Public Key Cryptography

BSc Blockchain, Crypto Economy & NFTs

Course Instructor

Module A: Blockchain Foundations

Learning Objectives

By the end of this lesson, you will be able to:

- Explain the fundamental difference between symmetric and asymmetric cryptography
- Understand how public-private key pairs enable secure communication
- Describe the Elliptic Curve Digital Signature Algorithm (ECDSA)
- Generate and verify digital signatures
- Derive blockchain addresses from public keys
- Recognize key security best practices for cryptocurrency wallets

Symmetric vs. Asymmetric Cryptography

Symmetric Cryptography

- Same key for encryption and decryption
- Fast and efficient
- Key distribution problem
- Examples: AES, DES, 3DES

Use Case:

- Encrypting large data volumes
- Disk encryption
- VPN tunnels

Asymmetric Cryptography

- Different keys: public and private
- Slower than symmetric
- No key distribution problem
- Examples: RSA, ECC, ECDSA

Use Case:

- Digital signatures
- Key exchange
- Blockchain transactions

Key Insight: Blockchains rely exclusively on asymmetric cryptography for identity and authorization

The Key Distribution Problem

Symmetric Encryption Challenge:

Alice wants to send an encrypted message to Bob:

1. Alice encrypts message with secret key K
2. Alice sends encrypted message to Bob
3. Bob needs key K to decrypt
4. **Problem:** How does Alice securely share K with Bob?

Traditional Solutions:

- Physical key exchange (courier, in person)
- Pre-shared keys (exchanged before communication)
- Key distribution centers (trusted third parties)

Asymmetric Cryptography Solution:

- Bob publishes his public key openly
- Alice encrypts with Bob's public key
- Only Bob's private key can decrypt
- No secret key exchange needed

Public-Private Key Pairs

Core Concept:

- Mathematically related pair of keys
- Public key: freely shared, used to verify signatures
- Private key: kept secret, used to create signatures
- One-way mathematical relationship (hard to derive private from public)

Mathematical Property:
If Alice has key pair (PK_A, SK_A):

- Message encrypted with PK_A can only be decrypted with SK_A
- Signature created with SK_A can only be verified with PK_A

Blockchain Usage:

- Private key = your "password" to control funds
- Public key (or address) = your account identifier
- Losing private key = losing access to funds permanently
- No password reset mechanism exists

RSA vs. Elliptic Curve Cryptography

RSA (Rivest-Shamir-Adleman)

- Based on factoring large primes
- Key sizes: 2048-4096 bits
- Computationally intensive
- Well-established (1977)
- Used in TLS/SSL certificates

Security:

- 2048-bit key = 112-bit security
- 3072-bit key = 128-bit security

Why Blockchains Use ECC:

- Smaller key sizes = less blockchain storage
- Faster signature verification
- Mobile-friendly computation

Elliptic Curve (ECC)

- Based on discrete logarithm problem
- Key sizes: 256 bits
- More efficient computation
- Newer (1985)
- Used in Bitcoin, Ethereum

Security:

- 256-bit key = 128-bit security
- Equivalent to 3072-bit RSA

Elliptic Curve Mathematics: The Basics

Elliptic Curve Equation:

$$y^2 = x^3 + ax + b$$

Bitcoin's Curve (secp256k1):

$$y^2 = x^3 + 7 \quad \text{(parameters: } a = 0, b = 7\text{)}$$

Key Mathematical Properties:

- Defined over finite field (modulo large prime)
- Point addition operation creates a group
- Scalar multiplication: $Q = k \cdot G$ (k = private key, G = generator point, Q = public key)
- Easy to compute Q from k, infeasible to compute k from Q

Intuition:

- Private key = random 256-bit number
- Public key = point on elliptic curve (x, y coordinates)
- Trapdoor function ensures one-way relationship

Elliptic Curve Point Addition

Geometric Interpretation (simplified):

To add points P and Q on an elliptic curve:

1. Draw a line through P and Q
2. Line intersects curve at third point R'
3. Reflect R' across x-axis to get R = P + Q

Point Doubling (P + P):

- Draw tangent line at point P
- Find intersection with curve
- Reflect to get 2P

Scalar Multiplication:

$$k \cdot G = \underbrace{G + G + \cdots + G}_{k \text{ times}}$$

Computed efficiently using double-and-add algorithm (similar to exponentiation by squaring)

Key Generation:

- Choose random k (private key)
- Compute $Q = k \cdot G$ (public key)

The Discrete Logarithm Problem

Security Foundation:

Given:

- Generator point G (known)
- Public key $Q = k \cdot G$ (known)

Find:

- Private key k (unknown)

Computational Complexity:

- No known efficient algorithm to solve this
- Best attack: Pollard's rho algorithm
- For 256-bit keys: requires 2^{128} operations (infeasible)
- Quantum computers: Shor's algorithm reduces to polynomial time (future threat)

Practical Implication:

- Cannot derive private key from public key
- Cannot forge signatures without private key
- Blockchain security relies on this hardness assumption

Digital Signatures: Purpose and Properties

What is a Digital Signature?

- Cryptographic proof that a message was created by the holder of a private key
- Anyone with the public key can verify authenticity
- Provides authentication, integrity, and non-repudiation

Required Properties:

1. **Authentication:** Proves who created the signature
2. **Integrity:** Any modification to message invalidates signature
3. **Non-repudiation:** Signer cannot deny creating signature
4. **Unforgeable:** Impossible to create valid signature without private key

Blockchain Applications:

- Authorize cryptocurrency transactions
- Prove ownership of funds
- Create immutable audit trails
- Smart contract execution authorization

ECDSA: Elliptic Curve Digital Signature Algorithm

Signature Generation:

Given message m and private key k:

1. Compute message hash: $h = \text{SHA-256}(m)$
2. Generate random nonce: n (must be unique per signature)
3. Compute point: $R = n \cdot G$, extract x-coordinate as r
4. Compute: $s = n^{-1}(h + r \cdot k) \mod p$
5. Signature is pair: (r, s)

Signature Verification:

Given message m, signature (r, s), and public key Q:

1. Compute message hash: $h = \text{SHA-256}(m)$
2. Compute: $u_1 = h \cdot s^{-1} \mod p$, $u_2 = r \cdot s^{-1} \mod p$
3. Compute point: $R' = u_1 \cdot G + u_2 \cdot Q$
4. Extract x-coordinate r'
5. Verify: $r' = r$ (signature valid if true)

Critical Security: Nonce Reuse Vulnerability

The Danger of Reusing Nonces:

If the same nonce n is used to sign two different messages m_1 and m_2:

- Attacker obtains signatures: (r, s_1) and (r, s_2)
- Notice that r is the same (same nonce used)
- Can compute: $s_1 - s_2 = n^{-1}(h_1 - h_2) \mod p$
- Solve for n, then solve for private key k

Real-World Incident: Sony PlayStation 3 (2010)

- Sony used ECDSA to sign PS3 firmware
- Used the same nonce for all signatures
- Hackers recovered Sony's private key
- Enabled homebrew software and piracy

Best Practice:

- Use deterministic nonce generation (RFC 6979)
- Nonce derived from message hash and private key
- Ensures uniqueness without randomness

From Private Key to Blockchain Address

Bitcoin Address Derivation:

1. Generate random 256-bit private key k
2. Compute public key: $Q = k \cdot G$ (33 bytes compressed or 65 bytes uncompressed)
3. Hash public key: $h_1 = \text{SHA-256}(Q)$
4. Hash again: $h_2 = \text{RIPEMD-160}(h_1)$ (20 bytes)
5. Add version byte: $0x00$ for mainnet (21 bytes)
6. Compute checksum: first 4 bytes of SHA-256(SHA-256(version + h_2))
7. Concatenate: version + h_2 + checksum (25 bytes)
8. Encode in Base58: produces address like "1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa"

Why Multiple Hash Functions?

- SHA-256: widely trusted, fast
- RIPEMD-160: reduces address size (160 bits instead of 256 bits)
- Double hashing: additional security layer

Ethereum Address Derivation

Simplified Process:

1. Generate random 256-bit private key k
2. Compute public key: $Q = k \cdot G$ (uncompressed, 64 bytes excluding prefix)
3. Hash public key: $h = \text{Keccak-256}(Q)$ (32 bytes)
4. Take last 20 bytes of hash
5. Add "0x" prefix
6. Result: address like "0x742d35Cc6634C0532925a3b844Bc9e7595f0bEb"

Key Differences from Bitcoin:

- Uses Keccak-256 instead of SHA-256 + RIPEMD-160
- No checksum in base address (EIP-55 adds mixed-case checksum)
- Hexadecimal encoding instead of Base58
- Always 20 bytes (40 hex characters)

EIP-55 Checksum:

- Optional mixed-case encoding (e.g., "0x5aAeb6053f3E94C9b9A09f33669435E7Ef1BeAed")
- Capitalization pattern encodes checksum
- Detects typos without changing address length

Public Key Compression

Uncompressed Public Key (65 bytes):

- Format: $04 \| x \| y$
- Prefix: 0x04 (1 byte)
- x-coordinate: 32 bytes
- y-coordinate: 32 bytes
- Total: 65 bytes

Compressed Public Key (33 bytes):

- Format: $02/03 \| x$
- Prefix: 0x02 (if y is even) or 0x03 (if y is odd)
- x-coordinate: 32 bytes
- Total: 33 bytes
- y-coordinate can be reconstructed from x (elliptic curve equation)

Why Compression Matters:

- Reduces blockchain storage by 50%
- Faster transaction propagation
- Lower transaction fees (Bitcoin SegWit incentivizes compressed keys)
- Bitcoin: both formats valid, Ethereum: only uncompressed

Signing a Bitcoin Transaction

Transaction Structure (Simplified):

- Inputs: references to previous transaction outputs (UTXOs)
- Outputs: new recipient addresses and amounts
- Signature: proves ownership of input UTXOs

Signing Process:

1. Construct transaction with inputs and outputs
2. Serialize transaction data
3. Append signature hash type (SIGHASH_ALL)
4. Compute double SHA-256 hash of serialized data
5. Sign hash using ECDSA with private key
6. Append public key and signature to transaction

Verification by Network Nodes:

1. Extract public key from transaction
2. Verify signature using ECDSA verification
3. Derive address from public key
4. Confirm address matches UTXO owner
5. Accept transaction if signature valid

Key Security Best Practices

Private Key Protection:

- Never share private keys or seed phrases
- Use hardware wallets (Ledger, Trezor) for large amounts
- Store backups in multiple secure physical locations
- Use strong passwords for wallet encryption
- Avoid storing keys on internet-connected devices

Common Threats:

- Phishing attacks (fake wallet websites)
- Clipboard malware (replaces copied addresses)
- Keyloggers (record password entry)
- SIM swapping (hijack 2FA)
- Physical theft (unencrypted paper wallets)

Defense Strategies:

- Multi-signature wallets (require M-of-N keys)
- Air-gapped signing (offline devices)
- Regular security audits
- Test small amounts before large transfers

Hierarchical Deterministic Wallets (BIP-32)

Problem with Random Key Generation:

- Need to back up every new private key separately
- Managing hundreds of keys becomes impractical
- Risk of losing individual keys

HD Wallet Solution:

- Generate all keys from single master seed (usually 12-24 words)
- Deterministic derivation: same seed always produces same keys
- Hierarchical structure: master key -¿ account keys -¿ address keys
- Only need to back up seed phrase once

Derivation Path Example:

- Bitcoin: `m/44'/0'/0'/0/0`
- Ethereum: `m/44'/60'/0'/0/0`
- Each level represents: purpose / coin type / account / change / index

BIP-39 Mnemonic:

- Seed phrase = 12-24 English words from standard wordlist
- Easy to write down and remember
- Encodes 128-256 bits of entropy

Multi-Signature Wallets

Concept:

- Require multiple signatures to authorize a transaction
- Common schemes: 2-of-3, 3-of-5, etc.
- M-of-N threshold: need M signatures out of N total keys

Use Cases:

- Corporate treasuries (require CEO + CFO approval)
- Estate planning (family members hold keys)
- Escrow services (buyer + seller + mediator)
- Enhanced security (attacker must compromise multiple keys)

Bitcoin Implementation:

- P2SH (Pay-to-Script-Hash) addresses start with "3"
- Redeem script specifies signature requirements
- All participating public keys embedded in script

Ethereum Implementation:

- Smart contract-based (e.g., Gnosis Safe)
- More flexible logic (time delays, spending limits)
- Can add/remove signers dynamically

Key Takeaways

- Public key cryptography solves the key distribution problem
- Private keys must remain absolutely secret
- ECDSA provides efficient digital signatures for blockchains
- Nonce reuse is a critical vulnerability
- Bitcoin and Ethereum use different address derivation schemes
- HD wallets enable convenient key management from a single seed
- Multi-signature wallets add an extra security layer

Core Principle:
"Not your keys, not your coins."
If you do not control the private keys to your cryptocurrency addresses, you do not truly own the funds. Exchanges and custodians can be hacked, frozen, or shut down.

Discussion Questions

1. Why do blockchains use elliptic curve cryptography instead of RSA?
2. What would happen if two users randomly generated the same private key?
3. How does address derivation protect privacy compared to reusing the same address?
4. Why is the discrete logarithm problem crucial for blockchain security?
5. What are the trade-offs between convenience and security when storing private keys?
6. How would quantum computing impact ECDSA-based blockchains?

Next Lesson Preview: L06 Bitcoin Protocol Deep Dive

Topics to be covered:

- Unspent Transaction Output (UTXO) model
- Bitcoin transaction structure in detail
- Transaction inputs and outputs
- Bitcoin Script language
- Transaction verification process
- Common transaction types (P2PKH, P2SH, SegWit)
- Transaction lifecycle from creation to confirmation

Preparation:

- Review how digital signatures work
- Explore a Bitcoin block explorer (e.g., blockchain.com)
- Observe transaction structure in real blocks