

Signing and Verifying



Sign with private key, verify with public key—anyone can verify

ECDSA: The Signature Algorithm

Elliptic Curve Digital Signature Algorithm:
Signing:

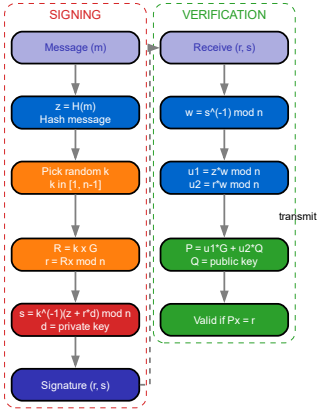
1. Hash the message: $z = H(m)$
2. Pick random k, compute $R = k \times G$
3. Compute $s = k^{-1}(z + r \cdot d) \mod n$
4. Signature $= (r, s)$

Verification:

1. Compute $u_1 = z \cdot s^{-1}$ and $u_2 = r \cdot s^{-1}$
2. Compute point $P = u_1 \times G + u_2 \times Q$
3. Valid if $P_x = r$

The math ensures only the private key holder can create valid signatures

ECDSA: Visual Process Flow



Signa

(r, s) verifiable by anyone with public key, but only created with private key

Transaction Signing Example

Alice sends 1 BTC to Bob:

1. Alice creates transaction: "Send 1 BTC to Bob's address"
2. Alice hashes the transaction data
3. Alice signs with her private key
4. Alice broadcasts: Transaction + Signature
5. Network verifies using Alice's public key
6. If valid, transaction is added to mempool

Key Insight: Alice proves ownership without revealing her private key.

Signa

enable trustless transfer of digital assets

Transaction Signing: Visual Flow

The signature travels with the transaction, allowing anyone to verify authenticity

Merkle Trees

What is a Merkle Tree?

Definition: A binary tree of hashes where each non-leaf node is the hash of its children.

Structure:

- **Leaves**: Hashes of individual transactions
- **Intermediate nodes**: Hash of child nodes
- **Root**: Single hash representing all transactions

In Blockchain:

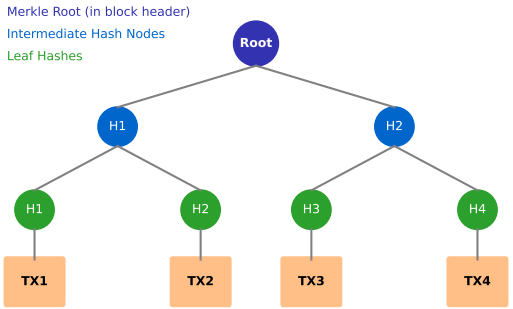
- Merkle Root stored in block header
- Efficiently summarizes all block transactions

Name

after Ralph Merkle who patented the concept in 1979

Merkle Tree Structure

Merkle Tree: Efficient Transaction Verification



`Root = H( H(H(TX1)||H(TX2)) || H(H(TX3)||H(TX4)) )`

The root hash changes if any transaction is modified

Merkle Proofs: Efficient Verification

Problem: Verify a transaction is in a block without downloading all transactions.

Solution: Merkle Proof

- Need only $\log_2(n)$ hashes to verify
- For 1000 transactions: only 10 hashes needed
- For 1 million transactions: only 20 hashes needed

How It Works:

1. Request proof for specific transaction
2. Receive transaction + sibling hashes on path to root
3. Recompute root and compare with block header

Merk

proofs enable light clients on mobile devices

SPV: Simplified Payment Verification

Full Node:

- Downloads entire blockchain (720 GB)
- Validates all transactions
- Maximum security

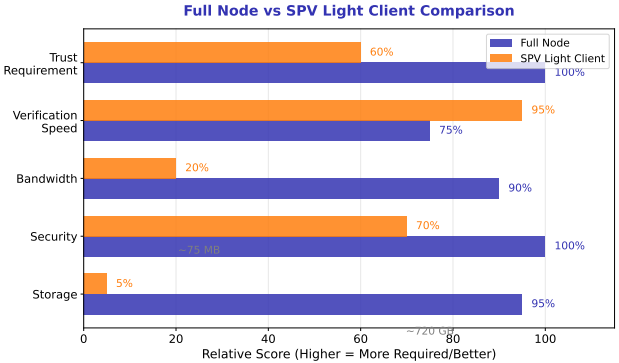
Light Client (SPV):

- Downloads only block headers (75 MB)
- Uses Merkle proofs to verify transactions
- Trusts that longest chain is valid
- Suitable for mobile wallets

SPV

was described in the original Bitcoin whitepaper

SPV vs Full Node: Comparison



SPV

enables mobile wallets with trade-offs in security and decentralization

Putting It Together

Cryptography in a Bitcoin Transaction

Complete Flow:

1. **Keys**: Alice generates private/public key pair
2. **Address**: Hash of public key becomes Bitcoin address
3. **Transaction**: Alice creates transfer to Bob
4. **Signature**: Alice signs with private key
5. **Verification**: Network verifies signature
6. **Merkle Tree**: Transaction added to block's Merkle tree
7. **Block Hash**: Block header (including Merkle root) is hashed
8. **Chain**: Block hash links to next block

Every step uses cryptographic primitives we covered today

Complete Bitcoin Cryptography Flow

All

8 cryptographic steps from key generation to blockchain linkage

Security Assumptions

Blockchain Security Relies On:

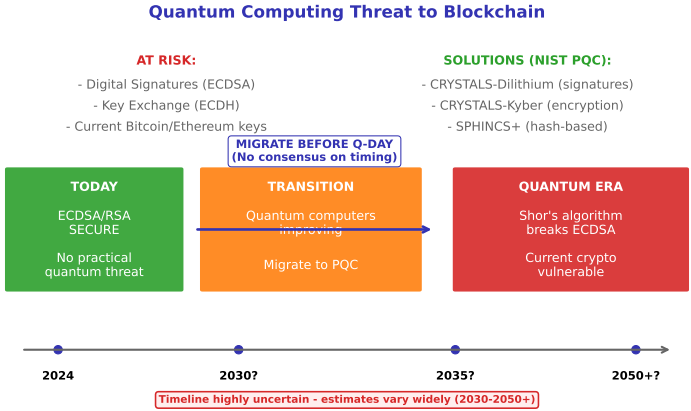
- SHA-256 remains collision-resistant
- Discrete logarithm problem remains hard
- ECDSA signatures cannot be forged
- Random number generation is truly random

Potential Threats:

- Quantum computers could break ECDSA
- SHA-256 weaknesses discovered
- Poor random number generation

Post-quantum cryptography research is ongoing

The Quantum Computing Threat



NIST standardized post-quantum algorithms in 2024—blockchain migration planning has begun

Preparing for Post-Quantum Cryptography

Shor's Algorithm Threatens:

- ECDSA (signatures)—basis of Bitcoin/Ethereum security
- RSA (encryption)
- Diffie-Hellman key exchange

NIST Post-Quantum Standards (2024):

- **CRYSTALS-Dilithium**: Digital signatures (likely Bitcoin successor)
- **CRYSTALS-Kyber**: Key encapsulation/encryption
- **SPHINCS+**: Hash-based signatures (conservative option)

Timeline: Cryptographically relevant quantum computers estimated 2030-2040. These estimates are highly uncertain and debated among experts.

"Harv

now, decrypt later" attacks mean migration should begin before quantum computers exist

Key Takeaways

What You Learned Today:

1. Hash functions create unique, fixed-size fingerprints with avalanche effect
2. Public/private keys enable asymmetric cryptography
3. Digital signatures prove ownership without revealing secrets
4. Merkle trees enable efficient verification with $O(\log n)$ complexity

Core Insight: Blockchain security is built on mathematical hardness assumptions—problems that are easy to verify but practically impossible to reverse.

Next lesson: Consensus Mechanisms

Questions for Reflection

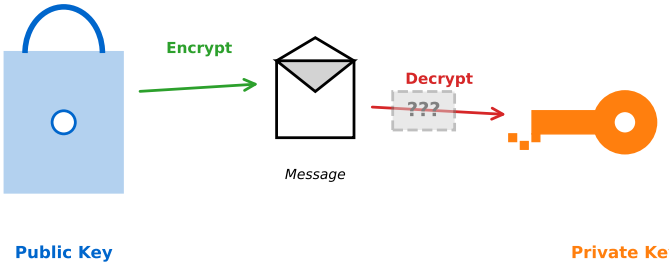
1. Why is the one-way property of hash functions essential for mining?
2. How does ECDSA prove ownership without revealing the private key?
3. Why are Merkle trees more efficient than hashing all transactions together?
4. What would happen to Bitcoin if SHA-256 was broken?

Discussion: Should blockchain systems prepare for quantum computing threats now?

Consi

these questions before our next session

Public Key Cryptography



Data Sources and References

Cryptographic Standards and Documentation:

- NIST Cryptographic Standards
- Bitcoin Wiki - secp256k1
- SEC 2 Elliptic Curve Standards
- BIP-39 Mnemonic Standard
- Python Cryptography Library Docs

Note: These resources provide technical specifications and implementation details for the cryptographic primitives discussed in this lesson.

All

URLs verified as of January 2026

Thank You

Questions?

Course materials: digital-ai-finance.github.io/crypto-economics

Appendix: Mathematical Foundations

Technical: ECDSA Mathematics

Key Generation:

- Choose random $d \in [1, n-1]$ (private key)
- Compute $Q = d \times G$ (public key)
- G is the generator point, n is the curve order

Signature Generation:

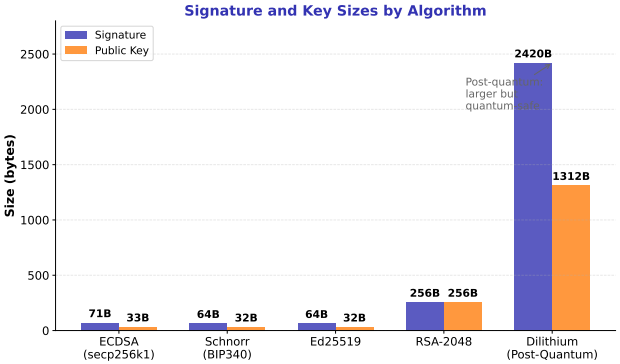
1. Hash message: $e = H(m)$
2. Choose random $k \in [1, n-1]$
3. Compute $(x, y) = k \times G$, set $r = x \mod n$
4. Compute $s = k^{-1}(e + rd) \mod n$
5. Signature is (r, s)

Signature Verification:

1. Compute $w = s^{-1} \mod n$
2. Compute $u_1 = ew \mod n$ and $u_2 = rw \mod n$
3. Compute $(x', y') = u_1 \times G + u_2 \times Q$
4. Valid if $r = x' \mod n$

The security relies on the difficulty of the discrete logarithm problem on elliptic curves

Signature Algorithm Sizes Comparison



Post-quantum signatures (Dilithium) are larger but provide quantum resistance

Technical: Hash Collision Mathematics

Birthday Paradox:
For a hash function with n-bit output, expected collisions after $\sqrt{2^n} = 2^{n/2}$ hashes.

For SHA-256 (256-bit output):

- Expected collision after 2^{128} hashes
- At 1 trillion hashes/second: $\approx 10^{22}$ years
- Universe age: $\approx 1.4 \times 10^{10}$ years

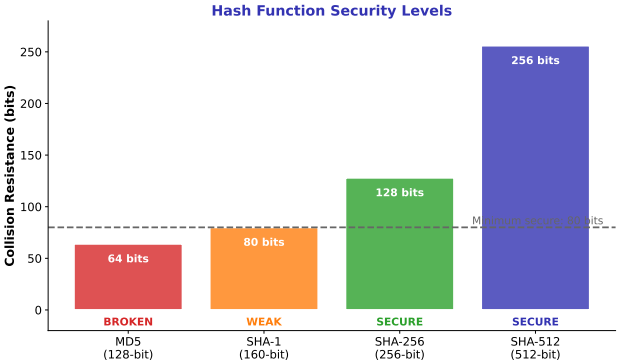
Preimage Resistance:
Finding input x such that $H(x) = y$ requires 2^{256} attempts on average.

Bitcoin Mining Context:

- Network performs $\approx 600 \times 10^{18}$ hashes/second
- Still negligible compared to 2^{256} ($\approx 10^{77}$)

SHA-256 security margins are enormous by any practical measure

Hash Function Security Comparison



MD5 and SHA-1 are considered broken; SHA-256 and SHA-512 remain secure

Technical: Bitcoin Address Formats

Legacy (P2PKH) - starts with "1":

- Original format: RIPEMD-160(SHA-256(pubkey))
- Base58Check encoding
- Example: 1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2

SegWit (P2WPKH) - starts with "bc1q":

- Native SegWit format (BIP-84)
- Bech32 encoding (lowercase, no ambiguous chars)
- Lower fees, better error detection
- Example: bc1qar0srrr7xfkvy5l643lydnw9re59gtzzwf5mdq

Taproot (P2TR) - starts with "bc1p":

- Newest format (BIP-86, activated 2021)
- Enhanced privacy and smart contract capability
- Example: bc1p5d7rjq7g6rdk2yhzks9smlaqtedr4dekq08ge8ztwac72sfr9rusxg3297

Newe

formats offer lower fees and enhanced features

Technical: Entropy and Key Security

Private Key Entropy Requirements:

- Bitcoin private key: 256 bits of entropy
- Must be cryptographically random
- Poor randomness = predictable keys = stolen funds

Sources of Randomness:

- **Good**: Hardware RNG (CPU RDRAND), /dev/urandom
- **Bad**: Time-based seeds, weak PRNGs
- **Famous failure**: Android SecureRandom bug (2013)

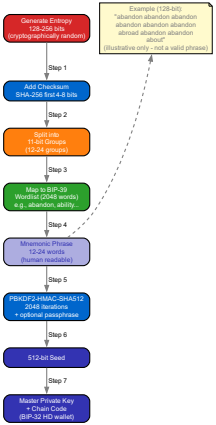
BIP-39 Mnemonic Seeds:

- 12-24 words encoding 128-256 bits of entropy
- Human-readable backup format
- Checksum prevents transcription errors
- Example: "abandon abandon ... about" (test vector)

Neve

generate keys with weak randomness—use established wallets

BIP-39: Mnemonic Seed Generation



The 12-24 word phrase encodes entropy and enables deterministic key derivation