1. Field of Cryptology

Classification:
- ** Symmetric Ciphers → Block / Stream
- ** Asymmetric Ciphers
- ** Protocols(symetric+ assmtrc)

2. Some Basic Facts

- Ancient encryption: Egypt (2000 B.C.)
- Symmetric ciphers: used until 1976
- Asymmetric ciphers:introduced by Diffie-Hellman-Merkle (1976)
- Hybrid systems use both(TLS,HTTP,IPsec,SSH)
  Symmetric → encryption & authentication
  Asymmetric → key exchange & signatures

3. Symmetric Cryptography

Term	Meaning
Alternative names	Private-key, secret-key, single-key
Key (K)	Same key for encryption & decryption
Plaintext (x)	Original message
Ciphertext (y)	Encrypted message

*Encryption → $y = e_K(x)$ *Decryption → $x = d_K(y)$

Secure only if key K is secret!(algorithm = public but key is scret)

Key Challenge: Secure transmission/storage of K , Brute forcing

Why Cryptanalysis?

No cipher can be **mathematically proven to be completely secure**.The **only practical way to ensure a cipher is strong** is to test it against attacks.

Kerckhoff's Principle:

"A cryptosystem should remain secure even if everything except the key is public.(untill key is secret)"

Attack Types in Cryptanalysis

1. Classical Cryptanalysis (Depends on the algorithm)

*__Mathematical analysis:__ Solve equations or exploit algorithmic weaknesses.

*__Brute force attack:__ Try all possible keys until the correct one is found.

2. Implementation Attack

*Exploit how the cipher is implemented.

*__ex:__ Side-channel attacks(measure power consumption, timing, electromagnetic leaks).

3. Social Engineering

*__Attack the human element rather than the algorithm.__

*__Examples:__ Phishing, tricking users to reveal keys or passwords.

Type	Example
Classical	Frequency analysis
Mathematical	Equation solving
Brute Force	Try all keys
Implementation	Power/timing analysis
Social Engineering	Tricking users

5. Brute-Force Attack

*Treats cipher as a black box

*Requires at least one known plaintext-ciphertext pair

*Test keys until $dK(y_0) = x_0$ systematically

Key Length	Key Space	Security Lifetime
64 bit	2^{64}	Short-term
128 bit	2^{128}	Long-term
256 bit	2^{256}	Long-term (Quantum-resistant)

Substitution Cipher

Idea: Replace each letter with another fixed letter.

Example: ABBA → kddk

Attacks for substitution attack

*__Brute-Force(Exhaustive Key Search):__ $26! \approx 2^{88}$ keys (too many, but still breakable)

*__Frequency Analysis:__

Common English letters: E (13%), T (9%), A, O, I, N...

Repeated patterns reveal plaintext.

Lesson: A large keyspace ≠ security — must resist *all* attacks.

Modular Arithmetic

Used in: Affine, Caesar, RSA, ECC

Definition: $a \equiv r \bmod m \rightarrow (r - a)$ divisible by m

Example: $12 \equiv 3 \bmod 9$, $37 \equiv 1 \bmod 9$, $-7 \equiv 2 \bmod 9$

Inverse Exists If: $\gcd(a, m) = 1$

Example: $7^{-1} \bmod 9 = 4$ (since $7 \times 4 \equiv 1 \bmod 9$)

Shift (Caesar) Cipher

Formula:

*Encryption → $y = x + k \bmod 26$

*Decryption → $x = y - k \bmod 26$

Ex: k = 7 → ATTACK → HAAHRH

Weakness: Only 26 keys → easily broken via brute-force,frequency analysis.

10. Affine Cipher

Formula:

*Encryption → $y = (K_1X + K_2) \bmod 26$

*Decryption → $X = K_1^{-1}(y - K_2) \bmod 26$

Requirement:

*K_1 must has inverse . so $\gcd(K_1, 26) = 1$. K_2 ona ekk

Gcd(8,26)

26= 4* 8 +2

8= 4*2 +0

gcd = 2

To get inverse we need extended E.A

Check a * $a^{-1} \bmod 26 = 1$

$3 * a^{-1} \bmod 26 = 1$?

a = 3 ➔

gcd(3,26)

*26 = 8*3 +2*

*3 = 1*2 +1*

*2=1*2+0*

gcd(3,26)

E.E.A

2= 26 – 8*3 ← A

1= 3 – 1(26 – 8*3)

1 = 9*3 -1*26 ← B

1 = 9*3 -1* 26 mod 26

9*3 mod 26 =1

$a^{-1} = 9$ (decrypt by 9 and encrypt by 3)

11. Vigenère Cipher

Idea: Use multiple Caesar shifts with a repeating key

Formula: ci = pi + ki mod 26

Period = Length of key

Weakness: Repetition patterns → key period guessing.

Perfect form: One-Time Pad (OTP) → key = plaintext length ∞ !Perfect security but impractical)

Columnar Transposition

Method: Rearrange plaintext in rows → read column-wise.

*Preserves letter frequencies

*More secure than substitution

*Harder to analyze via bigram/trigram frequencies

ENIGMA Machine

Type- Electro-mechanical cipher device

Used by- Nazi Germany (WWII)

Mechanism- Rotors change wiring after each keystroke

Decryption- Cracked by Allied cryptologists (Bletchley Park)

Use – known Plain text , reverse engineering

Stream Ciphers

1. Stream Ciphers in Cryptology

Types of Ciphers

*__Symmetric Ciphers__

__-Block Ciphers__ (encrypt fixed-size blocks)

__-Stream Ciphers__ (encrypt bit-by-bit)

*__Asymmetric Ciphers__

Origin: Stream ciphers were invented in **1917 by Gilbert Vernam.**

2. Stream Cipher vs. Block Cipher

Feature	Stream Cipher	Block Cipher
Encryption Unit	Individual bits	Fixed-size blocks
Speed	Usually small and fast,lightweight	Larger and slower
Common Use	Embedded systems (e.g., GSM A5/1) Mobile and hardware systm	Internet applications

Encryption and Decryption

For each bit:

*__Encryption:__ $y_i = x_i \oplus s_i$

*__Decryption:__ $x_i = y_i \oplus s_i$

Since XOR is reversible, encryption = decryption.

x_i	s_i	$y_i = x_i \oplus s_i$
0	0	0
0	1	1
1	0	1
1	1	0

How create stream of keys

*When you give the key , it generate stream of keys(k→ s_i)

*Security is depends on the S_i not K

*This is random, reproduceable (both can generate same), unpredictable (50%),depnds on A.S.S

4. Synchronous vs. Asynchronous Stream Ciphers

Type	Depends on	Notes
Synchronous	Key (and maybe IV) only	Keystream independent of ciphertext
Asynchronous	Key + Ciphertext	Uses ciphertext feedback

Security Requirement:

Keystream s_i must be **random (Pr(s_i=0)=Pr(s_i=1)=0.5)** and **reproducible**.

Random Number Generators (RNGs)

Types:

1.True RNG (TRNG) – Based on physical randomness (e.g., thermal noise, radioactive decay)

*Output: non-reproducible, unpredictable

*Used for: keys, nonces

*Example: Hardware entropy sources

2.Pseudo RNG (PRNG) – Based on a **seed**

*Output: reproducible and predictable

*Not actelly rndm ,hardcoded to prgm (initial is same &rest= same)

*Coming from here and there , predictable, reprocable

*In java *ran function*

*use for computer games

Not cryptographically secure due to linearity.

3.Cryptographically Secure PRNG (CSPRNG)

*Output **must be unpredictable** (even with known previous bits).

*Essential for cryptography and stream ciphers.

*__CSPRNGs__ are vital for unpredictability in cryptography.

One-Time Pad (OTP)

*__Developed by:__ Mauborgne, based on Vernam's cipher

*__Encryption:__ $y_i = x_i \oplus k_i$

*__Decryption:__ $x_i = y_i \oplus k_i$

*__Unconditionally Secure:__Even with infinite cmput^,cnt be broken.

*__Condition:__ Key used **only once** and must be **truly random**.

*__Disadvantage:__ Key length = message length→impractical for large data.

*Unlimited , Random, Unpredictable

One-Time Pad is perfectly secure but impractical.

7. Linear Feedback Shift Registers (LFSRs)

*Sequence generator built from **flip-flops** with **XOR feedback**.

*Degree m = number of storage elements.

*__Maximum sequence length:__ $2^m - 1$

Example (m=3):

CLK	FF_2	FF_2	FF_1
0	1	0	0
1	0	1	0
2	1	0	1
3	1	1	0
4	1	1	1
5	0	1	1
6	0	0	1
7	*1*	*0*	*0*
8	*0*	*1*	*0*
9	*1*	*0*	*1*

*Ekama wdht setup kloth reprocuable and its depend on number of registrs

*Q – bit 255 gnn registrs = 2^r -1 =255

Polynomial Representation: to get max bits gnn

Primitive policnnal = 1 and 1n bedann puluwn

Parallelly connectd , feedback wdht connct krnn

$P(x) = x^m + p_{r-1}x^{m-1} + ... + p_1x + p_0$

Single LFSRs are **predictable** – if 2m bits are known, coefficients can be recovered.

Combining several LFSRs → stronger cipher(good security)

Modern Stream Cipher – Trivium

Single LFSRs are weak, but **combined nonlinear ones** (like in Trivium) can be secure.

*Based on **3 Nonlinear LFSRs (NLFSRs)**

*Register lengths: **93, 84, 111 bits**

*__Total bits:__ 288

*__Operations:__ 3 AND gates, 7 XOR gates

Initialization

*Load 80-bit key into B

*Load 80-bit IV into A

*Set C_{108}, C_{109}, $C_{111} = 1$

*Clock 1152 times before output

Encryption

Keystream s_i = XOR of all 3 NLFSR outputs.

Register	Length	Feedback bit	Feedforward bit	AND inputs
A	93	69	66	91, 92
B	84	78	69	82, 83
C	111	87	66	109, 110

Data Encryption Standard (DES)

DES Facts

*Key size: **56 bits** (out of 64 bits, 8 bits are parity bits),insec

*Symmetric , block cipher

*Developed by **IBM (Lucifer cipher)** under NSA's influence

*__Standardized in 1977__ by NBS (now NIST)

*__aBest-studied__ symmetric cipher for 3 decades

*__Replaced by AES (2000)__

*__3DES (Triple DES)__ still used — very secure

2. Cryptographic Principles

Shannon's Primitives

Confusion – hides(obscured) relationship between key and ciphertext (via substitution)

Diffusion – spreads plaintext influence over many ciphertext bits (1 bit change krn ciphr ek entire change werw ywa permutation)

Product Ciphers:alternate confuse^+diffuse^ → strng encr^.

Modern block ciphers like DES and AES are **product ciphers**.

3. Overview of the DES Algorithm

*Rounds= 16

*Structure= Feistel Network

*Operations per Round =

Expansion → XOR with key → Substitution (S-box) →Permutation Each round uses a unique subkey derived from the main key via key schedule.

4. DES Feistel Network

Steps per Round

1. Split plaintext into **two halves**: L (left) and R (right)
2. Compute:
3. $L_i = R_{i-1}$
4. $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$
5. Swap halves
6. After round 16 → final swap → **Final Permutation (IP^{-1})**

Advantage:

Encryption and Decryption use the same process, only keys are reversed.

Internal Structure of DES

(a) Initial and Final Permutations

*Bitwise permutations defined by tables:

Initial Permutation (IP)

Final Permutation (IP^{-1})

*They are inverses of each other.

*Describe by tables IP and IP^{-1}

*changing places , not random , no encryption

*output = IP version of plain text

(b) Encryption

* Left side block= encrypting

* Right side block swapping to left side block

(c) f-Function (Main Operation)

Input: 32-bit R_{i-1} and 48-bit round key K_i

Output: 32-bit result. **Steps:**

1. **Expansion (E):**

*Expands 32-bit R → 48 bits (duplicates some bits)

*Provides diffusion.(32 →48)(16 bits duplicate)

2. **XOR with Round Key:**

*Expanded R ⊕ K_i

*Bitwise XOR

*Round keys are derived from the main key in the DES reschedule

3. **S-Boxes (Substitution):**

*8 S-boxes (each: 6-bit in → 4-bit out)

*Non-linear → provides confusion

*Core of DES security

4. **Permutation (P):**

*Rearranges bits

*Provides diffusion across S-box outputs

*Output bit of one s-box effect sevral S-Boxes in next round

After ~5 rounds, **every bit** of output depends on **every bit of key and plaintext**.

6. Key Schedule

*Generates **16 subkeys** (K_1 … K_{16}), each 48 bits.

*Bits 8,16,24. . are for error checking so not use

Steps:1. Start with **64-bit key** (8 parity bits → removed)

→ 56 bits remain (Permuted Choice 1, PC-1)

2.Split into C_0 and D_0 (28 bits each)

3.Rotate left each round:

Rounds 1, 2, 9, 16 → 1-bit rotation

Others → 2-bit rotation

4.Apply **Permuted Choice 2 (PC-2)** → select 48 bits as subkey K_i

After 16 rounds, total rotations = 28 → $C_0 = C_{16}$ and $D_0 = D_{16}$.

7. Decryption

*Same process as encryption, but **keys are applied in reverse order.**

Encryption	Decryption
Round 1 uses K_1	Round 1 uses K_{16}
Round 16 uses K_{16}	Round 16 uses K_1

Reason: Feistel structure ensures encrypt^+decryp^ with reversed keys. Rotation pattern for decryption:

*No rotation in round 1

*1-bit right rotation in rounds 2, 9, 16

*2-bit right rotation in all other rounds

8. Security of DES

Criticisms

*__Small key space:__ 2^{56} possible keys → vulnerable to brute force

*__Secret S-box design:__initially suspected backdoor(later disproven)

Strengths

*Resistant to **Differential** and **Linear Cryptanalysis**

*No known practical analytical attacks

Weakness

*__Brute force attack feasible__ with modern computing.

9. History of DES Attacks

Year	Attack / Achievement	Notes
1977	Diffie & Hellman	Proposed key search feasibility
1990	Biham & Shamir	Differential cryptanalysis
1993	Matsui	Linear cryptanalysis
1998	EFF "Deep Crack"	Broke DES in **56 hours** ($250k cost)
2006–08	COPACOBANA FPGA cluster	Broke DES in **6.4 days** ($10k cost)

Today: 56-bit DES can be broken in hours/days.

Triple DES (3DES)

*__Triple encryption__ using DES to extend key length.

*Common variant:C = $E_{K1}(D_{K2}(E_{K1}(P)))$

*Effective key length: **112 bits**

*Still used in **banking and legacy systems**.

*Setting $K_1 = K_2 \rightarrow$ single DES.

No practical attacks yet

Alternatives to DES

Algorithm	Block Size (bits)	Key Length (bits)	Remarks
AES / Rijndael	128	128 / 192 / 256	Official DES replacement
Triple DES	64	112 (effective)	Conservative, legacy systems
MARS	128	128 / 192 / 256	AES finalist
RC6	128	128 / 192 / 256	AES finalist
Serpent	128	128 / 192 / 256	AES finalist
Twofish	128	128 / 192 / 256	AES finalist
IDEA	64	128	Patented

AES is the global standard since 2001, providing:

*Larger block(128bits)/key sizes(128(10rounds),192(12), 256(14))

*Higher security

*Efficiency in hardware and software.

Block Cipher Modes of Operation & Security

Block Ciphers Overview

*Block ciphers are versatile:

-Encrypt data in fixed-size blocks

-Can be used for **stream ciphers, MACs, hash functions, PRNGs, key establishment**

*__Security enhancements:__

-Key whitening

-Multiple encryption (double/triple encryption)

Modes of Operation

Encrypt long plaintexts securely using different modes:

Mode	Key Idea	Advantages	Disadvantages
ECB – Electronic Codebook	Encrypts each block independently: yi = Ek(xi)	Parallelizable, simple, bit errors localized	Deterministic, same plaintext → same ciphertext, vulnerable to block substitution/reordering
CBC – Cipher Block Chaining	Each block XORed with previous ciphertext: yi = Ek(xi ⊕ yi-1)	Randomized (with IV), hides patterns	Sequential, not parallel, IV must be unique (not secret)
OFB – Output Feedback	Turns block cipher into stream cipher: keystream generated via cipher outputs	Stream-like encryption, errors don't propagate	Sequential generation, stateful
CFB – Cipher Feedback	Stream cipher using ciphertext to generate keystream	Can encrypt partial blocks, nondeterministic	Sequential, stateful
CTR – Counter	Keystream = Ek(IV		counter), XORed with plaintext
GCM – Galois Counter Mode	CTR encryption + authentication (MAC)	Confidentiality + integrity + authentication, parallelizable	Slightly more complex, requires correct IV & GF operations

ECB Weakness Example

*Repeated plaintext blocks → repeated ciphertext blocks

*Vulnerable to substitu^attacks(ex:tampering with e-bank transfers)

CBC Security Notes

*Uses **IV** to randomize first block

*Probabilistic: same plaintext → different ciphertext each time (with new IV)

*Prevents block substitution attacks if IV changes

CTR & GCM

*__CTR:__Parallelizable stream cipher,counter ensures unique keystream

*__GCM:__ Provides **encryption + authentication**

- MAC generated using Galois Field operations
- Ensures **message integrity & authenticity**

Exhaustive Key Search (Brute Force)

*DES: key length 56 bits → feasible with modern computing

*Probability of false keys depends on key space and block size

*Using multiple plaintext–ciphertext pairs reduces false positives:

Expected false keys = $2^{(k - t*n)}$ (k=key bits, n=block size, t=pairs)

Increasing Block Cipher Security

1.Double Encryption

*Encrypt with kL then kR

*Vulnerable to **Meet-in-the-Middle attack** → effective security ~2^{k+1}

2.Triple Encryption (3DES)

*y = Ek3(Ek2^-1(Ek1(x)))

*Effective key length ~112 bits

3.Key Whitening

*XOR plaintext and ciphertext with extra keys k1, k2

*Mainly increases resistance to brute-force

*Variant: **DESX**

Key Takeaways

*__ECB__: simple but insecure for repeated patterns

*__CBC, OFB, CFB, CTR, GCM__: provide varying levels of security, randomness, and parallelization

*__GCM__: best for combined confidentiality + integrity + authenticity

*__Double encryption__: marginal benefit; __triple encryption__ doubles effective key length

*__Key whitening__: low computational cost, improves brute-force resistance

Introduction to Public-Key Cryptography

Symmetric Cryptography Revisited

Same Key Used: Encryption and decryption use the *same secret key*

*__Fast and Efficient__, but has major limitations:

Key Distribution Problem: Securely sharing keys is difficult.

Scalability Issue: For n users, need $n(n-1)/2$ keys.

No Non-Repudiation: Either party can deny sending/receiving msg.

Principles of Asymmetric Cryptography

*Introduced in **1976** by **Whitfield Diffie** and **Martin Hellman** (and Ralph Merkle).

Party	Action	Operation
Alice	Encrypts message x using Bob's public key	F(x) = easy
Bob	Decrypts ciphertext y using his private key	F'(y)

*The keys are *mathematically related* but it's infeasible to derive Kpr from Kpub.

* **Disadvantage:** Slower (1000 times), only encrypt a small amount of data

Basic Public-Key Encryption Process

Key distribution problem solved (though authentication of public keys is still needed).

Security Mechanisms Enabled by Public-Key Cryptography

* __Key Exchange(establishmnt)__ (e.g., Diffie-Hellman, RSA)

* __Digital Signatures & Non-Repudiation__(RSA,DSA, ECDSA)

* __User Identification__ (Challenge-Response protocols)

* __Encryption__ (RSA, ElGamal)

Hybrid Cryptography (Key Transport Protocol)

*__Asymmetric cryptography__=used to exchange symmetric key (slow). *__Symmetric cryptography__=used for actual data encryption (fast).

Mathematical Foundations

One-Way Functions

Functions that are easy to compute but hard to reverse.

Problem	Example Algorithms
Integer Factorization	RSA
Discrete Logarithm	Diffie-Hellman, ElGamal, DSA
Elliptic Curve Discrete Logarithm (generalization of discrete logarithm)	ECDH, ECDSA

The problems are considered mathematically hard, but no proof exists .

Key Lengths and Security Levels

Security Level	Symmetric	RSA / DL	ECC
Short-term (few hours/days)	64-bit	~700-bit	128-bit
Medium	80-bit	1024-bit	160-bit
Long-term	128-bit	3072-bit	256-bit

Quantum Threat:

Quantum computers could break RSA,ECC, DL algorithms in future.

Essential Number Theory = key in RSA

a) Euclidean Algorithm

Finds **GCD and then use EEA to get inverse**

b) Euler's Phi Function Φ(m)

*** how many numbrs in the set are relatively prime to m ?**

Z_8 = {0,1,2,3,4,5,6,7} and m =6 nm gcd (6,Z_8) hoyl 1 ena ewa gaana = **Φ(m)**

*Counts numbers less than m that are **coprime** to m.

*If $m = p \times q$, then $\Phi(m) = (p-1)(q-1)$

*P, q should be prime

Example:

Φ(899) = (29−1)(31−1) = 28 × 30 = **840**

Primes numbers = p-1 = Φ(m)

c) Fermat's Little Theorem

For prime p and integer a: $a^{(p-1)} \equiv 1 \pmod p$

Used to compute modular inverses.

d) Euler's Theorem

Generalization of Fermat's theorem:

If gcd(a, m) = 1, then

$a^{\Phi(m)} \equiv 1 \pmod m$

RSA

*__1976:__ Diffie & Hellman published the first public-key cryptography concept.

*__1977: Rivest, Shamir, Adleman (RSA)__ introduced the RSA algorithm.

*__RSA__ remains the most widely used asymmetric algorithm (though **ECC** is faster).

Core Principle

Operates over the **integer ring Zn**, where n = p × q and p, q are large primes. N is compliment .

*__Encryption and decryption__ are exponentiations modulo n.

Encryption: $y \equiv x^e \bmod n$

Decryption: $x \equiv y^d \bmod n$

*__Public Key:__ (n, e)

*__Private Key:__ d

*__Security Basis:__ It's computationally hard to derive d from (n, e) (depends on integer factorization).

*__Plaintext , ciphertext come from group Zn{1,2…..,n-1}__

Key Generation Algorithm

Output : public key (k_{pub} =(n,e) and private key (k_{pr} =d))

1. Choose two large primes **p**, **q**.
2. Choose **n = p * q**.
3. Compute **Φ(n) = (p−1)(q−1)**.
4. Choose **e public exponent** such that **1 < e < Φ(n)** and **gcd(e, Φ(n)) = 1**.
   Rules :come from group Z_n {1,…n-1} and **gcd(e, Φ(n)) = 1.**
5. Compute pvt key **d** as the modular inverse of e mod Φ(n):
   $d \times e \equiv 1 (\bmod \Phi(n))$

*EEA: e * ? mod* Φ(n) =1

gcd (e, Φ(n)) =1 , EEA argnn …e enne pvt key

6. Output:

*Public key: **kpub = (n, e)***

*Private key: **kpr = d***

*__Public key (e)__ can be short, **private key (d)** must be long.

Implementation Aspects

*RSA involves **modular exponentiat**^ :cmputat^nlly expensive with large numbers.

*__Slower__ than symmetric algorithms (DES, AES).

*Optimizations are crucial, especially for **smart cards**, **IoT**, or **mobile devices**.

Square-and-Multiply Algorithm

Efficient way to compute $x^e \bmod n$.

*Converts exponent e to binary.

*Performs squaring and multiplication only when a bit = 1.

*Significantly reduces operations.

Example: To compute x^{26}, binary of 26 = 11010_2

Speed-Up Techniques

1.Small public exponent (e)

*Common values: 3, 17, 65537 (2^{16}+1).

* Improves encryption speed without weakening security.(SSL,TLS)

2.Chinese Remainder Theorem (CRT)

*Accelerates decryption:

*Roughly **4× faster** than standard decryption.

*Exponentiation with pre-computation

Finding Large Primes

RSA key generation requires **two large primes p and q**.

*Each prime ≈ half the bit length of n.

*Typically **1024–4096 bits** total.

*Generated using:

Random Number Generator (RNG)

Primality Tests:RNG must be **cryptographically secure** to prevent attackers from predicting p, q.

Primality Tests

Probabilistic methods determine if a number is *likely* prime. 100% n , bt hodai

Test	Description
Fermat Test	Uses Fermat's Little Theorem: if $a^{(p-1)} \neq 1 \bmod p \rightarrow$ composite
Miller–Rabin Test	Repeated random checks → highly reliable

Output is "prime with very high probability".

Attacks and Countermeasures

Types of Attacks

1.Analytical Attacks

*Target the *mathematical structure*.

*: **Factoring n** to derive Φ(n).. ntnm p, q hoygnnw,then break

* Countermeasure: Use large n (1024 – 3072)

2.Protocol Attacks

*Exploit **malleability** of RSA (ciphertext alteration).

* Countermeasure: Use **secure padding schemes** (e.g., OAEP).

3.Implementation Attacks

*Exploit weaknesses in hardware/software implementation.

*Types: **Side-channel analysis:** Uses power, timing, or EM leaks.

Fault injection: Forces device errors to reveal private keys.

Mathematical attacks

Countermeasures: Physical shielding, random delays, fault detection, Combine RSA with **padding (OAEP, PSS)**

RMURA-KSHA

Diffie–Hellman Key Exchange (DHKE)

- Proposed by **Whitfield Diffie** and **Martin Hellman** (1976).
- Used in **SSH, TLS, IPSec**.
- A **key exchange** protocol — *not encryption*.
- For encryption, **ElGamal** is built on top of DH.

Setup
1.Choose a large prime p.
2.Choose a primitive root (generator) α
3.Public parameters: (p, α).(domain parameter)

Alice → Bob
Alice : (dennama mehm krnw ,exchange , samanai)
1.private key A_{pvt} = (1 < and <p-1)
2.public key $A_{pb} = \alpha^{\wedge} A_{pvt}$ mod p.

Shared Secret
Both compute:
kAB = α^(ab) mod p
Alice computes (B)^a mod p, Bob computes (A)^b mod p.
They both get the **same key kAB**.

Domain Parameter
Z_{29} **{1,…,28}**
This working for only Finite Celic Groups only. Z^*_n **Only Finite= limited**
Group =

- group operation is closed (grp ek athule 2k guna krm < Zn)
- else >Zn → (2*4) mod 5 mek < Zn
- a(a.b) = (a.b)c = assosiative
- neutral elements (identity element) must a.1 =a (0*1 ,1*1,2*1, all =a)
- group abelian = a.b = b.a = commutative
- gcd (Zn , mmbrs) = 1

*****cardinality = number of membrs
if cyclic group as following
Z^*_{11} - {1,2,3,4,5,6,7,8,9,10}
$3^0 = 1$ ord (3) = 5
$3^1 = 3$
$3^2 = 9$
$3^3 = 5$
$3^4 = 4$
$3^5 = 1$

Generator should be ::: Ord (n) = cardinality
Brute force not working to get P

2. Discrete Logarithm Problem (DLP)
Works in finite cyclic groups
Ex : Elliptic curve (goood but brute force can) ,Zp*

3. Attacks on DLP

Generic Algorithms
Work for all cyclic groups:

- Brute Force
- Shanka's Baby-Step Giant-Step
- Pollard's Rho
- Pohlig–Hellman

Non-generic Algorithms
Work for specific groups (like Zp):

- **Index Calculus Method**

Elliptic curves resist non-generic attacks → shorter key sizes are secure.

Recommended Key Size
Use **p ≥ 1024 bits** (for ~80-bit security).
For stronger security → **2048 bits or higher**.

4. Security of Diffie–Hellman
Attacker (Oscar) knows:

- Public data: p, α, A=α^a mod p, B=α^b mod p

Goal: *Compute shared key:

- k = α^(ab) mod p
- This is the **Diffie–Hellman Problem (DHP)**.
- Solving DHP requires solving the **DLP**.
- Hence, security of DH = difficulty of solving DLP.

5. ElGamal Encryption

Overview

- Proposed by **Taher ElGamal (1985)**.
- Based on **DLP** and **DH problem**.
- A **public-key encryption scheme** derived from DH. Not for key exchange
- DH break krot ELGamal th break , so sec same
- **Passive attack:** Needs to solve DLP to get message.
- **Active attack:** Possible if:
  Keys not authenticated.
  Same random i reused (leaks information).

6. Lessons Learned

Concept	Key Point
Diffie–Hellman	Key exchange protocol, not encryption
DLP	Core hardness assumption in many crypto systems
Key size	p ≥ 1024 bits (2048 for high security)
ElGamal	Extension of DH, probabilistic encryption
Security	Based on DLP and DHP; must avoid key reuse

Digital Signatures

1. Concept & Motivation
Problem:

- Symmetric crypto fails → both parties could cheat.
- **Solution:** Use **public-key cryptography** for verifiable proof.

2. Basic Principle

Function	Input	Key Used	Output
Signature Generation	Message x	Private Key Kpr	Signature s
Verification	(x, s)	Public Key Kpub	Valid / Invalid

- ✅ Only the private key owner can generate the signature.
- ✅ Anyone can verify using the public key.
- ✅ Signature changes for every message.

3. Security Services Provided

Service	Description
1. Confidentiality	Only authorized parties can read data
2. Integrity	Message not altered in transit
3. Authentication	Confirms sender's identity
4. Non-repudiation	Sender cannot deny sending the message
5. Identification	Verifies entity (user/device)
6. Access Control	Restricts resource access
7. Availability	System reliability ensured
8. Auditing	Logs activities for verification
9. Physical Security	Protects systems from physical attacks
10. Anonymity	Protects user identity

4. RSA Digital Signature Scheme

Key Generation
Same as RSA encryption:

- Choose p,q
- n = p × q
- φ(n) = (p−1)(q−1)
- choose e

d = e^{-1} check 7 mod 20 (EEA)
Public key = (n, e) Private key = d

Signing (Sign with private key)
s = x^d mod n

Verification
y' = s^e mod n
If x' ≡ x mod n → Valid
Else → Invalid

Digital Signatures
Ensure:
- ✅ Integrity
- ✅ Authentication
- ✅ Non-repudiation

RSA Digital Signature

- **Keys:**
  $n = p \times q,\ \varphi(n) = (p-1)(q-1)$
  Choose e, find $d \rightarrow e \times d \equiv 1 (mod \varphi(n))$
- **Sign:** $s = x^d \bmod n$
- **Verify:** $s^e \bmod n = x$
- **Security:** Same as RSA encryption (≥1024 bits)
- **Weakness:** Existential forgery (mitigate using **RSA-PSS**)
- **Performance:** Verification faster (small e)

ElGamal Digital Signature

- **Based on:** Discrete Logarithm Problem (DLP)
- **Keys:**
  Public: $(p, \alpha, \beta = \alpha^d \bmod p)$
  Private: d
- **Sign:**
  $$r = \alpha^{k_E} mod p$$
  $$s = (x - d \cdot r) \times k_E^{-1} mod (p-1)$$
- **Verify:** $\beta^r \times r^s \equiv \alpha^x \bmod p$
- **Note:** Never reuse k_E!
- **Risk:** Reused k_E → private key leak

Digital Signature Algorithm (DSA)

- **Standard:** U.S. DSS (based on ElGamal + SHA-1)
- **Keys:**
  - p, q such that $q \mid (p-1)$
  - α with order q
  - Public: $(p, q, \alpha, \beta = \alpha^d mod p)$, Private: d
- **Verify:** $w = s^{-1} mod q$
  $u1 = SHA(x) \cdot w mod q,\ u2 = r \cdot w mod q$
  $$v = (\alpha^{u1} \cdot \beta^{u2} mod p) mod q$$

✅ Valid if $v = r$

- **Security:**

Based on **DLP** in subgroup of order q
Larger p, q = higher security

p (bits)	q (bits)	Security (bits)
1024	160	80
2048	224	112
3072	256	128

Elliptic Curve DSA (ECDSA)

- **Based on:** Elliptic Curve Cryptography (ECC)
- **Efficiency:**
  - **160–256 bit ECC ≈ 1024–3072 bit RSA**
  - **Smaller keys, faster, less bandwidth**
- **Signature:** Two values (r, s)
- **Security:** Depends on Elliptic Curve Discrete Logarithm Problem (ECDLP)

Security Recommendations

Algorithm	Key Length (min)	Strength
RSA / DSA	≥ 1024 bits (preferably 3072)	High
ECDSA	160–256 bits	Equivalent strength
RSA-PSS	Recommended padding (anti-forgery)	Secure

8. Lessons Learned

Algorithm	Based On	Signature Length	Speed	Notes
RSA	Integer factorization	≥1024 bits	Fast verification	Most widely used
ElGamal	DLP	Variable	Moderate	Reuse of kE breaks security
DSA	ElGamal variant	320 bits	Slower verification	Standardized (NIST)
ECDSA	ECC	320–512 bits	Fast & lightweight	Same security with small keys

Cryptographic Hash

1. Motivation

Problem with Direct Signing of Long Messages
If a 256 MB file is signed directly with RSA (256-bit signature):

- **Computation Overhead:** 1 million RSA operations needed ❌
- **Message Overhead:** Total 512 MB transmitted ❌
- **Security Issue:** Attackers can replace/remove blocks ❌

✅ **Solution:**
Use a **hash function** to generate a *short, fixed-length digest* (fingerprint) of the message, and **sign only the hash**. efficient

2. Digital Signature with Hash Function

Process:
1. Compute hash of message → z = h(x)
2. Sign the hash → s = sigKpr(z)
3. Send (x, s) to receiver(hash ek , sign)
4. Receiver computes z' = h(x) and verifies verKpub(s, z')

3. Hash Function Behavior

Property	Description
Input	Arbitrary length message (no fix length , any length)
Output	Fixed length digest (128–512 bits)
Efficiency	Should be fast and computationally lightweight
Diffusion	Small input changes → large unpredictable output change

4. Security Properties

Property	Meaning	Goal
Preimage Resistance (One-Way,no reverse)	Given z, it's infeasible to find x s.t. h(x)=z	Protect message secrecy
Second Preimage Resistance (e hash ek gnn wen msg n)	Given x1, infeasible to find x2 ≠ x1 s.t. h(x1)=h(x2)	Protect message integrity
Collision Resistance (no any pairs with same hash) (easy for attckr , sec ta amrun)	Infeasible to find any pair (x1, x2) s.t. h(x1)=h(x2)	Prevent forgeries

✅ **Collision resistance** is most difficult to achieve.

5. Birthday Paradox & Hash Security

- **Birthday paradox:** Only 23 people → 50% chance two share same birthday
- For hash functions → collisions appear after roughly $\sqrt{(2^n)}$ operations.

Output Size	Approx. Security	Notes
128 bits	~64-bit collision resistance	Weak today
160 bits	~80-bit security	Minimum recommended
256 bits	~128-bit security	Strong for long term

- **Hence:** Hashes need ≥ 160 bits for 80-bit security, ≥ 256 bits for modern strength.

6. SHA-1 – Secure Hash Algorithm

- Member of **MD-4 family**
- Uses **Merkle–Damgård construction**
- **Output:** 160 bits
- **Input blocks:** 512 bits
- **Operations:** Bitwise AND, OR, XOR, complement, cyclic shifts
- **80 rounds** divided into **4 stages** of 20 rounds

Padding Rule
To ensure input fits 512-bit blocks:
x → x || 1 || k×0 || 64-bit(length)
k = 448 − (l + 1) mod 512
(where l = length of original message)

Computation Process
1. **Initialize** hash value H0 (five 32-bit registers: A–E)
2. **Divide** message into 512-bit blocks
3. **Expand** into 80 32-bit words W0..W79
4. **Perform 80 rounds** updating A–E
5. After each block:
6. H(i) = H(i−1) + (A,B,C,D,E)
7. **Final digest** Hn = h(x) (160 bits)

Stage Constants & Functions

Stage	Rounds	Constant	Function f(B,C,D)
1	0–19	5A827999	(BAC) ∨ (¬BAD)
2	20–39	6ED9EBA1	B ⊕ C ⊕ D
3	40–59	8F1BBCDC	(BAC) ∨ (BAD) ∨ (CAD)
4	60–79	CA62C1D6	B ⊕ C ⊕ D

8. Lessons Learned

Concept	Key Takeaway
Hash Functions	Keyless cryptographic primitives
Uses	Digital Signatures, MACs (HMAC), Data integrity
Security Requirements	One-wayness, 2nd-preimage resistance, Collision resistance
Minimum Output Length	≥ 160 bits (SHA-1 weak, prefer SHA-2/3)
MD5 & SHA-1	Insecure – should be phased out
SHA-2 / SHA-3	Secure and standardized
Applications	SSL/TLS, Blockchain, Password storage, Digital signatures

✅ **Summary Table**

Algorithm	Output (bits)	Status	Security Level	Notes / Common Use
MD4	128	❌ Broken	Very Low	Outdated, early hash design
MD5	128	❌ Insecure	Low	Vulnerable to collisions, legacy only
SHA-1	160	⚠️ Weak	~80 bits	Known collisions, legacy signatures
SHA-2 (SHA-224 / 256 / 384 / 512)	224–512	✅ Secure	128 bits+	Widely used, current standard
SHA-3 (Keccak)	224–512	✅ Secure	128 bits+	New NIST standard, strong against known attacks

Message Authentication Codes (MACs)

1. What is a MAC?
A **Message Authentication Code (MAC)** is a short **authentication tag** appended to a message to ensure:

- **Integrity** → Message hasn't been modified.
- **Authentication** → Message came from a legitimate sender.

Uses **symmetric key cryptography** - both sender and receiver share a **secret key** k.

2. Basic Principle

Step	Sender (Alice)	Receiver (Bob)
1	Compute m = MAC$_k$(x)	Recompute m' = MAC$_k$(x)
2	Send (x, m)	Compare m' with received m
✅		If m' = m → message is authentic

- **Computation:** m = MACk(x)

3. Properties of MACs

Property	Description
1. Cryptographic checksum	MAC is a secure authentication tag for message x
2. Symmetric	Same key for generation & verification.
3. Arbitrary message size	Can handle variable-length input messages.
4. Fixed output length	Output tag size is constant (e.g., 128 or 256 bits).
5. Integrity	Detects any message modification.
6. Authentication	Verifies sender's legitimacy.
7. No non-repudiation	Because both parties share the same key.

4. Implementations of MACs
There are **two main constructions:**
1. **Using Hash Functions → HMAC**
2. **Using Block Ciphers → CBC-MAC, GMAC**

5. MACs from Hash Functions

Simple Constructions
Type

- **Secret Prefix MAC**
- **Secret Suffix MAC**

Problems:

- Attackers can append data without knowing the key.
- Collisions in hash functions break security.

✅ **Solution:**
Use **nested hashing → HMAC**

6. HMAC (Hash-based Message Authentication Code)
Proposed by: Mihir Bellare, Ran Canetti, Hugo Krawczyk (1996)
Construction
HMACk(x) = h[(k+ ⊕ opad) || h[(k+ ⊕ ipad) || x]]
where:

- k+ → key padded to block length
- ipad = 0x36 repeated
- opad = 0x5C repeated
- ⊕ = XOR operation
- h() = cryptographic hash function (e.g., SHA-1, SHA-256)

✅ **Properties of HMAC**

Feature	Description
Structure	Inner + outer hash layers
Security	Provably secure (depends on hash collision resistance)
Flexibility	Works with any hash function (SHA-1, SHA-256, etc.)
Usage	Widely used in TLS, IPsec, SSH, HTTPS

7. MACs from Block Ciphers
Use **block ciphers** (like AES) instead of hash functions.

CBC-MAC (Cipher Block Chaining MAC)
Generation:
Divide message into blocks x1, x2, …, xn
y1 = Ek(x1 ⊕ IV)
yi = Ek(xi ⊕ yi−1) for i = 2..n
MAC = yn
Verification:

- Recompute MAC'
- If MAC' = MAC → valid message
- Else → tampered message ❌

✅ **Properties:**

- Based on AES (symmetric block cipher)
- Secure for **fixed-length** messages
- Used in financial transactions, embedded systems, etc.

GMAC (Galois Message Authentication Code)
Based on: AES in **Galois/Counter Mode (GCM)**
Purpose: Authentication only (no encryption)

Property	Description
Parallelizable	Suitable for high-speed (10 Gbit/sec) systems
Applications	Used in IPsec ESP & AH
Advantage	Very efficient, supports hardware acceleration

8. Comparison Summary

Type	Based On	Key Type	Non-Repudiation	Speed	Use Case
Digital Signature	Asymmetric (Public Key)	Public / Private	✅ Yes	❌ Slower	Email signing, legal docs
MAC (General)	Symmetric	Shared Secret	❌ No	✅ Faster	Network protocols
HMAC	Hash function	Symmetric	❌ No	✅ Fast & Secure	TLS, IPsec, APIs
CBC-MAC	Block cipher	Symmetric	❌ No	✅ Fast	Fixed-length messages
GMAC	AES-GCM	Symmetric	❌ No	⚡ Very Fast	IPsec, VPNs, High-speed links

9. Lessons Learned

- MACs provide **Message Integrity** ✅ and **Authentication** ✅
- MACs are **faster** than digital signatures because they use symmetric keys.
- MACs **do not** provide **non-repudiation** ❌.
- Implementations:
  HMAC → Built from hash functions (secure, widely used)
  CBC-MAC / GMAC → Built from block ciphers (efficient for hardware)

HMAC → Used in **TLS, IPsec, HTTPS, OAuth**
GMAC → Used in **IPsec ESP/AH**, **high-speed networks**

Key Establishment
Goal: Securely establish a shared key between communicating parties.

2. Key Establishment Classification

- **Key Agreement:** Both parties contribute to key generation (e.g., DHKE).
- **Key Transport:** One party securely generates and sends the key.

In an ideal key agreement protocol, no single party controls the key value.

3. Key Freshness
Why frequently change keys?

- Limits damage if a key is exposed.
- Makes cryptanalytic attacks harder (less ciphertext per key).
- Forces attackers to recover multiple keys instead of one.

4. Key Derivation
Goal: Generate new session keys efficiently.

- Derive session keys k_{ses} from master key k_{AB} using nonce r:
  $$k_{ses} = e_{k_{AB}}(r)$$
- Achieved via **hash or block cipher–based key derivation function (KDF)**.
- Provides key freshness without repeating full key establishment.

5. The n² Key Distribution Problem
For n users:

- Need $n(n-1)/2$key pairs → scales poorly.
- Example: 750 users → 561,750 keys!
- ❌ Not practical for large or dynamic networks.

6. Key Distribution Center (KDC)
How it works:

- Central trusted authority shares a long-term **Key Encryption Key (KEK)** with each user.
- Generates a **session key** for each communication pair.

✅ **Advantages**

- Only **n KEKs** (not n²)
- Scales better
- New user only needs key with KDC

⚠️ **Disadvantages**

- ❌ No Perfect Forward Secrecy (if KEK leaked → past data exposed)
- ❌ Single point of failure (KDC database)
- ❌ Communication bottleneck (KDC must be involved each time)

7. Kerberos Protocol

- Based on KDC
- Adds **authentication** + **key confirmation**
- Uses **timestamps** and **lifetime (T)** to prevent replay attacks
- Requires **synchronized clocks**

8. Asymmetric Key Establishment

Diffie–Hellman Key Exchange (DHKE)

- Public parameters: p, α
- Alice: $A = \alpha^a \bmod p$
- Bob: $B = \alpha^b \bmod p$
- Shared key: $k_{AB} = \alpha^{ab} \bmod p$

✅ Secure against passive attacks (if $p > 2^{1024}$).

9. Man-in-the-Middle (MitM) Attack

- Attacker intercepts and replaces public keys.
- Establishes two independent keys with Alice and Bob.
- Reads, modifies, or forges messages.
- Works on **any public-key scheme** (RSA, ECC, etc.).

Root Cause:
- Public keys are **not authenticated**.

10. Certificates
Purpose: Authenticate public keys → prevent MitM.
Issued by: Certificate Authority (CA).
Structure:
$$Cert(Alice) = (k_{pub}, ID(Alice), sig_{k_{CA}}(k_{pub}, ID(Alice)))$$

- Binds **identity → public key**
- Signed with CA's **private key**
- Verified using CA's **public key**

⚠️ The CA's **public key must itself be distributed via an authenticated channel!**

11. Diffie–Hellman with Certificates

- Alice & Bob verify each other's certificates using CA's public key.
- Only need to **trust CA once** (browser example).

✅ Prevents MitM if certificates are verified.

12. Public Key Infrastructure (PKI)
Definition:
System of CAs + support mechanisms for certificate management.
Real-World Standard: X.509 Certificates
Includes fields:

- Version, Serial Number
- Issuer Name
- Validity Period
- Subject Name
- Subject Public Key
- Signature Algorithm + Signature

13. PKI Issues
1.Cross-Certification (Chain of Trust):

- CA_1 certifies CA_2's public key.
- Enables trust delegation between organizations.

2.Certificate Revocation Lists (CRLs):

- Used when certificates are compromised or expired.
- Must be distributed regularly → bandwidth overhead.

✅ **Summary Table**

Concept	Purpose / Function	Notes / Risks
Key Freshness	Limits damage from key compromise	Rotate keys regularly
Key Derivation	Generate new session keys efficiently	Uses nonce + KDF
KDC	Centralized key distribution	Single point of failure
Kerberos	Authentication + replay protection	Timestamp-based
DHKE	Public-key key exchange	Vulnerable to MitM
Certificates	Authenticate public keys	Issued by CA
PKI	Manage certificates & trust	Uses X.509 format
CRL	Revoke compromised certs	High network load

Steganography – The Art of Covert Communication

Definition
The practice of **hiding secret data** within an **innocent-looking cover medium** (image, audio, video, text, etc.) so that the existence of the data is **undetectable**.

Steganography vs. Encryption

Aspect	Encryption	Steganography
Goal	Make data *unintelligible*	Make data *undetectable*
Visibility	Data is visibly scrambled	Data appears normal
Usage	Secure transmission	Covert communication
Can be combined?	✅ Yes — used together for better secrecy	

Etymology

- **Greek origin:**
  -- *Steganos* → "Covered/Protected"
  -- *Graphei* → "Writing"

→ **"Covered Writing"**
First recorded by *Johannes Trithemius (1499)*

Historical Examples

- Invisible ink
- Hidden words in text
- Wax tablets with secret layers
- Messages tattooed on scalps (hidden by hair)

Modern Uses
- **Legitimate:** DRM, copyright, watermarking
- **Illegitimate:** Espionage, crime, activism, malware
- Usage increasing → proliferation of data-hiding tools

Key Terms

Term	Meaning
Cover Object	The innocent carrier (image, packet, file, etc.)
Secret	The hidden data (text, image, etc.)
Carrier Object	Cover + Secret = *Stego Object*
Stego-Key	Key/method used to extract the secret

The Steganographic Triangle
1.Invisibility → Undetectable by humans/steganalysis
2Capacity → Amount of secret data that can be hidden
3Robustness → Resistance to transformations (cropping, conversion, etc.)
These three factors must be balanced for effective steganography.

Classification of Techniques
1.Substitution – Replace insignificant bits (e.g., LSB)
2.Transform Domain – Hide data in frequency components (e.g., DCT, DWT)
3.Spread Spectrum – Spread secret data over wide bandwidth
4.Statistical – Modify statistical properties
5.Cover Generation – Create synthetic covers with embedded secrets
6.Distortion – Introduce small distortions interpreted as data

LSB (Least Significant Bit) Embedding
Concept: Embed bits of secret data into the **least significant bits** of cover object bytes (often image pixels).
Byte: 01001100 → LSB changed to 01001101
Ex: Each pixel = 3 bytes (RGB)
Modify the LSB of each color byte to store message bits

Advantages	Limitations
Simple & easy to implement	Low robustness
Good capacity	Easily detected by steganalysis
Exploits human visual system (HVS) weakness	Alterations (compression, editing) can destroy data

Steganography Tools

Type	Tool	Use
Image LSB	**STEG-J**	Hide data in images
Text files	**SNOW**	Hide data using whitespace
Audio/Video	**STEGHIDE**	Embed in audio/video
Distributed Stego	—	Spread payload across multiple files for higher secrecy

RMD PRAKASH

🔹 **Key Takeaways**

- Steganography hides data **within normal-looking content**.
- Provides **confidentiality through concealment** (not encryption).
- **Balance** among invisibility, capacity, and robustness is essential.
- **LSB** is the most common and simplest method.
- Increasingly combined with **encryption** for stronger security.

Key Establishment & Distribution

Key Freshness:

- Each session/key should be unique to prevent replay attacks.
- Often ensured using **nonces** or **timestamps**.

Man-in-the-Middle Attack (MITM)

- Occurs when attacker intercepts key exchange.
- Example: intercepts Diffie-Hellman exchange → establishes separate keys with both parties.

Countermeasures: Use digital certificates. Authenticate public keys via **PKI**.

Certificates & Public Key Infrastructure (PKI)

- **Certificate:** Digital document binding public key → entity identity, signed by CA.
- **CA (Certificate Authority):** Trusted entity that signs certificates.
- **PKI:** Framework to manage keys, certificates, trust hierarchy.

RMPURAJAPAKSHA