Cryptography & Security: A Quantitative Deep Dive

Standalone Technical Lecture

Prof. Dr. Joerg Osterrieder

University Lecture Series

March 5, 2026

History of Cryptography Timeline

Classical Era

- **700 BC**: Spartan scytale – first transposition cipher
- **50 BC**: Caesar cipher – substitution with shift key
- **1940s**: Enigma broken by Turing et al.
- **1977**: DES standardized (56-bit key)

Modern Era

- **1978**: RSA – first practical public-key system
- **2001**: AES replaces DES (128/192/256-bit keys)
- **2009**: Bitcoin uses ECDSA on secp256k1
- **2024**: NIST post-quantum standards finalized

3000

years of cryptography: from physical devices to mathematical proofs of security

Kerckhoffs's Principle

Security through Obscurity

- Secret algorithm
- Reverse-engineerable
- No public audit

✗

Kerckhoffs's Principle

- Public algorithm
- Security in the key
- Peer-reviewed

✓

Formal Statement

A cryptosystem should be secure even if everything about the system, except the **key**, is public knowledge.

Implications for Blockchain

- Bitcoin Core is **open source** – anyone can audit
- Security rests on **private key secrecy**, not code secrecy
- Thousands of researchers verify cryptographic assumptions
- Obscure protocols often hide vulnerabilities

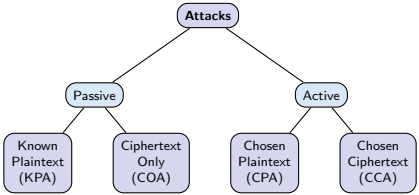
Key Insight

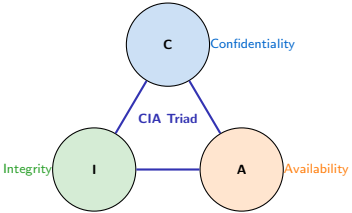
Open algorithms with secret keys $\gg$ secret algorithms with no review.

Augu

Kerckhoffs (1883): "The system must not require secrecy and can be stolen by the enemy without causing trouble."

Threat Model & Attack Taxonomy





Attack Hierarchy (Strength)

- COA $\subset$ KPA $\subset$ CPA $\subset$ CCA
- Scheme secure under CCA $\Rightarrow$ secure under all weaker models
- Modern goal: **IND-CCA2** (indistinguishability under adaptive CCA)

CIA in Blockchain Context

- **C**: Private keys, transaction privacy (zk-proofs)
- **I**: Hash chains, Merkle proofs, digital signatures
- **A**: P2P network, no single point of failure

A

well-defined threat model is prerequisite for any formal security analysis

Course Roadmap

8 Sections

1. **Introduction & Motivation** ✓
2. **Hash Functions Deep Dive** – SHA-256, SHA-3, birthday bound
3. **Symmetric Cryptography** – AES, modes, DH key exchange
4. **Asymmetric Cryptography** – RSA, ECC, ECDSA
5. **Digital Signatures** – Schnorr, multisig, BLS
6. **Key Management** – HD wallets, BIP-32/39/44
7. **Zero-Knowledge Proofs** – zk-SNARKs, zk-STARKs
8. **Post-Quantum Crypto** – lattices, hash-based sigs

Learning Objectives

- Master hash function internals and security proofs
- Understand elliptic curve mathematics for ECDSA
- Analyze key derivation and wallet security
- Evaluate zero-knowledge proof systems
- Assess post-quantum migration strategies

Central Question

How do cryptographic primitives enable trustless digital ownership and verifiable computation?

Prere

modular arithmetic, basic group theory, probability

Learning Objectives

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

1. **Explain** the mathematical properties of cryptographic hash functions (preimage, collision, avalanche)
2. **Compare** symmetric and asymmetric encryption schemes and their performance trade-offs
3. **Analyze** ECDSA signing and verification using elliptic curve arithmetic
4. **Evaluate** wallet security architectures (HD wallets, BIP-32/39/44 derivation paths)
5. **Assess** quantum computing threats and post-quantum cryptography migration strategies

Bloom

taxonomy levels: Remember → Understand → Apply → Analyze → Evaluate → Create

Hash Function Formal Properties

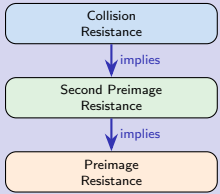
Definition

A cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^n$ maps arbitrary-length input to a fixed n-bit digest.

Three Security Properties

1. **Preimage Resistance:** Given y, it is computationally infeasible to find x s.t. $H(x) = y$.
   Complexity: $\mathcal{O}(2^n)$
2. **Second Preimage Resistance:** Given x_1, infeasible to find $x_2 \neq x_1$ s.t. $H(x_1) = H(x_2)$.
   Complexity: $\mathcal{O}(2^n)$
3. **Collision Resistance:** Infeasible to find *any* $x_1 \neq x_2$ with $H(x_1) = H(x_2)$.
   Complexity: $\mathcal{O}(2^{n/2})$ (birthday bound)

Property Hierarchy

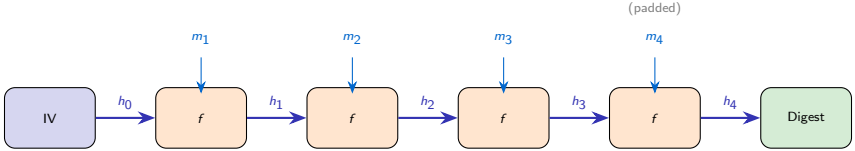


Note

Collision resistance is the *strongest* property. A collision-resistant hash is automatically second-preimage and preimage resistant.

SHA-256 ($n = 256$): preimage attack requires $\sim 2^{256}$ operations – far beyond any computational budget

Merkle–Damgård Construction



Algorithm

1. Pad message M to multiple of block size (512 bits for SHA-256)
2. Split into blocks $m_1, m_2, \ldots, m_L$
3. Initialize $h_0 = \text{IV}$ (fixed constants)
4. Iterate: $h_i = f(h_{i-1}, m_i)$ for $i = 1, \ldots, L$
5. Output: $H(M) = h_L$

Security Reduction

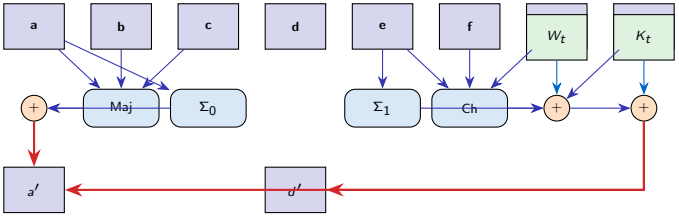
If f is a collision-resistant compression function, then the full Merkle–Damgård hash H is also collision-resistant.

Length Extension Weakness

Given $H(M)$ and $|M|$, one can compute $H(M \| \text{pad} \| M')$ **without knowing** M. This is why HMAC or SHA-3 is preferred for MACs.

SHA-256, SHA-512, MD5, and RIPEMD-160 all use Merkle–Damgård; SHA-3 uses the sponge construction instead

SHA-256 Internals: Compression Round



Round Operations (64 rounds)

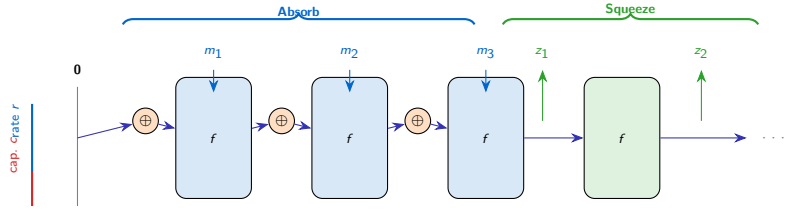
- $\mathrm{Ch}(e, f, g) = (e \wedge f) \oplus (\neg e \wedge g)$
- $\mathrm{Maj}(a, b, c) = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c)$
- $\Sigma_0(a) = \mathrm{ROTR}^2(a) \oplus \mathrm{ROTR}^{13}(a) \oplus \mathrm{ROTR}^{22}(a)$
- $\Sigma_1(e) = \mathrm{ROTR}^6(e) \oplus \mathrm{ROTR}^{11}(e) \oplus \mathrm{ROTR}^{25}(e)$

Parameters

- Block size: 512 bits (16 words of 32 bits)
- Digest: 256 bits (8 words)
- Rounds: 64 per block
- K_t: 64 round constants (fractional parts of cube roots of first 64 primes)
- W_t: message schedule (expanded from 16 to 64 words)

SHA-256 processes each 512-bit block through 64 rounds of mixing, shifting, and modular addition

Sponge Construction (Keccak / SHA-3)



Sponge Parameters

- State width: $b = r + c = 1600$ bits
- Rate r: bits absorbed/squeezed per call to f
- Capacity c: security parameter ($c = 2\times$ security level)
- SHA3-256: $r = 1088$, $c = 512$, security $= 128$ bits

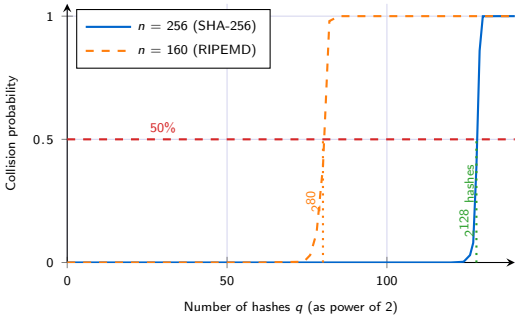
Advantages over Merkle–Damgård

- **No length extension attacks** by design
- Variable output length (SHAKE128, SHAKE256)
- Simpler security proof: random permutation model
- Algebraically distinct from SHA-2 (defense in depth)

Kecca

won the NIST SHA-3 competition (2012) among 64 submissions; standardized as FIPS 202

Birthday Bound & Collision Probability



Birthday Paradox

For $N = 2^n$ possible outputs and q queries:

$$P(\text{collision}) \approx 1 - e^{-q^2/(2N)}$$

At $q \approx \sqrt{N} = 2^{n/2}$, probability $\approx 50\%$.

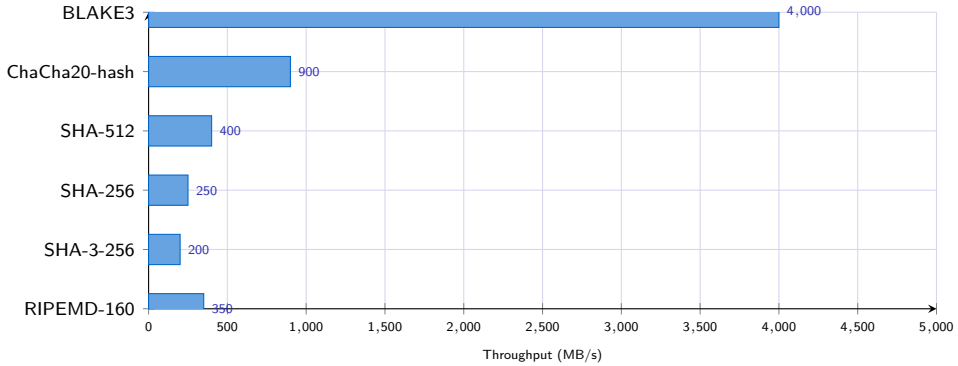
Practical Implications

- SHA-256: collision at $\sim 2^{128}$ operations
- RIPEMD-160: collision at $\sim 2^{80}$ operations
- MD5 ($n = 128$): collision at 2^{64} – **broken in practice**
- SHA-1 ($n = 160$): first collision found 2017 (SHAttered)

Bitco

uses double SHA-256 and RIPEMD-160; the birthday bound sets the effective security margin

Hash Function Throughput Comparison



Key Observations

- **BLAKE3**: tree-hashing parallelism $\rightarrow \sim 16\times$ SHA-256
- **SHA-512**: faster than SHA-256 on 64-bit CPUs
- **SHA-3**: secure but slower in software (fast in hardware)

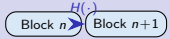
Blockchain Usage

- Bitcoin: SHA-256d (double hash)
- Ethereum: Keccak-256 (pre-NIST SHA-3 variant)
- Zcash: BLAKE2b for Equihash PoW
- Addresses: RIPEMD-160(SHA-256(pubkey))

Bench

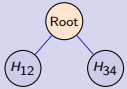
Hash Applications in Blockchain

Block Linking



prevHash field

Merkle Root



$O(\log n)$ proofs

Proof of Work

Find nonce s.t.
$H(\text{header} \| \text{nonce})$
$<$ target

Preimage hardness

Address Derivation

pubkey
↓ SHA-256
↓ RIPEMD-160
↓ Base58Check

One-way binding

Why Hash Functions Are Fundamental

- **Integrity**: any change invalidates the chain
- **Commitment**: miners commit to block content
- **Efficiency**: Merkle proofs enable light clients

Hash Count per Bitcoin Block

- Block header hash: $2\times$ SHA-256
- Merkle tree: $\sim 2N - 1$ hashes for N txs
- Addresses: SHA-256 + RIPEMD-160 each
- PoW: billions of trial hashes per second

Hash functions are the most-used cryptographic primitive in any blockchain system

Symmetric Encryption: Formal Definitions

Definition

A symmetric encryption scheme is a triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ over key space $\mathbb{K}$, message space $\mathbb{M}$, ciphertext space $\mathbb{C}$:

$$\mathcal{E}_k : \mathbb{M} \to \mathbb{C}, \qquad \mathcal{D}_k : \mathbb{C} \to \mathbb{M}, \qquad \forall\, m \in \mathbb{M},\ k \in \mathbb{K} :\ \mathcal{D}_k(\mathcal{E}_k(m)) = m$$

Security Notion: IND-CPA

An adversary $\mathcal{A}$ with access to encryption oracle $\mathcal{E}_k(\cdot)$ picks (m_0, m_1), receives $c^* = \mathcal{E}_k(m_b)$ for random b, and guesses b. The scheme is IND-CPA secure if:

$$\left|\Pr[\mathcal{A} \text{ wins}] - \tfrac{1}{2}\right| \leq \mathrm{negl}(\lambda)$$

Key Properties

- **Same key** for encryption and decryption
- Key must be shared via **secure channel**
- For n parties: $\binom{n}{2} = \frac{n(n-1)}{2}$ keys needed
- Very fast: hardware AES-NI $\sim$ 1 GB/s

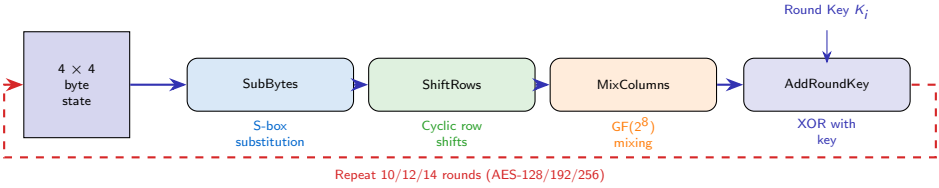
Blockchain Relevance

- Encrypted wallet files (AES-256-CBC)
- Secure P2P channels (AES-GCM in TLS)
- BIP-38 passphrase-protected private keys

Symm

crypto is $\sim 1000\times$ faster than asymmetric – used for bulk data, not authentication

AES Round Structure



Design Principles

- **Confusion**: SubBytes (non-linear S-box in GF(2^8))
- **Diffusion**: ShiftRows + MixColumns spread influence
- **Key mixing**: AddRoundKey via XOR
- Full diffusion after 2 rounds; 10+ for security margin

AES Variants

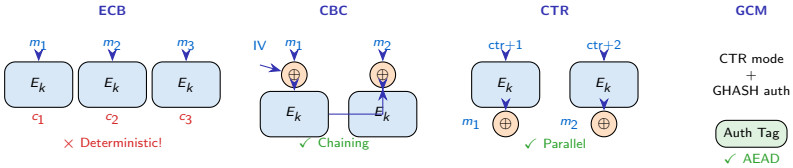
	128	**192**	**256**
Key bits	128	192	256
Rounds	10	12	14
Security	128	192	256

Best known attack: biclique – reduces AES-256 to $2^{254.4}$ (still impractical).

AES

(Rijndael) standardized by NIST in 2001; ubiquitous in TLS, disk encryption, and wallet protection

Block Cipher Modes of Operation



Mode Comparison

	ECB	CBC	CTR	GCM
IND-CPA	×	✓	✓	✓
Parallel enc	✓	×	✓	✓
Auth (AEAD)	×	×	×	✓

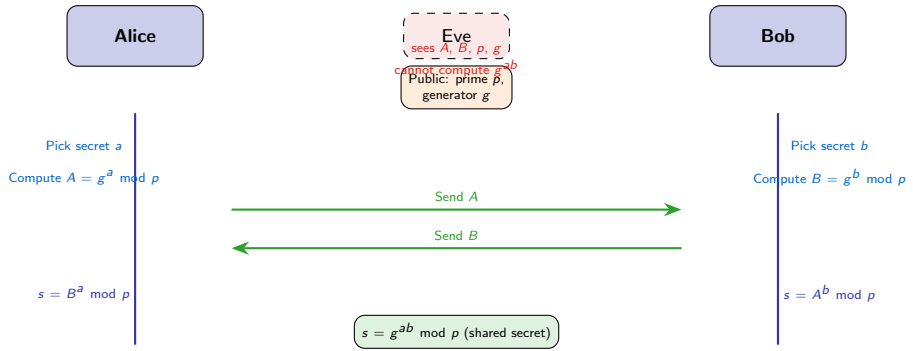
Blockchain Usage

- **AES-256-CBC**: Bitcoin Core wallet encryption
- **AES-GCM**: TLS 1.3 for P2P node communication
- **CTR mode**: Preferred for parallelism in hardware wallets
- Never use ECB for any security-sensitive application

GCM

(Galois/Counter Mode) provides both confidentiality and integrity – the modern standard for AEAD

Diffie–Hellman Key Exchange



Security Basis

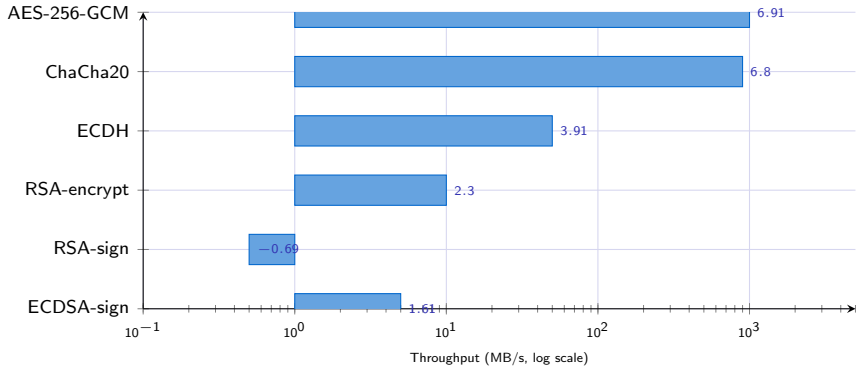
Decisional Diffie–Hellman (DDH):
Given (g, g^a, g^b), it is hard to distinguish g^{ab} from g^c for random c.

Limitations

- Vulnerable to **man-in-the-middle** without authentication
- No identity verification – must combine with signatures
- Basis for ECDH (Elliptic Curve DH) used in modern TLS

Diffie & Hellman (1976): first published public-key protocol; foundation of modern key exchange

Symmetric vs. Asymmetric: Performance Trade-offs



Symmetric Advantages

- 100–1000× faster than asymmetric
- Constant-time implementations easier
- Hardware acceleration (AES-NI, ARM CE)
- Used for bulk encryption after key exchange

Asymmetric Advantages

- No pre-shared secret required
- Digital signatures for authentication
- Key distribution scales: n keys for n parties
- Non-repudiation (signatures prove origin)

Hybri

RSA: Mathematical Foundations

Key Generation

1. Choose two large primes p, q (each ≥ 1024 bits)
2. Compute $n = p \cdot q$ and $\phi(n) = (p-1)(q-1)$
3. Choose e with $\gcd(e, \phi(n)) = 1$ (common: $e = 65537$)
4. Compute $d \equiv e^{-1} \pmod{\phi(n)}$ (extended Euclidean algorithm)
5. Public key: (n, e) Private key: (n, d)

Encryption / Decryption

$$\text{Encrypt:} \quad c = m^e \bmod n$$

$$\text{Decrypt:} \quad m = c^d \bmod n$$

Correctness: By Euler's theorem,

$$c^d = m^{ed} = m^{1+k\phi(n)} = m \cdot (m^{\phi(n)})^k \equiv m \pmod{n}$$

Security Assumptions

- **RSA problem**: Given $c = m^e \bmod n$, find m
- **Factoring**: Given $n = pq$, find p, q
- Best known: General Number Field Sieve

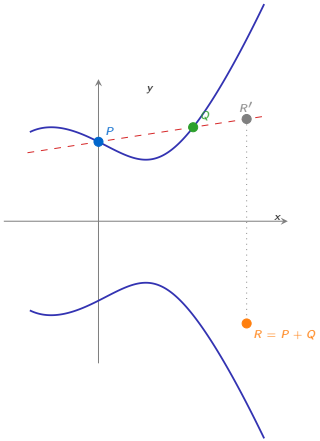
$$L_n\left[\tfrac{1}{3}, 1.923\right] = e^{1.923(\ln n)^{1/3}(\ln \ln n)^{2/3}}$$

- RSA-2048: $\sim 2^{112}$ security (sub-exponential)

Why Blockchain Doesn't Use RSA

RSA-2048 keys are 256 bytes vs. ECDSA-256 keys at 32 bytes. RSA signatures are $\sim$50$\times$ larger and $\sim$10$\times$ slower. ECC provides

Elliptic Curve Fundamentals



Weierstrass Form

$$y^2 = x^3 + ax + b, \qquad 4a^3 + 27b^2 \neq 0 \text{ (non-singular)}$$

Point Addition (Geometric)

1. Draw line through P and Q
2. Find third intersection R' with the curve
3. Reflect R' across x-axis: $R = P + Q$

Algebraic Formulas

For $P = (x_1, y_1)$, $Q = (x_2, y_2)$, $P \neq Q$:

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1},$$
$$x_3 = \lambda^2 - x_1 - x_2,$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

Point at infinity $\mathcal{O}$ serves as identity.

Ellipt

curves over finite fields form an abelian group – the foundation of ECC

ECDSA Key Generation & secp256k1 Parameters

secp256k1 Parameters (Bitcoin)

Parameter	Value
Curve	$y^2 = x^3 + 7$ over $\mathbb{F}_p$
p	$2^{256} - 2^{32} - 977$
a	0
b	7
G (generator)	Specified 256-bit (x, y) point
n (order)	$\approx 2^{256}$ (exact 256-bit prime)
h (cofactor)	1

Why secp256k1?

Koblitz curve: no suspicious constants ("nothing up my sleeve"), efficient endomorphism for $\sim 30\%$ speedup.

Key Generation Algorithm

1. Select random $d \in [1, n-1]$
   (private key: 256-bit integer)
2. Compute $Q = d \cdot G$
   (public key: point on curve)
3. Verify $Q \neq \mathcal{O}$ and Q is on the curve

Security Guarantee

Given $Q = dG$, finding d requires solving the **Elliptic Curve Discrete Logarithm Problem (ECDLP)**.

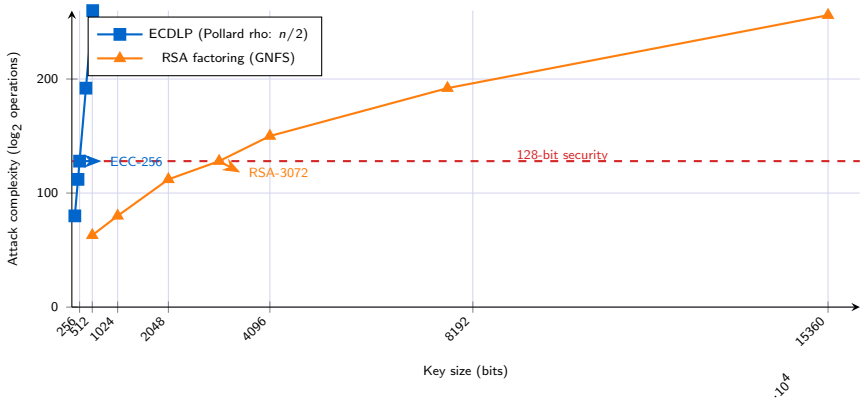
Best known attack: Pollard's rho
Complexity: $\mathcal{O}(\sqrt{n}) \approx 2^{128}$ operations

Key sizes for 128-bit security:
• RSA: 3072 bits • ECC: 256 bits

ECDS

private key = 32 bytes; public key = 33 bytes (compressed) – compact enough for every transaction

Attack Complexity: RSA vs. ECDLP



ECDLP Hardness

- Best generic attack: $\mathcal{O}(\sqrt{n})$ (Pollard rho)
- No sub-exponential algorithm known
- 256-bit key $\rightarrow 2^{128}$ security

RSA Factoring

- GNFS: sub-exponential $L[1/3, 1.923]$
- 3072-bit key $\rightarrow 2^{128}$ security
- $12\times$ larger key for same security level

ECC

Modern Elliptic Curve Designs (Bernstein et al.)

Two complementary curves from the same mathematical family, optimized for different uses.

Property	Ed25519 (EdDSA)	Curve25519 (X25519)
Primary use	Digital signatures	Key exchange (ECDH)
Curve form	Twisted Edwards: $-x^2 + y^2 = 1 + dx^2y^2$	Montgomery: $y^2 = x^3 + 486662x^2 + x$
Field prime	$p = 2^{255} - 19$	$p = 2^{255} - 19$
Security level	$\sim$ 128 bits	$\sim$ 128 bits
Signature size	64 bytes	–
Public key size	32 bytes	32 bytes
Deterministic nonce	Yes (RFC 8032)	–
Constant-time	By design	By design
Side-channel safe	Yes	Yes

Advantages over secp256k1

- **Deterministic**: no random nonce $\rightarrow$ no nonce-reuse vulnerability
- **Faster**: $\sim 3\times$ signature verification speed
- **Complete formulas**: no edge cases in point addition

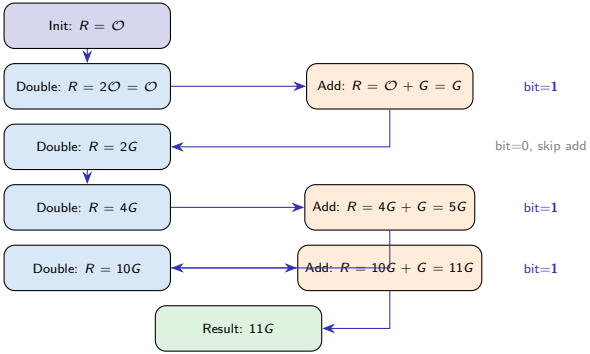
Adoption

- Solana, Cardano, Polkadot: Ed25519
- Signal Protocol, WireGuard: X25519/Ed25519
- SSH, TLS 1.3: Ed25519 supported
- Bitcoin: still secp256k1 (network effect)

Ed25

EC Point Multiplication: Double-and-Add

$d = 11 = (1011)_2$



Algorithm

Input: scalar $d = (d_{l-1} \ldots d_1 d_0)_2$, point G
Output: $Q = dG$

1. $R \leftarrow \mathcal{O}$ (point at infinity)
2. For $i = l - 1$ down to 0:
   1. $R \leftarrow 2R$ (point doubling)
   2. If $d_i = 1$: $R \leftarrow R + G$ (point addition)
3. Return R

Complexity

- l doublings + HW(d) additions
- For 256-bit d: ~ 256 doublings + ~ 128 additions
- Total: $\mathcal{O}(\log d)$ group operations
- Constant-time variant: Montgomery ladder

Side-Channel Warning

Naive double-and-add leaks the scalar d via timing. Always use constant-time implementations.

Doub

Why Blockchain Chose Elliptic Curve Cryptography

Comparison Summary

Property	RSA-3072	ECDSA-256	Ed25519
Security level	128 bits	128 bits	$\sim$ 128 bits
Private key size	3072 bits (384 B)	256 bits (32 B)	256 bits (32 B)
Public key size	3072 bits (384 B)	257 bits (33 B)	256 bits (32 B)
Signature size	3072 bits (384 B)	512 bits (64 B)	512 bits (64 B)
Sign speed	$\sim$ 0.5 MB/s	$\sim$ 5 MB/s	$\sim$ 15 MB/s
Verify speed	$\sim$ 10 MB/s	$\sim$ 3 MB/s	$\sim$ 9 MB/s
Quantum threat	Shor's (factor)	Shor's (ECDLP)	Shor's (ECDLP)

Blockchain Adoption

Network	Signature Scheme
Bitcoin	ECDSA (secp256k1) + Schnorr (Taproot)
Ethereum	ECDSA (secp256k1)
Solana	Ed25519
Cardano	Ed25519
Polkadot	Ed25519 + Sr25519 (Schnorr)
Monero	Ed25519 (ring signatures)

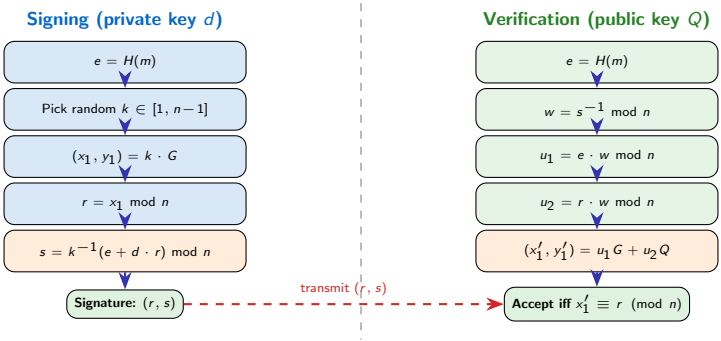
Why ECC Won

- **Size**: 32 B key vs. 384 B $\rightarrow$ smaller transactions
- **Speed**: 10$\times$ faster signing
- **Bandwidth**: millions of txs/day $\times$ savings
- **Storage**: every byte on-chain is permanent
- **Proven**: 25+ years of cryptanalysis

ECC'

compact keys and fast operations make it the ideal choice for bandwidth-constrained blockchain networks

ECDSA: Signing & Verification Formalized



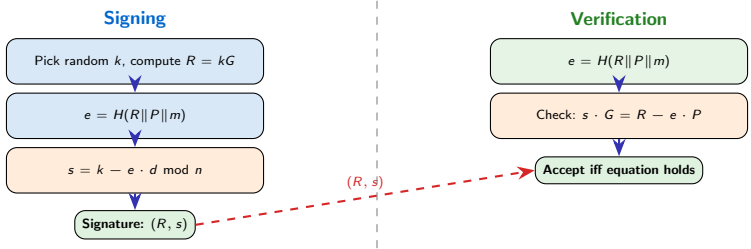
Correctness

$u_1 G + u_2 Q = u_1 G + u_2 dG = (u_1 + u_2 d)G = s^{-1}(e + dr)G = kG$, so x-coordinate matches r.

ECDS

(ANSI X9.62, 1998): the digital signature standard used in Bitcoin since block 0

Schnorr Signatures



Key Property: Linearity

Schnorr signatures are **linear**: given signatures (R_1, s_1) and (R_2, s_2), one can combine them:

$$R = R_1 + R_2, \quad s = s_1 + s_2$$

This enables native signature aggregation (MuSig).

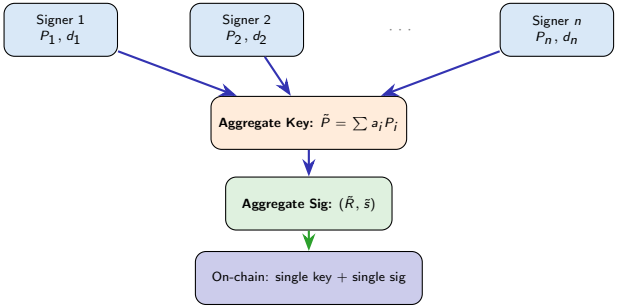
Advantages over ECDSA

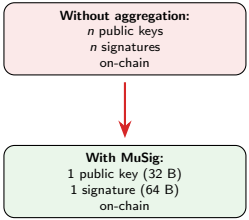
- **Provably secure** under random oracle model
- **Non-malleable**: unique encoding of (R, s)
- **Batch verification**: check multiple sigs faster
- **Simpler**: fewer operations than ECDSA

Bitcoin Taproot (BIP-340)

Activated Nov 2021; uses Schnorr on secp256k1 (x-only public keys, 32 bytes).

Signature Aggregation: MuSig Protocol





MuSig2 Protocol (3 rounds → 2 rounds)

1. Key aggregation: $a_i = H(L \| P_i)$, $\tilde{P} = \sum a_i P_i$
2. Nonce exchange: each signer sends R_i; $\tilde{R} = \sum R_i$
3. Partial sigs: $s_i = k_i + e \cdot a_i \cdot d_i$; aggregate $\tilde{s} = \sum s_i$

Space Savings

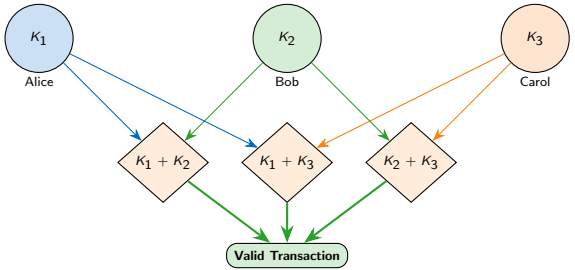
- 100-of-100 multisig: from 6400 B to 96 B
- Indistinguishable from single-signer tx
- Improved fungibility and privacy

MuSi

(2020): interactive 2-round protocol enabling key aggregation with provable security

Multi-Signature Schemes: t-of-n

2-of-3 Multisig



$\binom{3}{2} = 3$ valid combinations

General: $\binom{n}{t}$ combinations

Implementation Approaches

- **Script-based** (Bitcoin P2SH): explicit t-of-n in script
- **Schnorr/MuSig**: aggregated on-chain (Taproot)
- **Shamir + MPC**: threshold signatures off-chain

Common Configurations

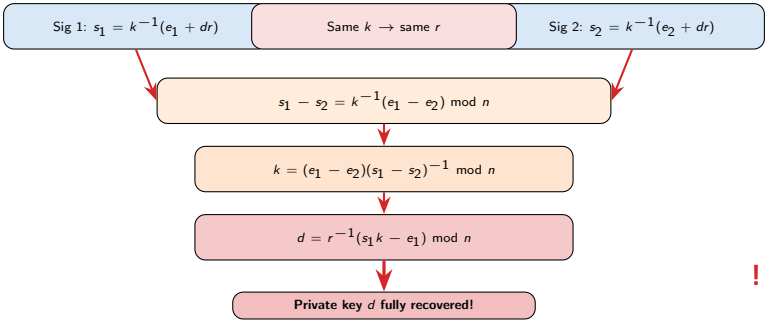
Setup	Use Case
2-of-3	Personal wallet + backup + partner
3-of-5	Corporate treasury
6-of-9	DAO governance
n-of-n	Unanimous consent (Lightning)

Multi

sig eliminates single points of failure: no single key compromise can drain funds

Nonce Reuse Vulnerability

DANGER: Same nonce k used twice



!

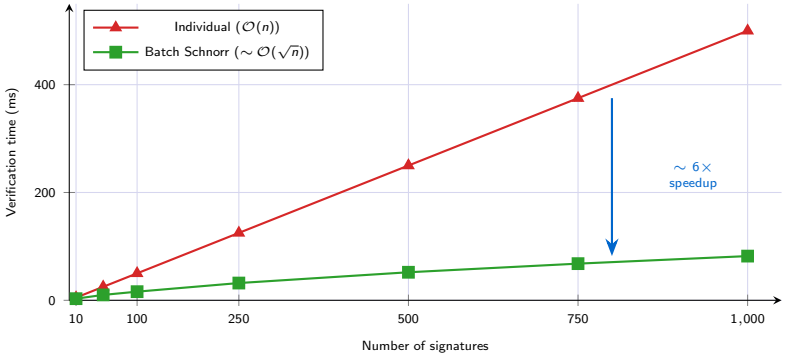
Real-World Incidents

- **Sony PS3 (2010)**: ECDSA with constant k; entire private key extracted; enabled homebrew
- **Android Bitcoin wallets (2013)**: faulty SecureRandom reused k values
- **Blockchain.info (2014)**: weak RNG led to nonce collisions

Defenses

- **RFC 6979**: deterministic $k = \text{HMAC}(d, H(m))$
- **Ed25519**: nonce derived from private key hash (by design)
- **Schnorr BIP-340**: auxiliary randomness mixed in
- Never use raw `random()` for nonce generation

Batch Verification Performance



How Batch Verification Works

Instead of n individual checks $s_i G = R_i + e_i P_i$, compute one multi-scalar multiplication:

$$\left(\sum z_i s_i\right) G = \sum z_i R_i + \sum z_i e_i P_i$$

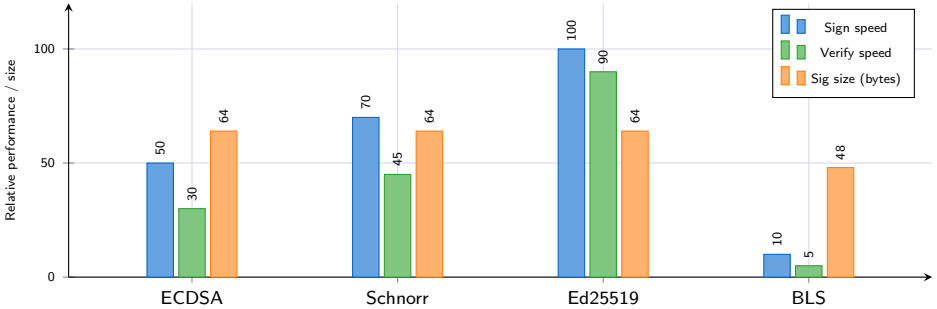
where z_i are random weights (for soundness).

Blockchain Impact

- **Block validation**: verify all txs in a block at once
- **Node sync**: significantly faster initial block download
- **Schnorr advantage**: ECDSA does not support batch verification
- Saves $\sim$ 50% CPU during peak load

Batch

Signature Schemes Comparison



Feature Matrix

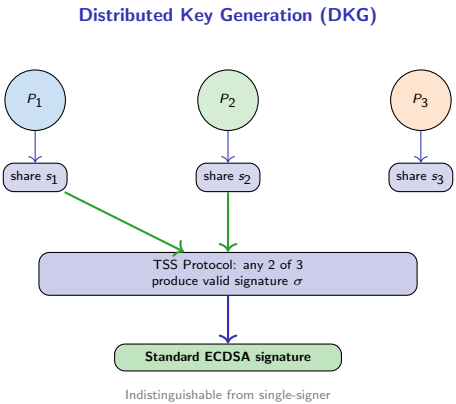
	ECDSA	Schnorr	Ed25519	BLS
Batch verify	×	✓	✓	✓
Aggregation	×	✓	×	✓
Det. nonce	RFC6979	BIP-340	Native	–
Pairing-free	✓	✓	✓	×

When to Use Which

- **ECDSA**: legacy compatibility (Bitcoin, Ethereum)
- **Schnorr**: aggregation, Taproot multisig
- **Ed25519**: new chains (Solana, Cardano)
- **BLS**: consensus sigs (Ethereum 2.0 validators)

signatures enable n validator signatures to aggregate into a single 48-byte proof – essential for PoS scalability

Threshold Signatures (TSS)



TSS vs. Multisig

	TSS	Multisig
On-chain footprint	Single sig	t sigs + script
Privacy	Hidden t-of-n	Visible on-chain
Gas cost	Standard tx	Higher (more data)
Key rotation	Re-share (no tx)	New address
Chain support	Any ECDSA chain	Chain-specific

Mathematical Foundation

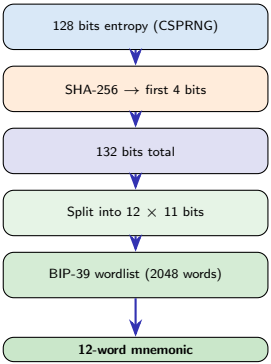
Private key d is **never reconstructed**. Each party holds Shamir share s_i such that $d = \sum_{i \in S} \lambda_i s_i$ (Lagrange coefficients). Signing uses MPC protocol where each party contributes partial signature.

Adoption

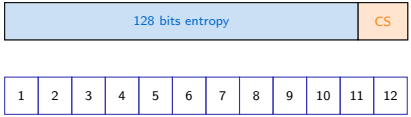
Fireblocks, ZenGo, Binance custody – all use TSS for institutional key management.

TSS provides multisig security with single-signature efficiency – the private key never exists in one place

BIP-39: Mnemonic Seed Phrases



Bit-level breakdown



12 groups of 11 bits = 132 bits

Example

```
abandon ability able about
above absent absorb abstract
absurd abuse access accident
```

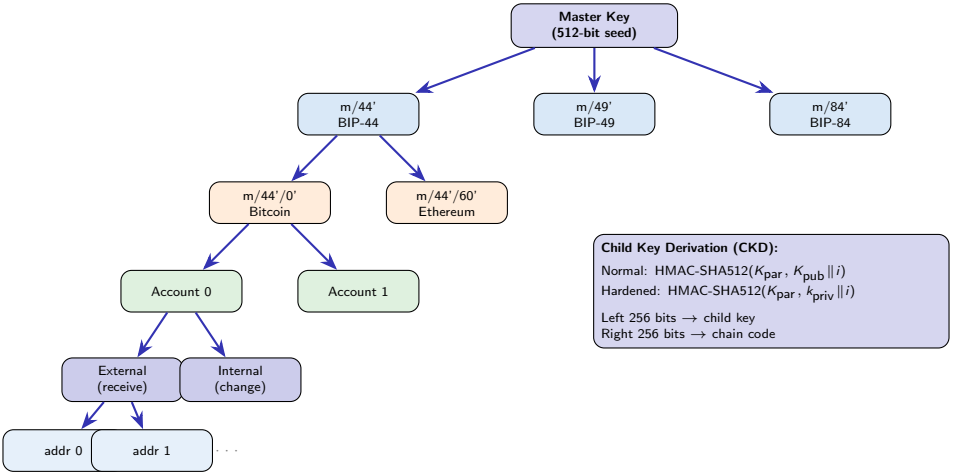
Each word encodes 11 bits of entropy

Entropy → Mnemonic → Seed

Mnemonic + optional passphrase $\xrightarrow{\text{PBKDF2 (2048 rounds)}}$ 512-bit seed → master key for HD wallet (BIP-32).

BIP-39 supports 12, 15, 18, 21, or 24 words (128–256 bits entropy); human-readable backup of all keys

BIP-32: Hierarchical Deterministic Wallets



Key Properties

- Single seed → unlimited keys

Normal vs. Hardened Derivation

- **Normal** ($i < 2^{31}$): xpub can derive child public keys

BIP-44: Multi-Account HD Derivation Paths

Standard Derivation Path

```
m / purpose' / coin_type' / account' / change / address_index
```

Level	Field	Values	Description
1	Purpose	$44'$ (BIP-44), $49'$ (SegWit), $84'$ (native SegWit)	Wallet standard
2	Coin type	$0'$ = BTC, $60'$ = ETH, $501'$ = SOL	SLIP-44 registry
3	Account	$0', 1', 2', \ldots$	User accounts
4	Change	0 = external (receive), 1 = internal (change)	Address role
5	Index	$0, 1, 2, \ldots$	Sequential addresses

Example Paths

Path	Meaning
`m/44'/0'/0'/0/0`	1st BTC receive address
`m/44'/0'/0'/1/0`	1st BTC change address
`m/44'/0'/1'/0/0`	2nd BTC account, 1st addr
`m/44'/60'/0'/0/0`	1st ETH address
`m/84'/0'/0'/0/0`	1st native SegWit BTC addr

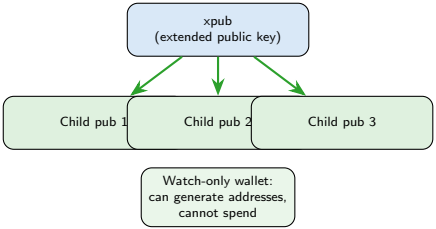
Why Standardized Paths Matter

- **Interoperability**: import seed in any BIP-44 wallet
- **Recovery**: deterministic path = predictable keys
- **Privacy**: new address per transaction (gap limit 20)
- **Organization**: separate accounts for different purposes

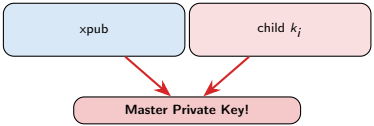
BIP-44 enables wallet portability: the same 12/24 words work across Ledger, Trezor, MetaMask, and Electrum

Key Derivation Security Analysis

Normal Child Derivation (Safe)



xpub + Child Private Key = DANGER



$$k_{\text{master}} = k_i - \text{HMAC}(\text{chaincode}, K_{\text{pub}} \| i)_L$$

Hardened Derivation (Fix)

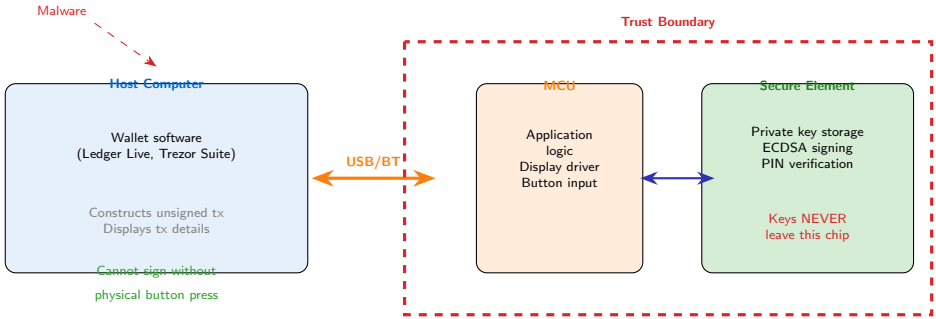
Uses private key in HMAC input $\rightarrow$
xpub alone cannot derive child keys
$\rightarrow$ chain broken, master key safe

Critical Rule

Never share xpub if any child private key might be exposed. Use hardened derivation (indicated by $'$) for account-level keys. Normal derivation is safe only below the hardened boundary.

This is why BIP-44 uses hardened derivation for purpose, coin type, and account levels

Hardware Wallet Architecture



Security Model

- **Air gap**: keys generated and stored on-device
- **Physical confirmation**: button press required
- **Tamper resistance**: secure element detects physical attacks
- **Open source firmware**: verifiable signing logic

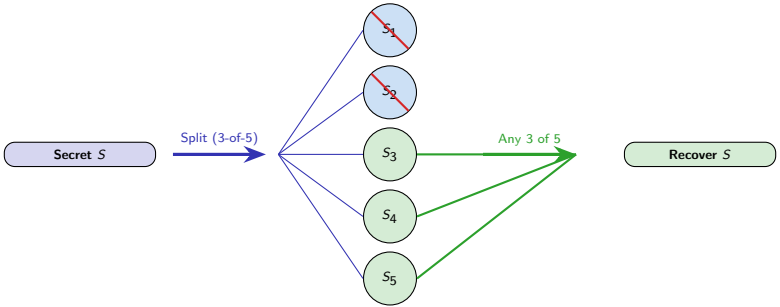
Threat Mitigation

Threat	Defense
Malware on host	Keys never on host
Supply chain attack	Attestation + open firmware
Physical theft	PIN + passphrase
Firmware backdoor	Reproducible builds
Side-channel	Secure element hardening

Hardw

wallets (Ledger, Trezor, Coldcard) provide the strongest practical key security for individual users

Wallet Recovery: Shamir's Secret Sharing



Polynomial Concept ($t = 3$)

$p(x) = S + a_1 x + a_2 x^2$

Share i: $(i,\ p(i))$

Any 3 points $\rightarrow$ unique degree-2 polynomial

Properties

- **Information-theoretic**: $t-1$ shares reveal *zero* information

Inheritance Planning

- 2-of-3: owner, spouse, lawyer
- 3-of-5: geographic distribution (safe deposit boxes)

Multi-Party Computation (MPC) Wallets

How MPC Signing Works

1. **Key generation**: n parties jointly generate key shares $k_1, \ldots, k_n$ via distributed key generation (DKG)
2. **No single key**: the private key sk is *never* assembled on any device
3. **Threshold signing**: t-of-n parties run interactive protocol producing a single valid ECDSA/EdDSA signature
4. **On-chain**: appears as a standard single-signature transaction

Key Advantage over Multisig

MPC is **protocol-agnostic**: works on any chain without native multisig support. Gas cost equals a single-sig tx.

MPC vs. Multisig vs. Shamir

Property	MPC	Multisig	Shamir
Key assembly	Never	N/A	At signing
On-chain cost	1-sig	t-sig	1-sig
Chain support	Any	Chain-specific	Any
Key rotation	Yes	New address	Re-share
Privacy	High	Visible on-chain	High

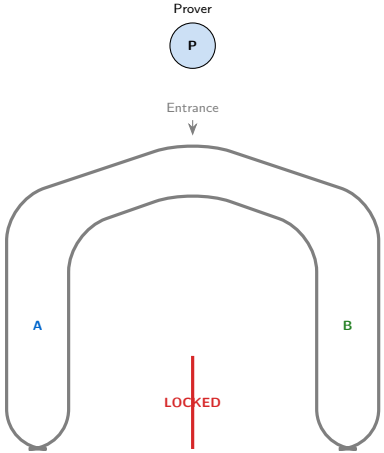
Institutional MPC Providers

- **Fireblocks**: institutional custody, \$6T+ transferred
- **ZenGo**: consumer MPC wallet (2-of-2)
- **Coinbase WaaS**: MPC-based wallet-as-a-service
- **Lit Protocol**: decentralized MPC key management

MPC

wallets eliminate single points of failure without on-chain overhead; dominant institutional custody model since 2022

Zero-Knowledge Proofs: The Ali Baba Cave



Protocol (repeat n times):
1. P enters cave, picks side A or B
2. V arrives, calls out A or B
3. P emerges from requested side (uses secret to unlock door if needed)
4. Prob(cheat) $= (1/2)^n$

ZK Properties:
- **Completeness**: honest prover always convinces verifier
- **Soundness**: cheater caught with prob $1 - (1/2)^n$
- **Zero-knowledge**: verifier learns nothing except validity

Formal Definition

A zero-knowledge proof for language L satisfies: (1) **Completeness**: $x \in L \Rightarrow \Pr[\text{Verify accepts}] = 1$; (2) **Soundness**: $x \notin L \Rightarrow \Pr[\text{Verify accepts}] \leq \text{negl}(\lambda)$; (3) **Zero-Knowledge**: $\exists$ simulator $\mathcal{S}$ producing indistinguishable transcripts without witness.

ZK-SNARKs vs. ZK-STARKs

Property	ZK-SNARKs	ZK-STARKs
Full name	Succinct Non-interactive Argument of Knowledge	Scalable Transparent Argument of Knowledge
Trusted setup	**Required** (toxic waste)	**Not required**
Proof size	$\sim$288 bytes (constant)	$\sim$45–200 KB (polylog)
Verification time	$\sim$10 ms (constant)	$\sim$50–100 ms (polylog)
Prover time	Moderate	Moderate
Quantum resistance	No (relies on pairings)	Yes (hash-based)
Math basis	Elliptic curve pairings	Hash functions + FRI
Post-quantum	Vulnerable	Resistant
Key deployment	Zcash, Tornado Cash, Filecoin, Groth16	StarkNet, StarkEx, zkSync (hybrid)

Trusted Setup Problem

- SNARKs require a ceremony generating CRS
- "Toxic waste" (secret randomness) must be destroyed
- If any participant is honest, setup is secure
- Zcash: multi-party computation ceremonies

Blockchain Applications

- **Privacy**: Zcash shielded transactions
- **Scaling**: zk-Rollups (Ethereum L2)
- **Compliance**: prove solvency without revealing balances
- **Identity**: prove age without revealing birthdate

Trade-off: SNARKs have smaller proofs but need trust; STARKs are transparent but proofs are larger

Commitment Schemes

Hash Commitment

Commit: $C = H(m \| r)$, send C

time passes

Reveal: send (m, r), check $H(m \| r) = C$

Binding: cannot find $m' \neq m$ with $H(m' \| r') = C$ (collision resistance)
Hiding: C reveals nothing about m (preimage resistance + random r)

Pedersen Commitment

$$C = m \cdot G + r \cdot H$$

Homomorphic property:
$C_1 + C_2 = (m_1 + m_2)G + (r_1 + r_2)H$
Can verify $m_1 + m_2 = m_3$ without revealing m_1, m_2, m_3!

Used in:
- Confidential Transactions (Monero, Liquid)
- Range proofs (Bulletproofs)
- Coin mixing protocols

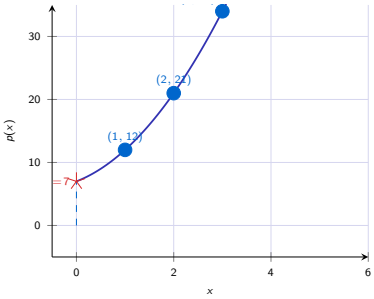
Commitment in Blockchain

Every transaction is a commitment: miners commit to a block (Merkle root) before broadcasting. Lightning HTLCs use hash commitments for atomic cross-channel payments.

Comm

are the "sealed envelope" of cryptography: binding and hiding until opened

Shamir's Secret Sharing: Mathematical Detail



Construction (t-of-n)

1. Secret $S = p(0)$, pick random $a_1, \ldots, a_{t-1}$
2. Polynomial: $p(x) = S + a_1 x + \cdots + a_{t-1} x^{t-1}$
3. Share i: $(i, p(i))$ for $i = 1, \ldots, n$
4. Any t shares $\rightarrow$ Lagrange interpolation $\rightarrow p(0) = S$

Lagrange Interpolation

Given points $(x_1, y_1), \ldots, (x_t, y_t)$:

$$S = p(0) = \sum_{i=1}^{t} y_i \prod_{\substack{j=1 \\ j \neq i}}^{t} \frac{-x_j}{x_i - x_j}$$

All arithmetic in $\mathbb{F}_p$ (finite field).

Security

$t-1$ shares give *no information*: any secret is equally consistent with $t-1$ points (infinite polynomials pass through $t-1$ points).

Exam

$p(x) = 7 + 3x + 2x^2$ **(secret** $= 7$**, threshold** $= 3$**): any 3 of 5 shares recover the secret**

Homomorphic Encryption Basics

Definition

A homomorphic encryption scheme allows computation on ciphertexts: $\text{Dec}(f(\text{Enc}(m_1), \text{Enc}(m_2))) = f(m_1, m_2)$ without decrypting.

Types of Homomorphic Encryption

Type	Operations	Example
Partial (PHE)	One type	RSA ($\times$) Paillier ($+$) ElGamal ($\times$)
Somewhat (SHE)	Both, limited depth	BGN
Fully (FHE)	Both, unlimited	TFHE, CKKS

Practical Considerations

- FHE: $\sim$ 10,000$\times$ slower than plaintext operations
- Ciphertext expansion: $\sim$ 100$\times$ size increase
- Bootstrapping: refresh noise for unlimited depth
- Active research: TFHE, CKKS improving rapidly

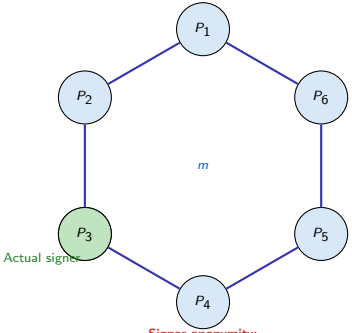
Blockchain Applications

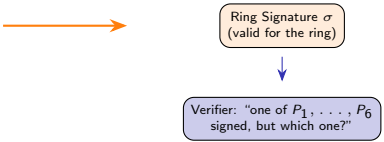
- **Private smart contracts**: compute on encrypted state
- **Encrypted auctions**: bid without revealing price
- **Private voting**: tally without decrypting votes
- Fhenix, Zama: FHE-enabled blockchains

RSA Multiplicative Homomorphism

$\text{Enc}(m_1) \cdot \text{Enc}(m_2) = m_1^e \cdot m_2^e = (m_1 m_2)^e = \text{Enc}(m_1 \cdot m_2) \pmod{n}$

Ring Signatures & Stealth Addresses





Stealth Addresses

1. Receiver publishes (A, B)
2. Sender picks random r
3. Computes $R = rG$ (ephemeral)
4. One-time address:
   $P = H(rA)G + B$
5. Only receiver can derive
   private key for P

Result: unlinkable payments

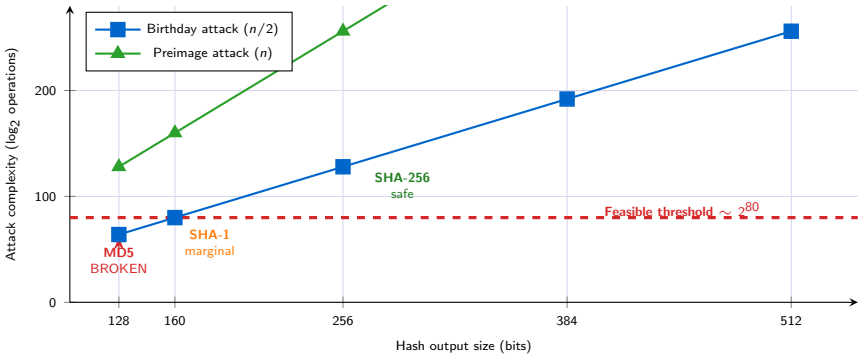
Monero Privacy Stack

- **Ring signatures**: hide sender among decoys

Trade-offs

- Ring size ↑ → privacy ↑, tx size ↑

Birthday Attack: Complexity Analysis



Attack History

- **MD5** (128-bit): collision in 2^{24} – seconds on laptop
- **SHA-1** (160-bit): SHAttered attack (2017, 2^{63})
- **SHA-256** (256-bit): 2^{128} – safe for decades

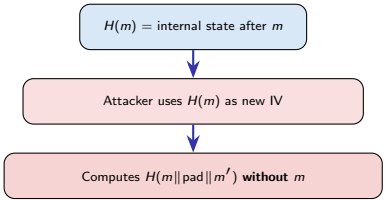
Implications for Blockchain

- Bitcoin addresses use RIPEMD-160: birthday at 2^{80}
- But address collision requires matching private key
- SHA-256 for block hashing: 2^{128} birthday bound
- Grover's algorithm: halves exponent (post-quantum)

Rule

Length Extension Attack & HMAC Defense

Length Extension Attack



Breaks: $\text{MAC}(k, m) = H(k \| m)$ (naive MAC)
Attacker can forge $\text{MAC}(k, m \| \text{pad} \| m')$

HMAC Defense

$$\text{HMAC}(k, m) = H(k \oplus \text{opad} \| H(k \oplus \text{ipad} \| m))$$

Why HMAC is safe:
- Outer hash wraps inner hash output
- Attacker cannot extend: would need to invert the outer hash first
- Two independent hash applications
- PRF under assumption H is PRF on compression function

Also immune:
- SHA-3 / KMAC (sponge: no internal state leak)
- BLAKE2 (built-in keyed mode)
- SHA-512/256 (truncated output)

Practical Impact

Flickr API (2009) used $H(\text{secret} \| \text{params})$ – vulnerable to forgery. Any Merkle–Damgård hash with $H(k \| m)$ as MAC is broken. Always use HMAC or a keyed hash mode.

Lengt

extension exploits Merkle–Damgård's iterative structure – the internal state after hashing IS the hash output

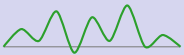
Side-Channel Attacks on Cryptographic Implementations

Timing Attack

Execution time varies with secret key bits

Key bit = 1 → slower

Power Analysis



Power consumption correlates with data

DPA: statistical analysis

Cache Timing

Cache hit/miss pattern reveals memory access

Flush+Reload, Prime+Probe

Mitigations

- **Constant-time code**: no secret-dependent branches
- **Masking**: randomize intermediate values
- **Shielding**: Faraday cages, power filtering
- **Blinding**: randomize inputs before computation

Notable Attacks

- **Spectre/Meltdown**: CPU speculation leaks
- **ECDSA timing**: OpenSSL key extraction (2011)
- **AES cache**: full key recovery in minutes
- **Power analysis**: smartcard key extraction

Side-channels attack the implementation, not the algorithm – correct code with wrong timing is still broken

Social Engineering Attacks on Crypto Users

Phishing

Fake wallet sites
clone MetaMask UI
steal seed phrase
approval scams

$1B+ lost (2023)

SIM Swap

Port victim's phone
intercept SMS 2FA
access exchange accts
social engineer carrier

FBI: $72M in 2023

Clipboard Hijack

Malware replaces
copied addresses
victim sends to
attacker's wallet

Undetectable visually

Ice Phishing

Trick user into signing
token approval tx
drain via transferFrom
no seed phrase needed

Top DeFi attack vector

Why Crypto Users Are Targets

- Irreversible transactions (no chargeback)
- Pseudonymous: attacker hard to trace
- Self-custody: no bank fraud department
- High value per compromise

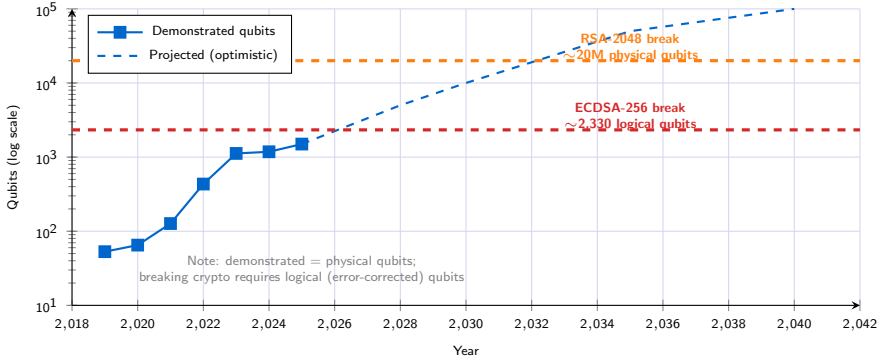
Defense Strategies

- **Hardware wallet**: signs offline, immune to phishing
- **Address whitelisting**: only send to verified addresses
- **FIDO2/WebAuthn**: phishing-resistant 2FA (not SMS)
- **Approval hygiene**: revoke unused token approvals
- **Verify on device**: confirm address on hardware screen

Socia

engineering bypasses all cryptographic security – the human is always the weakest link in the system

Quantum Threat: Shor's Algorithm



What Shor's Algorithm Breaks

- **RSA**: factors $n = pq$ in $\mathcal{O}((\log n)^3)$
- **ECDSA/EdDSA**: solves ECDLP in $\mathcal{O}((\log n)^3)$
- **DH/ECDH**: computes discrete log
- All current public-key crypto is vulnerable

Timeline Estimates

- **2025**: ~1,500 noisy physical qubits
- **2030–2035**: possible cryptographically relevant QC
- **Harvest now, decrypt later**: adversaries storing data today
- Migration should begin **now**

Shor

Grover's Algorithm: Impact on Symmetric Crypto

Grover's Algorithm

Quantum search over N elements in $\mathcal{O}(\sqrt{N})$ time vs. $\mathcal{O}(N)$ classically. Effectively halves the security parameter of symmetric primitives.

Algorithm	Classical Security	Post-Quantum Security	Recommendation
AES-128	2^{128}	2^{64} (insufficient)	Upgrade to AES-256
AES-192	2^{192}	2^{96} (marginal)	Acceptable for now
AES-256	2^{256}	2^{128} (safe)	**Recommended**
SHA-256 (preimage)	2^{256}	2^{128} (safe)	No change needed
SHA-256 (collision)	2^{128}	2^{85} (marginal)	Consider SHA-384+
HMAC-SHA-256	2^{128}	2^{85} (marginal)	Consider HMAC-SHA-512
ChaCha20 (256-bit)	2^{256}	2^{128} (safe)	**Recommended**

Key Insight

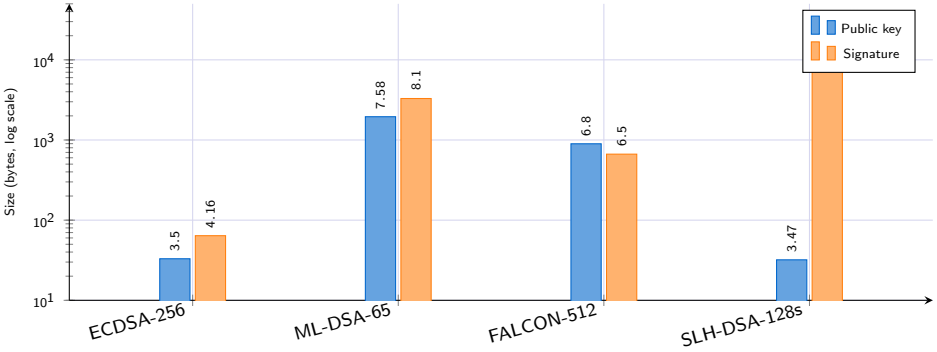
- Grover is a **square-root speedup** only
- Unlike Shor, does not break symmetric crypto
- Solution: double the key size

Bitcoin Impact

- SHA-256 mining: 2^{128} post-quantum – safe
- RIPEMD-160 addresses: 2^{80} preimage – marginal
- ECDSA: broken by Shor (not Grover)
- Priority: replace signatures, not hashes

(1996): generic quantum speedup – symmetric crypto survives by doubling key sizes

Post-Quantum Cryptography: NIST Standards



NIST PQC Standards (2024)

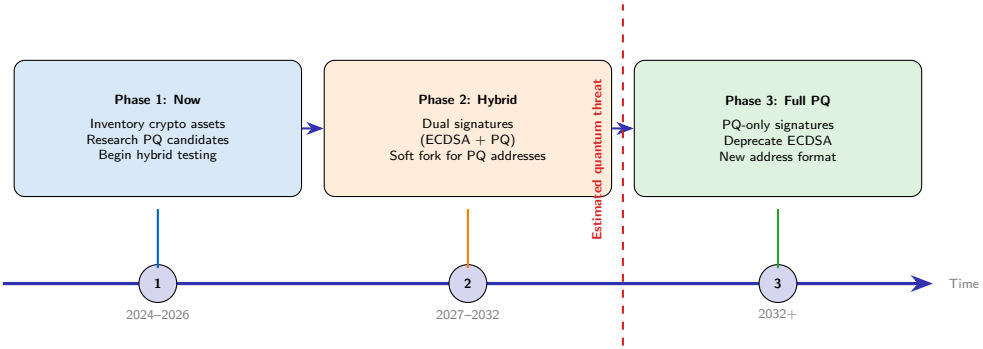
Standard	Basis	Type
ML-KEM	Lattices (MLWE)	Key encapsulation
ML-DSA	Lattices (MLWE)	Digital signature
SLH-DSA	Hash-based	Digital signature
FN-DSA	Lattices (NTRU)	Digital signature

Blockchain Challenges

- **Size**: PQ signatures are 10–100× larger
- **On-chain cost**: every byte is permanent and replicated
- **Compatibility**: address format changes needed
- **Performance**: lattice ops are fast; hash-based are slower

NIST finalized ML-DSA (Dilithium), ML-KEM (Kyber), and SLH-DSA (SPHINCS+) as first post-quantum standards in 2024

Blockchain Post-Quantum Migration Roadmap



Migration Challenges

- **Consensus**: hard/soft fork for new sig types
- **Lost coins**: addresses with lost keys cannot migrate
- **Backward compatibility**: support old and new formats
- **Testing**: PQ schemes less battle-tested

Current Initiatives

- **Bitcoin**: BIP proposals for PQ addresses (research)
- **Ethereum**: EIP-7693 (PQ account abstraction)
- **QRL**: quantum-resistant ledger (XMSS signatures)
- **NIST**: PQ standards finalized (Aug 2024)

now, decrypt later" – the migration to post-quantum cryptography is urgent even before quantum computers arrive

Summary & Key Formulas

Hash Functions

- Preimage: $\mathcal{O}(2^n)$ Collision: $\mathcal{O}(2^{n/2})$
- SHA-256: 64 rounds, 512-bit blocks, 256-bit digest
- Sponge (SHA-3): $b = r + c$, no length extension

ECDSA (secp256k1)

- Key: $d \in [1, n-1]$, $Q = dG$
- Sign: $r = (kG)_x$, $s = k^{-1}(e + dr)$
- Verify: $u_1 G + u_2 Q$, check $x = r$
- Security: 2^{128} (Pollard's rho)

Schnorr

- Sign: $R = kG$, $e = H(R \| P \| m)$, $s = k - ed$
- Verify: $sG = R - eP$
- Key advantage: linearity $\rightarrow$ aggregation

Diffie–Hellman

- Shared secret: $s = g^{ab} \bmod p$
- DDH assumption: (g^a, g^b, g^{ab}) vs. (g^a, g^b, g^c)

Shamir's Secret Sharing

- $p(x) = S + a_1 x + \cdots + a_{t-1} x^{t-1}$
- t points $\rightarrow$ Lagrange interpolation $\rightarrow S = p(0)$
- $t-1$ shares: zero information

Post-Quantum

- Shor: breaks RSA, ECDSA in $\mathcal{O}((\log n)^3)$
- Grover: $\sqrt{N}$ search $\rightarrow$ double symmetric key sizes
- NIST PQC: ML-DSA (lattice), SLH-DSA (hash)

Key Takeaways

1. ECC: compact keys, fast ops, proven security
2. Nonce reuse = total key compromise

Open Questions for Discussion

1. **When will quantum computers threaten Bitcoin?**

   Given current qubit progress ($\sim$1,500 in 2025) and the $\sim$2,330 logical qubits needed for ECDSA-256, what is a realistic timeline? How should the Bitcoin community prepare?

2. **Is perfect privacy possible on a public blockchain?**

   Ring signatures, zk-SNARKs, and stealth addresses provide strong privacy. Can we achieve both full transparency (for auditing) and full privacy (for users) simultaneously?

3. **Usability vs. security in key management?**

   24-word seed phrases, hardware wallets, and Shamir splits maximize security but create friction. How do we make self-custody accessible to non-technical users without compromising security?

Thank you!

Prof. Dr. Joerg Osterrieder