Cryptography | CRY

Zusammenfassung

1. SECURITY

Attacker Models: *CPA (can perform encryption queries, e.g. PGP with public key), CCA (can perform encryption + decryption queries, e.g. TLS)*
Security Goals: *IND (Indistinguishability) (ciphertexts for different messages look random/same), NM (Nonmalleability) (impossible to create ciphertext for related message without key)*
Semantic Security combines goal + model:
IND-CPA (ciphertext leaks no info if key is secret), IND-CCA (IND against attacker who can also make decryption queries), NM-CPA (can't forge cipher for related message knowing only the cipher), NM-CCA (can't forge cipher knowing cipher + able to make decryption queries)

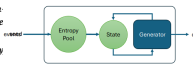
2. RANDOMNESS

Why it matters: Cryptography requires unpredictable secrets, nonces & IVs — without unpredictability, crypto is impossible.
Good randomness: Uniform distribution = *low entropy. Shannon entropy* measures the "surprise factor" of a distribution — lower entropy = more predictable.
Problem on PCs: Entropy sources (user input, temperature, lava lamps) are non-uniform or too slow.
PRNG *(Pseudo Random Number Generator)*: Fast, indistinguishable from true randomness. Seeded — same seed → same output.
PRF *(Pseudo Random Function Family)*: Deterministic output from key + input — same key & input → same output, but looks random.

2.1. PRNG

Security: Must use *cryptographic* PRNGs for crypto. Requires:
Prediction-resistance: given output sequence, next output is unpredictable.
Backtracking-resistance: given output sequence, previous state is unrecoverable.

Operation: State machine — state feeds generator, part of output feeds back into state
State seeded from entropy pool.
`init()` init pool, `refresh(r)` add entropy
`next(n)` output n random bits



2.2. PRF

Pseudo Random Function Family: key + input → deterministic output; looks random if key unknown. Believed example: SHA-256 HMAC.
KDF *(Key Derivation Function)*: deterministically derives new keys from a given key *(IPsec, WPA2/3, Argon2, crypto wallets — seed phrase reconstructs secret key).*
PRP *(Pseudo Random Permutation)*: bijective PRF — no collisions, every output has a pre-image. Believed example: AES.

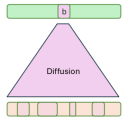
3. SYMMETRIC CRYPTOGRAPHY

Same key for encryption & decryption. Primitives: *Block Cipher (plaintext → random-looking ciphertext, reversible), Stream Cipher (random string XORed with stream), Hash (fingerprint of data), MAC (tamper-proof channel), AEAD (encryption + auth; associated data authenticated but not encrypted).*

3.1. BLOCK CIPHER

Used in TLS/HTTPS, IPsec, BitLocker, Kerberos. $D(K, E(K, P)) = P$. Block: 128 bit, Key: 128/256 bit.

Security goal: Output is a *PRP* — looks random, key hard to guess from ciphertext.
Block size trade-off: smaller = faster but vulnerable to codebook attacks; 128 bit fits CPU registers.
Confusion: complex relation between input/output bits. **Diffusion:** 1 input bit change → ½ output bits change. **Round paradigm:** simple function repeated many times, key expanded each round.

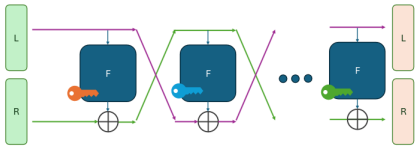


Round paradigm: simple function repeated many times, key expanded each round.

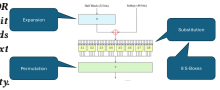
3.1.1. Feistel Network (DES)

$$L' = R,$$

Split block into halves L, R. Each round: $R' = Lo + F(R, K_i)$. 16 rounds, key rotated each round. Decrypt = same process, reversed key order.



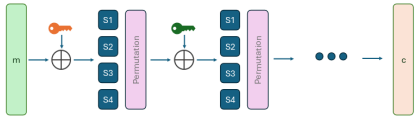
F-Function: 32 bit → expand to 48 bit → XOR with 48-bit subkey → 8 S-boxes (6→4 bit non-linear) → P-box permutation (spreads each S-box output across 4 S-boxes next round).



3DES: DES×3, 168-bit key → 112-bit security.

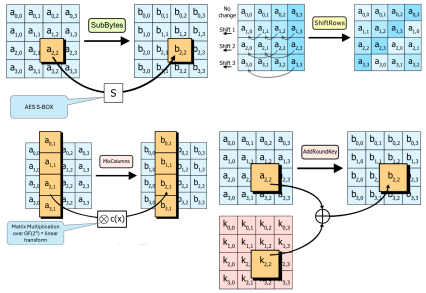
3.1.2. Substitution Permutation Network (AES)

Each round: XOR block with round key → S-boxes → permutation layer → next round.

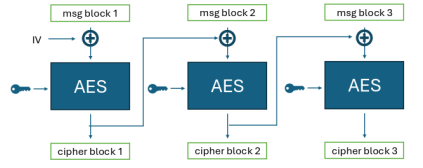


AES
128-bit block, operates on bytes arranged in 4 × 4 matrix. Rounds:
KeyExpansion: derives 128-bit round key each round.
SubBytes: S-box per byte — only non-linear step, protects against linear attacks.
ShiftRows: row i shifted by i positions.
MixColumns: each column multiplied by fixed matrix → every input byte affects every output byte.
AddRoundKey: each byte XORed with round key.

ShiftRows + MixColumns add diffusion and are easily reversible. Implemented via lookup tables or hardware; most modern CPUs have native AES instructions.



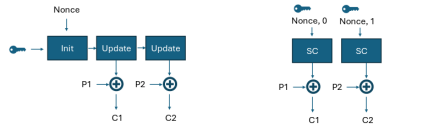
Modes of Operation
ECB: each block encrypted independently — **insecure**, patterns visible in ciphertext.
CBC: random IV XORed with first block; each cipher block feeds next round. Decrypt: reverse, XOR with previous cipher block. Same message encrypts differently each time.



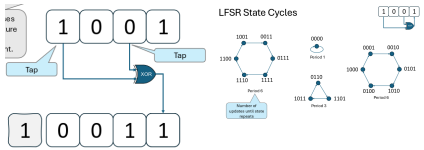
CTR: AES(Nonce ‖ counter) XORed with message block. Each block independently decryptable — good for unreliable networks.

3.2. STREAM CIPHER

Keystream XORed with message bits. Uses key + nonce (sent along) for uniqueness. Faster than block ciphers on hardware; no padding oracle attacks.
Security: keystream must be indistinguishable from true randomness (→ PRNG).
Stateful: each step depends on previous state; no random access.
Counter-based: counter as input; allows random access & partial recovery *(e.g. CTR mode).*



LFSR *(Linear Feedback Shift Register)*: new bit = linear combination of state; left-shift, output leftmost bit, append new bit. Goal: maximize cycle length — factor



LFSR polynomial to check *($X^4 + X^2 + 1$ reducible → shorter cycle).*

$$X^4X^3 + X^2X^2 + 1$$

Filtered LFSR: nonlinear function of state bits, but still solvable.

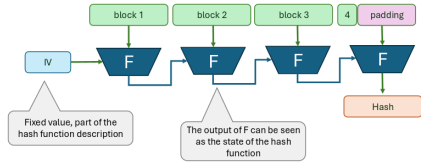
NFSR: hard to predict but period hard to compute. **Grain-128a:** lightweight cipher = NFSR + LFSR.
Salsa20: counter-based; F(key, nonce, ctr) + add back input → XOR with plaintext. Quarter Round uses ARX (add, rotate, XOR) for confusion + diffusion. **ChaCha20:** Salsa20 variant with stronger QR, larger initial state → less likely nonce reuse.

3.3. HASH FUNCTIONS

Any-length input → fixed-length fingerprint. Used in git, malware detection, deduplication, blockchain.
Collisions unavoidable (more inputs than outputs) → output must be **unpredictable**. CRC not suitable for crypto.
Preimage resistance: given y, infeasible to find x s.t. $H(x) = y$.
2nd preimage resistance: given x, infeasible to find $x' \neq x$ s.t. $H(x') = H(x)$.
Collision resistance: infeasible to find any $x \neq x'$ s.t. $H(x) = H(x')$.
Preimage resistant ⇏ collision resistant: dropping 1 bit keeps preimage resistance but birthday problem makes collisions easy.

3.3.1. Merkle-Damgård *(MD5, SHA-1/2)*

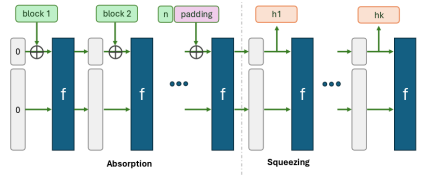
Compression function F: n bits in, m bits out. Message split into blocks, chained through F with IV. Block cipher can implement F.



Length extension attack: knowing H(secret ‖ data), an attacker can compute a valid H(secret ‖ data ‖ pad ‖ ext) without knowing the secret.

3.3.2. Sponge Construction *(SHA-3)*

Permutation-based; uses PRP at core. **Resistant to length extension attacks.**



3.4. MESSAGE AUTHENTICATION

Goal: authentic channel — proof data comes from sender. Uses shared key.
MAC *(Message Authentication Code)*: keyed function → tag = fingerprint of message sent alongside it. Used in IPsec, TLS.
Security: infeasible to forge tags or derive key from tags.
Replay attacks: MACs don't protect by default — Eve can replay valid messages. Fix: include counters.

3.4.1. HMAC

MACs are essentially PRFs: key + message → deterministic but random-looking output. Secure PRF = secure MAC.
Secret prefix $H(K \parallel M)$: vulnerable to length extension.
Secret suffix $H(M \parallel K)$: no length extension, but requires collision-resistant hash.
Envelope $H(K \parallel M \parallel K)$: better, still needs collision resistance.
HMAC (RFC 2104): $H(K \oplus \text{OPAD} \mid H(K \oplus \text{IPAD} \mid M))$ — two hashes, OPAD = 5c5c.., IPAD = 3636.. (block-length). Secure even with structurally weak H; only needs PRF compression function.

3.4.2. CMAC

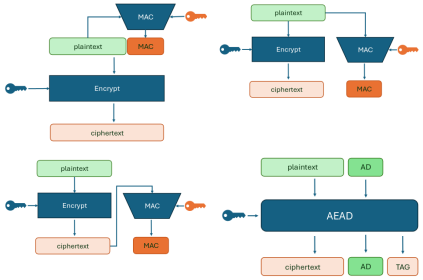
Block-cipher based MAC, similar to CBC-mode but last block processed differently; only last cipherblock kept.

3.4.3. Dedicated MAC

Crypto hash functions give stronger guarantees than needed — PRF-like compression suffices since attacker doesn't know the key (collisions useless without it).
Universal Hash: $\text{UH}(K, M)$: $\Pr[\text{UH}(K, M) = \text{UH}(K, M')] \approx 0$. Easier than full PRF, fast to evaluate.
One-Time MAC: key used only once (reuse leaks key info). Uses universal hash, very fast.
Poly1305-AES: $\text{MAC}(K_1, K_2, N, M) = \text{UH}(K_1, M) + \text{AES}(K_2, N)$ — one-time MAC + AES with two keys and nonce.

3.5. AUTHENTICATED ENCRYPTION (AEAD)

Combines confidentiality + authenticity in one. Associated Data (AD) *(e.g. network headers)* is public, authenticated but **not** encrypted.
Encrypt-and-MAC: risk of leaking info — MAC not designed to hide data.
MAC-then-Encrypt: used in TLS — vulnerable to padding oracle attacks.
Encrypt-then-MAC: provably secure, used in IPsec. ✓
Overall: encrypt plaintext → MAC over ciphertext → send → validate MAC → decrypt.



3.5.1. AES-GCM

Industry standard AEAD *(TLS, SSH, IPsec)*. Two components:
AES-CTR: message encryption.
GHASH: authentication via finite field multiplication.
IV **must** be unique. Fast on modern CPUs, fully parallelisable.

3.6. HARD PROBLEMS

Easy one way, infeasible to reverse. Used to build cryptographic schemes.
Types: *Decision (yes/no, e.g. "is 42 prime?", Halting problem undecidable), Functional (answer is a value, e.g. factoring, TSP path), Optimisation (best/approx solution, e.g. timetable, ML).*
Feasibility: $O(n^k)$ polynomial = feasible *(sorting, DB search, multiplication)* $O(2^n)$ super-polynomial = infeasible *(factoring, sudoku).*
Complexity classes: *P*: solvable in $O(n^k)$. *NP*: solution verifiable in poly time, finding it may be hard *(e.g. does K exist s.t. $C = \text{AES}(K, P)$?)*. *NP-Complete*: solving one solves all NP problems in poly time. *PSPACE*: solvable with $O(n^k)$ memory.
Factoring: given n, find primes p, q s.t. $n = pq$. Only one solution. Naïve: test all $x \in [2, \sqrt{N}]$ → $O(n^{n/2})$; faster algorithms exist but still exponential. RSA: public key n, secret p, q.
Discrete Log: group $G = (\mathbb{Z}_p^*, *)$, generator g *(every element in G is a multiple of g)*, find k s.t. $g^k = y$. Public key $y = g^k$, secret k. Must use specific large subgroups — small ones trivially solvable.

3.7. ASYMMETRIC CIPHERS

Public key encryptys, private key decrypts. Much slower than symmetric → typically only used to encrypt a symmetric key *(e.g. AES key)* for a hybrid scheme.

3.7.1. RSA

Supports both encryption+decryption and signing+verification (not the same operation).
Setup: $n = pq$ for large primes p, q. Choose e coprime to $\varphi(n) = (p-1)(q-1)$. Public key: (n, e). Private key: d s.t. $ed \equiv 1 \bmod(\varphi(n))$.
Encrypt: $y = x^e \bmod n$. **Decrypt:** $x = y^d \bmod n$.
Correctness: $y^d = (x^e)^d = x^{ed} = x^{1+q\varphi(n)} = x \cdot (x^{\varphi(n)})^q = x \cdot 1 = x$.
Security: breaking requires $d \rightarrow$ needs $\varphi(n) \rightarrow$ requires factoring n.
Example: $p = 3, q = 11, n = 33, \varphi(n) = 20, e = 3, d = 7$. Enc $x = 2$: $y = 2^3 \bmod 33 = 8$. Dec: $8^7 \bmod 33 = 2$ ✓
Only encrypts messages smaller than key size. Use established libraries — own implementation risky.

3.7.2. ElGamal

Based on hardness of discrete log. Group G of prime order q, generator g. Choose random $sk \in [0, q-1]$, compute $pk = g^{sk}$.
Enc: map $m \rightarrow M$ as group element; choose random $r \in [0, q-1]$; return $(c_1, c_2) = (g^r, M \cdot pk^r)$.
Dec: compute $s = c_1^{sk}$, recover $M = c_2 \cdot s^{-1}$, map $M \rightarrow m$.
DDH assumption: for random a, b, g^{ab} is indistinguishable from a random group element. ElGamal security reduces to DDH → IND-CPA secure.

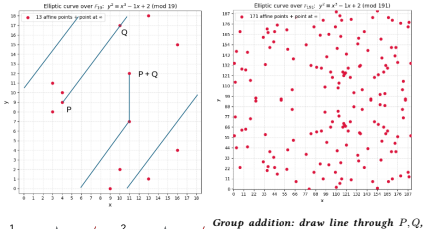
3.8. KEY EXCHANGE

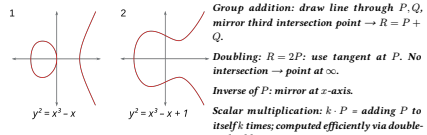
Authentic channel needed to prevent MITM — but authentic channels need shared keys. Chicken-and-egg problem solved by asymmetric crypto + signatures.

KEX *(Key Exchange Protocol)*: both parties contribute randomness to derive a shared key.
KEM *(Key Encapsulation Mechanism)*: one party generates key, encrypts with recipient's public key and sends it.

3.9. ELLIPTIC CURVES

Defined by Weierstrass equation $y^2 = x^3 + ax + b$. Over finite fields: looks like a random point cloud.





Group addition: draw line through P, Q, mirror third intersection point → $R = P + Q$.
Doubling: $R = 2P$: use tangent at P. No intersection → point at ∞.
Inverse of P: mirror at x-axis.
Scalar multiplication: $k \cdot P$ = adding P to itself k times; computed efficiently via double-and-add.

ECDLP: given points P and G, find scalar k s.t. $P = kG$. Believed hard for specific curves over finite fields.
Key size advantage: 128-bit security → 256-bit EC key vs 3072-bit RSA key. Smaller keys = faster computation and less bandwidth.

3.9.1. ECDH Key Exchange

Alice picks random a, sends $A = aG$. Bob picks random b, sends $B = bG$. Both over authentic channel.
Shared secret: $abG = aB = bA$. Attacker sees only A and B — cannot recover abG without solving ECDLP.
Use $H(abG)$ as actual key since abG is a curve point, not a uniform bitstring.
Security chain: DLP ≥ CDH ≥ DDH in hardness.

Breaking DLP breaks CDH and DDH.
If CDH holds, attacker cannot learn the full shared secret.
If DDH holds, the shared secret is indistinguishable from random.

3.10. SIGNATURES

Authentic channels need signatures; signatures need no prior shared secret — breaks the chicken-and-egg problem.
PKI: binds public keys to identities via X.509 certificates issued by Certificate Authorities. Bob verifies Alice's public key via CA signature.
Hash-then-sign: hash message first → fixed, predictable work regardless of message size.
Properties: *Correctness (valid sigs always verify), Integrity (any alteration invalidates), Unforgeability (infeasible to fake), Authenticity (confirms sender), Non-repudiation (signer can't deny).*
EUF-CMA: hard to forge even after seeing many signatures. Vulnerable to transaction malleability — valid signature for a different-but-related message.
SUF-CMA: closes transaction malleability gap — no valid alternative signature exists.

3.10.1. ECDSA

Public params: curve $C(F)$, base point G, prime n = group order. Private key: random $d \in \mathbb{Z}_n$. Public key: $P = dG$.
Sign: $h = H(m)$ as integer in $\mathbb{Z}_n$; random k; $(x, y) = kG$; $r = x \bmod n$; $s = (h + rd) \cdot k^{-1} \bmod n$. Signature: (r, s).
Verify: $u = hs^{-1} \bmod n$, $v = rs^{-1} \bmod n$; compute $(x', y') = uG + vP$; check $x' \equiv r$.

3.10.2. Ed25519

No random nonce needed → no nonce reuse risk. Faster than ECDSA. Same public params style as ECDSA.
Keygen: private key = random bitstring k *(usually 256 bit)*. Derive $a \parallel h = \text{SHA-512}(k)$. Public key: $P = aG$, use h for signing.
Sign: $r = H(h \parallel m)$, $R = rG$, $S = r + H(R \parallel P \parallel m) \cdot a \bmod n$. Signature: (R, S).
Verify: compute $H(R \parallel P \parallel m)$; check $SG = R + H(R \parallel P \parallel m) \cdot P$.

4. POST-QUANTUM CRYPTOGRAPHY

Quantum computers use *qubits (multiple states beyond 0/1, reversible gates)* — fundamentally different from classical computing. Shor's algorithm solves factoring and discrete log efficiently → breaks RSA, ECDSA, DH. Must prepare before quantum computers are widespread.
Grover's algorithm: speeds up brute-force search from $O(n)$ to $O(\sqrt{n})$ — not fast enough to break symmetric crypto, just double key size *(AES-128 → AES-256).*
Shor's algorithm: factors 2048-bit RSA modulus with only 2000–4000 qubits. Breaks all asymmetric schemes based on factoring or discrete log.
Affected: asymmetric encryption, key exchange, signing. Symmetric + hash functions remain safe with larger keys.

4.1. PQC APPROACHES

Lattices: Shortest Vector Problem in high-dimensional lattices is hard — basis for KEM, signatures, encryption.
Code-based: error-correcting codes, NP-hard.
Isogenies: maps between elliptic curves, random walks on curve graph — proven vulnerable.
Multivariate: hard polynomial equations *(e.g. Unbalanced Oil and Vinegar signatures).*
Hash-based: built on quantum-secure hash functions like SHA-2.

NIST approved schemes:

Purpose	*Scheme*	*Basis*
Signing	Dilithium	Lattice
Signing	Falcon	Lattice
Signing	SPHINCS+	Hash
KEM	Kyber	Lattice

PQC schemes are less tested, some broken already, and generally require more key space and computation.

4.2. PQC MIGRATION

1. Inventory: catalogue all crypto in your system — protocols, libraries, keys/certificates, secrets. Connect to policy questions *(acceptable TLS cipher suites? key protection requirements? approved libraries?).* Keep inventory + policy versioned and updated regularly. Ensure supply chain partners are also PQC-ready. Enables *cryptographic agility* — easy scheme replacement when something breaks.
2. Priorities: assess harvest-now-decrypt-later attacks — attacker stores encrypted data today, decrypts once quantum computer available. Rank impact of each vulnerable scheme as High/Medium/Low *(data loss, impersonation, etc.)* similar to standard threat modelling.
3. Migration: stay agile — build plan, execute, periodically reevaluate. Imperfect execution beats inaction. Replace components in-place or migrate service to new platform, then retire old.

5. ALGEBRAIC STRUCTURES

Structure	*Operations*	*Key Property*	*Examples*
Group	one $(*)$	assoc., identity, inverses	$\mathbb{Z}_n, +\, \mathbb{Z}_n^*, \cdot$
Ring	two $(+, *)$	group under $+$, monoid under $*$	$\mathbb{Z}_n, +, \cdot\ \mathbb{Z}[x]$
Field	two $(+, *)$	ring + mult. inverse for all $\neq 0$	$\mathbb{Z}_p, \mathbb{Q}, \mathbb{R}, GF(2^8)$

5.1. GROUPS

$(G, *)$ valid iff: *Associativity* $(a * b) * c = a * (b * c)$, *Neutral element* $a * e = a$, *Inverse* $a * \hat{a} = e$.
Group order $\text{ord}(G)$: size of G, e.g. $\text{ord}(\mathbb{Z}_m) = m$.
Element order $\text{ord}(a)$: smallest k s.t. $a^k = e$.
Generator g: $\text{ord}(g) = \text{ord}(G)$ — every element $a \in G$ can be written as g^k.
Example: $G = \mathbb{Z}_{15}^*$, $\text{ord}(G) = \varphi(15) = 8$. $\text{ord}(4) = 2$ since $4 \cdot 4 = 1$

5.2. RINGS

$(R, +, *)$: $(R, +)$ is abelian group, $(R, *)$ is monoid *(assoc. + neutral element, no inverses needed)*, distributive law holds. Division not always possible.
Examples: $(\mathbb{Z}_n, +, \cdot)$, $(\mathbb{Z}, +, \cdot)$, $\mathbb{Z}[x]$

5.3. FIELDS

Commutative ring where every non-zero element has a multiplicative inverse → $(\mathbb{F} \setminus \{0\}, *)$ is itself a group. Division always possible (except by 0).
Examples: $(\mathbb{Q}, +, \cdot)$, $(\mathbb{R}, +, \cdot)$, $(\mathbb{C}, +, \cdot)$
Finite fields: only two types: $GF(p) = \mathbb{Z}_p = \mathbb{F}_p$ for prime p, or $GF(p^k)$ for prime p and integer k.
Why prime p? Composite $n = p \cdot q \rightarrow p \cdot q = 0 \bmod(n)$ (zero divisors) → no mult. inverse for all elements → not a field.

5.4. EXTENSION FIELDS

Build larger field containing $\mathbb{F}$ as subset. Classic: $\mathbb{C}$ extends $\mathbb{R}$ by defining i s.t. $i^2 + 1 = 0 \rightarrow \mathbb{C} = \{a + bi \text{ mid } a, b \in \mathbb{R}\}$. For finite fields: $GF(p^k)$ extends $\mathbb{Z}_p$.

5.5. POLYNOMIAL RINGS

$\mathbb{F}[x]$: all polynomials over $\mathbb{F}$ — forms a ring.
Addition: component-wise mod p *(XOR for $\mathbb{Z}_2$)*. **Multiplication:** standard poly mult., reduce coefficients mod p.
$\mathbb{Z}_2$: $(x^2 + x + 1)(x^2 + 1) = x^4 + x^3 + 2x^2 + x + 1 = x^4 + x^3 + x + 1$ *since* $2 = 0$
Modulo $m(x)$: $\mathbb{F}[x]/m(x)$ — polynomials of degree $< \deg(m)$; multiply then reduce mod $m(x)$.
Irreducible $m(x)$: cannot be factored → $\mathbb{F}[x]/m(x)$ becomes a **field**, not just a ring.
AES field: $GF(2^8) = \mathbb{Z}_2[x]/m(x)$, $m(x) = x^8 + x^4 + x^3 + x + 1$ — elements are bytes, mult. is mod this polynomial.