Lesson 03 — Cryptographic Foundations

Study Notes

Cryptoeconomics — BSc Level

Joerg Osterrieder

2025

Contents

Learning Objectives

By the end of this lesson, students should be able to:

1. **State** the five properties of a cryptographic hash function and explain why each is required for blockchain security.
2. **Calculate** the birthday paradox collision bound for SHA-256 and interpret its practical implications.
3. **Describe** asymmetric key-pair cryptography and explain the mathematical intuition behind elliptic curve operations.
4. **Trace** the ECDSA signing and verification process step-by-step, identifying where the private key is used.
5. **Derive** a Bitcoin address from a private key using the full pipeline: private key → public key → hash → address.
6. **Explain** BIP-39 mnemonic seeds and BIP-32/44 HD wallet derivation paths.
7. **Assess** the threat quantum computing poses to current blockchain cryptography and name the NIST post-quantum alternatives.

Hash Functions Deep Dive

Formal Definition and Properties

Cryptographic Hash Function

A **cryptographic hash function** is a deterministic map

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

taking an input of arbitrary length to a fixed-length n-bit digest. For blockchain applications, five properties are required:

1. **Deterministic**: $H(x) = H(x)$ always.
2. **Efficient**: $H(x)$ computable in polynomial time.
3. **Pre-image resistant (one-way)**: Given y, finding x such that $H(x) = y$ requires $\Theta(2^n)$ work.
4. **Second pre-image resistant**: Given x, finding $x' \neq x$ with $H(x') = H(x)$ requires $\Theta(2^n)$ work.
5. **Collision resistant**: Finding *any* $x \neq x'$ with $H(x) = H(x')$ requires $\Theta(2^{n/2})$ work (birthday bound).

SHA-256 Step by Step

SHA-256 (Secure Hash Algorithm, 256-bit output) is the hash function used throughout Bitcoin. Its processing pipeline is:

1. **Padding**: The input message is padded to a length that is a multiple of 512 bits. Padding appends a '1' bit, then zeros, then a 64-bit encoding of the original message length.
2. **Parsing**: The padded message is divided into N blocks of 512 bits each.
3. **Initialisation**: Eight 32-bit hash values $H_0^{(0)}, \ldots, H_7^{(0)}$ are initialised to the fractional parts of the square roots of the first 8 primes.
4. **Message schedule**: For each 512-bit block, 64 32-bit words $W_0, \ldots, W_{63}$ are derived using XOR and bitwise rotation operations.
5. **Compression**: 64 rounds of mixing using the current hash values, the message schedule words, and 64 round constants (derived from cube roots of primes). Each round applies bitwise operations: Σ_0, Σ_1, Ch, Maj, and modular addition.
6. **Final hash**: After all blocks are processed, the eight 32-bit hash values are concatenated to produce the 256-bit digest.

Bitcoin applies SHA-256 *twice* (SHA-256^2, often written H^2) for block hashing and transaction IDs. The double application mitigates theoretical length-extension attacks.

The Avalanche Effect

A single-bit change in the input produces an entirely different output. This is called the **avalanche effect** or **strict avalanche criterion**. For SHA-256, changing one character typically flips ≈ 128 of the 256 output bits—close to the expected 50% for a well-designed hash function.

Example: Avalanche Demonstration

```
SHA-256(''Hello'') = 185f8db32271fe25f561...
SHA-256(''Hello.'') = 4daa93b5a...  (completely different)
```

Only one character was added; more than half the output bits changed. This means an attacker cannot iteratively *nudge* the hash toward a target—each trial is statistically independent.

The Birthday Paradox and Collision Security

The birthday paradox tells us that collisions become likely after $\approx \sqrt{2^n} = 2^{n/2}$ trials, not 2^n. For SHA-256 ($n = 256$):

- **Collision threshold**: $2^{128} \approx 3.4 \times 10^{38}$ hash evaluations.
- **Time at current hashrate**: Bitcoin's network performs $\approx 9 \times 10^{20}$ SHA-256^2 evaluations per second. Reaching 2^{128} evaluations would take $\dfrac{2^{128}}{9 \times 10^{20}} \approx 10^{17}$ years — millions of times the age of the universe.

Algorithm	Output bits	Collision work	Status
MD5	128	2^{64}	Broken (collisions found 2004)
SHA-1	160	2^{80}	Broken (SHAttered attack 2017)
SHA-256	256	2^{128}	Secure
SHA-512	512	2^{256}	Secure (conservative margin)

Public Key Cryptography

Symmetric vs. Asymmetric Cryptography

- **Symmetric cryptography** (e.g., AES): A single shared secret key is used for both encryption and decryption. The key distribution problem—securely sharing the key before communication begins—is a fundamental limitation.
- **Asymmetric cryptography** (e.g., RSA, ECDSA): A mathematically linked *key pair* is generated. The **public key** can be distributed freely; the **private key** is kept secret. Data encrypted with the public key can only be decrypted with the private key, and vice versa. The key distribution problem disappears.

Blockchain uses asymmetric cryptography *not for encryption* but for **digital signatures**: proving ownership and authorising transactions.

Elliptic Curve Mathematics: Intuition

Bitcoin uses the **secp256k1** elliptic curve, defined over a prime field:

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

where $p = 2^{256} - 2^{32} - 977$ is a 256-bit prime.

Key operations on an elliptic curve:

- **Point addition**: Draw a line through two points P and Q on the curve. The line intersects the curve at a third point; its reflection over the x-axis is $P + Q$.
- **Point doubling**: Draw the tangent to the curve at P; the reflected intersection is $P + P = 2P$.
- **Scalar multiplication**: $kP = P + P + \cdots + P$ (k times). Using the *double-and-add* algorithm, this requires only $O(\log k)$ operations.

The security assumption (Elliptic Curve Discrete Logarithm Problem, ECDLP):

Given P (a known base point G) and $Q = kG$, find k.

No efficient classical algorithm exists for this problem on properly chosen curves. The best known algorithm (Pollard's rho) requires $O(\sqrt{n}) \approx 2^{128}$ operations for secp256k1.

Example: Why ECC over RSA?

Algorithm	Key Size	Security Level	Notes
RSA	1024 bit	≈ 80 bit	Obsolete
RSA	3072 bit	≈ 128 bit	Comparable to ECC-256
ECC-256	256 bit	≈ 128 bit	Bitcoin/Ethereum standard
ECC-384	384 bit	≈ 192 bit	TLS standard

ECC provides equivalent security with 12× smaller keys. On a blockchain where every byte of every transaction is stored by every full node for all time, compact signatures are economically significant.

ECDSA on secp256k1

Key Generation

1. Choose a random integer $d \in [1, n-1]$ where n is the order of the secp256k1 curve group ($n \approx 1.158 \times 10^{77}$). This is the **private key**.
2. Compute $Q = d \times G$ where G is the fixed generator point of secp256k1. This is the **public key** (a point on the curve, represented as 64 bytes in uncompressed form or 33 bytes compressed).

The private key d is a single 256-bit number, typically generated by a cryptographically secure random number generator (CSPRNG).

Signing Process

To sign message m with private key d:

1. Compute the hash of the message: $e = H(m)$ (SHA-256 for Bitcoin transactions).
2. Choose a fresh random integer $k \in [1, n-1]$ (**nonce**).
3. Compute the curve point $(x_1, y_1) = k \times G$.
4. Set $r = x_1 \bmod n$. If $r = 0$, go back to step 2.
5. Compute $s = k^{-1}(e + r \cdot d) \bmod n$. If $s = 0$, go back to step 2.
6. The signature is the pair (r, s).

Common Pitfall

Critical: The nonce k must be freshly generated and secret for each signature. If k is reused across two signatures, an attacker can algebraically recover the private key d from the two (r, s) pairs and the two messages. This vulnerability was exploited in the PlayStation 3 hack (2010)

and has compromised Bitcoin wallets using buggy random number generators.

Verification

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

1. Compute $e = H(m)$.
2. Compute $w = s^{-1} \bmod n$.
3. Compute $u_1 = e \cdot w \bmod n$ and $u_2 = r \cdot w \bmod n$.
4. Compute the curve point $(x_1, y_1) = u_1 \times G + u_2 \times Q$.
5. The signature is valid if and only if $r \equiv x_1 \pmod{n}$.

The verification is correct because:

$$u_1 G + u_2 Q = (ew)G + (rw)(dG) = w(e + rd)G = \frac{e + rd}{s} G = \frac{e + rd}{k^{-1}(e + rd)} G = kG$$

whose x-coordinate is r by construction.

Digital Signatures

Digital Signature

A **digital signature** is a cryptographic scheme for demonstrating the authenticity and integrity of a digital message. It provides:

- **Authentication**: The signature could only have been produced by the holder of the corresponding private key.
- **Non-repudiation**: The signer cannot later deny having signed (assuming the private key was not compromised).
- **Integrity**: Any modification to the message after signing invalidates the signature.

Use in Bitcoin Transactions

Every Bitcoin transaction input must include a valid ECDSA signature over the transaction data. The full process:

1. **Alice constructs** the transaction: specifying which UTXOs to spend (inputs), the recipient address and amount (outputs), and the transaction fee.
2. **Alice hashes** the serialised transaction data using SHA-256[2].
3. **Alice signs** the hash with her private key using ECDSA, producing (r, s).
4. **Alice broadcasts**: the transaction data together with the signature (r, s) and her public key.

5. **Nodes verify**: using Alice's public key, they check that (r, s) is a valid ECDSA signature over the transaction hash.
6. **Acceptance**: If valid, nodes relay the transaction; if invalid (wrong signature or spent inputs), they drop it.

The critical property: Alice *proves ownership* of the UTXOs without ever revealing her private key. The private key never leaves Alice's device.

Address Generation

The derivation pipeline from private key to Bitcoin address consists of four steps, each adding a layer of protection:

Step	Operation
1. Private key	256-bit random integer d
2. Public key	$Q = d \times G$ on secp256k1. Compressed form: 33 bytes (prefix 02 or 03 + 32-byte x-coordinate).
3. Hash	Apply SHA-256 to compressed public key, then RIPEMD-160: produces a 20-byte (160-bit) hash. This "Hash160" is the *public key hash (PKH)*.
4. Address	Prepend version byte (`00` for mainnet P2PKH), append 4-byte checksum (first 4 bytes of SHA-256^2 of versioned hash), encode in Base58Check. Result: a 26–34-character string starting with "`1`".

Why the double hashing in step 3? SHA-256 followed by RIPEMD-160 (sometimes called HASH160) reduces the output length from 256 to 160 bits, making addresses shorter. It also provides defence-in-depth: breaking both functions would be required to derive a private key from an address.

Modern address formats:

- **SegWit (P2WPKH, "bc1q")**: Uses Bech32 encoding, lower fees due to SegWit discount, better error detection.
- **Taproot (P2TR, "bc1p")**: Uses Bech32m encoding, Schnorr signatures, enhanced privacy (all spend conditions look identical on-chain when using key-path spend).

Common Pitfall

Addresses are not public keys. An address is a *hash* of a public key. An attacker who compromises SHA-256 and RIPEMD-160 would still need to solve the ECDLP to obtain the private key. However, once an address has been used to *send* (not just receive), the public key is revealed in the spending transaction's input script, reducing this protection to just the ECDLP.

Merkle Trees and SPV

Detailed Proof Mechanism

A Merkle proof of inclusion for transaction T_i in a block with n transactions consists of:

- The hash of T_i: $h_i = H(T_i)$.
- The **authentication path**: a sequence of sibling hashes $h_{s_1}, h_{s_2}, \ldots, h_{s_k}$ at each level of the tree from h_i up to the root, where $k = \lceil \log_2 n \rceil$.
- The **position bits**: a bit string indicating whether each sibling is left or right at each level.

To verify: starting from h_i, combine with h_{s_1} (in the correct left/right order) and hash; repeat up the tree until the root is computed. If the computed root matches the Merkle root in the block header, the transaction is proven to be in the block.

Efficiency Analysis

Transactions n	**Proof hashes**	**Proof size (bytes)**	**Full block hashes**
$2^4 = 16$	4	128	16
$2^{10} = 1024$	10	320	1024
$2^{13} \approx 8192$	13	416	8192
$2^{20} \approx 10^6$	20	640	10^6

The $O(\log n)$ scaling means a proof for a transaction in a block with one million other transactions requires only 640 bytes, compared to $32 \times 10^6 = 32\,\text{MB}$ to download all transaction hashes. This is what makes mobile SPV wallets practical.

SPV Security Trade-offs

SPV provides **transaction inclusion proofs** but does not verify:

- Whether the inputs being spent are actually unspent (requires UTXO set knowledge).
- Whether the block itself follows all consensus rules (requires full node validation).

An SPV client trusts that the longest chain is valid, relying on the economic disincentive for miners to build on invalid blocks. Full nodes provide stronger but more resource-intensive verification.

BIP-39 and HD Wallets

BIP-39: Mnemonic Seed Generation

BIP-39 Mnemonic Phrase

BIP-39 is a standard for encoding entropy as a sequence of human-readable words, making wallet backups reliable and resistant to transcription errors.

1. Generate E bits of cryptographically secure entropy ($E \in \{128, 160, 192, 224, 256\}$ bits, corresponding to 12–24 words).
2. Compute checksum: append the first $E/32$ bits of SHA-256(entropy) to the entropy.
3. Split the combined bit string into groups of 11 bits.
4. Map each 11-bit group to one word from the BIP-39 wordlist (2048 words; $2^{11} = 2048$).
5. Optionally add a passphrase ("25th word") for extra security.
6. Stretch the mnemonic + passphrase using PBKDF2 (2048 rounds of HMAC-SHA512) to produce a 512-bit seed.

The 11-bit checksum in the final word(s) allows wallets to detect transcription errors: a mistyped word will almost certainly fail the checksum. This is why BIP-39 wallets say "invalid mnemonic" when a word is wrong.

BIP-32 and BIP-44: Hierarchical Deterministic Wallets

BIP-32 (HD Wallets) defines a way to derive a tree of key pairs from a single 512-bit seed, using HMAC-SHA512 as the derivation function:

$$\text{child}_i = \text{CKD}(\text{parent key},\ i)$$

where CKD is the Child Key Derivation function. This allows:

- An unlimited number of private keys from a single backup.
- **Hardened derivation** (index $\geq 2^{31}$): child key cannot be derived from the parent public key; used for account-level keys.
- **Normal derivation** (index $< 2^{31}$): child public key derivable from parent public key; allows watch-only wallets.

BIP-44 standardises the derivation path structure:

$$m \,/\, \text{purpose}' \,/\, \text{coin\_type}' \,/\, \text{account}' \,/\, \text{change} \,/\, \text{address\_index}$$

Example: BIP-44 Derivation Path for Bitcoin

The path `m/44'/0'/0'/0/5` means:

- `44'`: BIP-44 purpose (hardened)

- `0'`: Bitcoin mainnet (coin type 0, hardened)
- `0'`: First account (hardened)
- `0`: External chain (receiving addresses)
- `5`: Sixth address in this account

The same seed with `m/44'/60'/0'/0/5` would produce the sixth Ethereum receiving address (coin type 60).

Post-Quantum Cryptography

The Quantum Threat to ECDSA

Classical computers require $O(2^{128})$ operations to break secp256k1 via Pollard's rho algorithm—computationally infeasible. However, **Shor's algorithm** running on a sufficiently large quantum computer can solve the ECDLP in $O(n^3)$ polynomial time, where n is the number of qubits encoding the key size.

Algorithms threatened by Shor's algorithm:

- ECDSA (Bitcoin and Ethereum signatures)
- RSA encryption
- Diffie-Hellman key exchange

SHA-256 hash functions are threatened only by **Grover's algorithm**, which reduces the effective security from 2^n to $2^{n/2}$. For SHA-256 this means 2^{128} quantum operations—still practically infeasible. Doubling hash output length (SHA-512) provides the same classical security margin post-quantum.

NIST Post-Quantum Standards (2024)

NIST completed its post-quantum standardisation process in 2024, finalising three algorithms:

Algorithm	Type	Basis	Blockchain relevance
CRYSTALS-Dilithium	Signature	Lattice (Module-LWE)	Leading candidate to replace ECDSA
CRYSTALS-Kyber	KEM	Lattice (Module-LWE)	Key encapsulation (not signatures)
SPHINCS+	Signature	Hash-based	Conservative; larger signatures (∼8KB)

Trade-offs of post-quantum signatures:

- ECDSA signature: 64 bytes.

- Dilithium signature: ≈ 2420 bytes ($\sim 38\times$ larger).
- SPHINCS+ signature: ≈ 8000 bytes ($\sim 125\times$ larger).

Larger signatures mean higher on-chain storage costs and lower effective throughput. Layer-2 solutions and batched signature schemes are being researched to mitigate this.

Timeline and "Harvest Now, Decrypt Later"

Expert estimates for cryptographically relevant quantum computers (capable of breaking 256-bit ECDSA) range from 2030 to 2040, though significant uncertainty exists. The **"harvest now, decrypt later"** threat model is already relevant: adversaries recording today's blockchain transactions could decrypt exposed public keys in the future. Any address that has *broadcast a spending transaction* (thereby revealing its public key) is theoretically vulnerable once quantum computers are available. Unspent outputs at fresh addresses (where only the address hash is known) retain additional protection.

Common Pitfall

Reused addresses increase quantum risk: If an address has never sent a transaction, only its HASH160 is known—not the public key. An attacker would need to break both SHA-256 and RIPEMD-160 before applying Shor's algorithm. Wallets that generate a new address for every transaction are therefore more quantum-resilient under current usage.

Practice Problems

1. **Hash Properties.** Explain why *collision resistance* is weaker than *pre-image resistance*. Which does Bitcoin's mining puzzle require, and why?

   Solution: Pre-image resistance requires finding *any* x for a given $y = H(x)$. Collision resistance requires finding *any* pair $x \neq x'$ with $H(x) = H(x')$—no specific target is required. Collision search benefits from the birthday paradox, reducing effective work from 2^n to $2^{n/2}$. Bitcoin's mining puzzle requires *pre-image resistance with a partial constraint*: finding a nonce such that $H(\text{header}) < \text{target}$. This is closer to a partial pre-image search, which cannot use the birthday speedup because miners must target a specific hash range, not just any collision.

2. **ECDSA Nonce Reuse Attack.** Alice signs two messages m_1 and m_2 using the same nonce k. Both signatures produce the same $r = (kG)_x \bmod n$. Show algebraically how an attacker can recover Alice's private key d from (r, s_1, m_1) and (r, s_2, m_2).

   Solution: From the signing equation: $s_1 = k^{-1}(e_1 + rd) \bmod n$ and $s_2 = k^{-1}(e_2 + rd) \bmod n$. Subtract: $s_1 - s_2 = k^{-1}(e_1 - e_2)$, so $k = (e_1 - e_2)(s_1 - s_2)^{-1} \bmod n$. Then from $s_1 = k^{-1}(e_1 + rd)$: $d = (s_1 k - e_1) r^{-1} \bmod n$. Both e_1 and e_2 are computable from public messages, and r, s_1, s_2 are in the signatures. All of Alice's future and past transactions signed with that private key are now compromised.

3. **Address Derivation Pipeline.** List the steps and algorithms required to go from a randomly

generated 256-bit number to a Bitcoin Legacy (P2PKH) address. For each step, state what property the step adds (e.g., "reduces linkability", "adds error detection").

Solution:

(a) Random 256-bit integer d (private key): entropy source.

(b) $Q = d \times G$ on secp256k1 (public key): enables asymmetric cryptography; one-way via ECDLP.

(c) $H_1 = \text{SHA-256}(Q_{\text{compressed}})$: produces fixed 32-byte value; collision resistant.

(d) $H_2 = \text{RIPEMD-160}(H_1)$: shortens to 20 bytes; defence-in-depth (requires breaking two hash functions to link address to public key).

(e) Prepend version byte `0x00`: identifies network.

(f) Checksum $= \text{SHA-256}^2(\texttt{0x00} \| H_2)[0:4]$: error detection.

(g) Base58Check encode: removes visually ambiguous characters (0, O, I, l); human-readable.

4. **Post-Quantum Threat Prioritisation.** A Bitcoin developer argues: "We don't need to worry about quantum computers for Bitcoin because SHA-256 is quantum-safe." Explain what is correct and what is incomplete about this statement.

   Solution: The statement is *partially correct*: SHA-256 is indeed resistant to Grover's algorithm (reduces security from 2^{256} to 2^{128}, which remains infeasible). Block hashing and Merkle trees are therefore not significantly threatened. However, Bitcoin's *signature scheme* (ECDSA on secp256k1) *is* threatened by Shor's algorithm, which would allow an attacker to derive private keys from public keys in polynomial time. Since public keys are exposed in spending transactions (and are derivable from addresses in some circumstances), ECDSA is the primary quantum vulnerability in Bitcoin, not SHA-256.

Key Takeaways

- The five properties of cryptographic hash functions are: deterministic, efficient, pre-image resistant, second pre-image resistant, and collision resistant. All five are required for blockchain security.
- SHA-256 applies 64 rounds of bitwise mixing per 512-bit input block. Its birthday-bound collision resistance is 2^{128}, effectively infeasible to attack classically.
- Elliptic curve cryptography (secp256k1: $y^2 = x^3 + 7 \bmod p$) provides 128-bit security with only 256-bit keys, via the hardness of the ECDLP.
- ECDSA signing uses a secret nonce k; reusing k for two signatures immediately leaks the private key.
- Bitcoin address generation: private key $\xrightarrow{\times G}$ public key $\xrightarrow{\text{SHA-256 + RIPEMD-160}}$ public key hash $\xrightarrow{\text{Base58Check}}$ address. Each step is one-way; recovering the private key from an address requires

breaking both hash functions and ECDLP.

- BIP-39 encodes entropy as 12–24 words using a 2048-word list. BIP-32/44 derives an entire tree of key pairs from one seed via hierarchical deterministic derivation.
- Shor's algorithm threatens ECDSA (but not SHA-256). NIST standardised post-quantum replacements in 2024: CRYSTALS-Dilithium for signatures, CRYSTALS-Kyber for key encapsulation, SPHINCS+ as a conservative hash-based fallback. Migration planning is underway in the blockchain community.

Further Reading

- Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*, Section 4 (Merkle trees). https://bitcoin.org/bitcoin.pdf
- Antonopoulos, A. M. (2017). *Mastering Bitcoin* (2nd ed.), Chapters 4–5 (Keys and Addresses, Transactions). https://github.com/bitcoinbook/bitcoinbook
- Johnson, D., Menezes, A., & Vanstone, S. (2001). The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, 1(1), 36–63.
- NIST (2022). *Post-Quantum Cryptography: Selected Algorithms 2022*. NIST FIPS 203/204/205 (2024). https://csrc.nist.gov/projects/post-quantum-cryptography
- Bitcoin Improvement Proposals: BIP-39 (Mnemonic), BIP-32 (HD Wallets), BIP-44 (Derivation paths).
- Narayanan, A. et al. (2016). *Bitcoin and Cryptocurrency Technologies*, Chapter 1 (Cryptographic Primitives). https://bitcoinbook.cs.princeton.edu/
- Merkle, R. (1980). Protocols for public key cryptosystems. *Proceedings of the IEEE Symposium on Security and Privacy*. The original paper on hash trees.