SCHEME OF INSTRUCTION AND EXAMINATION
B. E – Honors

Code	Course Title	Scheme of Instruction			Contact Hrs/Wk	Scheme of Examination			Credits	**To be offered Semester**
		L	**T**	**P**		**Hrs**	**CIE**	**SEE**		
Theory										
HR501CS	Fundamentals of Data Engineering	3	0	-	3	3	40	60	3	**V**
HR601CS	Complexity Theory	3	0	-	3	3	40	60	3	**VI**
HR602CS	Formal Methods and Program Analysis	3	0	-	3	3	40	60	3	**VI**
HR701CS	Advanced Operating Systems	3	0	-	3	3	40	60	3	**VII**
HR702CS	Systems Design	3	0	-	3	3	40	60	3	**VII**
HR801CS	Applied Cryptography	3	0	-	3	3	40	60	3	**VIII**
		18	**0**	**-**	**18**	**18**	**240**	**360**	**18**	

HR501CS	**Fundamentals of Data Engineering**					
Prerequisites	Database Management Systems		**L**	**T**	**P**	**C**
			3	**0**	**0**	**3**
Evaluation	**CIE**	**40 Marks**	**SEE**		**60 Marks**	

Course Objectives:	
1.	Understand the Role of Data Engineering in Modern Organizations
2.	Design Scalable and Reliable Data Architectures
3.	Select and Implement Appropriate Technologies Across the Data Engineering Lifecycle
4.	Optimize Data Storage and Ingestion Techniques

Course Outcomes - At the end of the course students should be able to	
1.	Demonstrate a comprehensive understanding of the data engineering lifecycle, encompassing essential skills, activities, and the roles of a data engineer in modern business settings.
2.	Design and assess scalable and reliable data architectures that integrate modern data platforms like data lakes and warehouses.
3.	Utilize decision-making skills to select optimal technologies and frameworks based on team capabilities, performance, cost, and interoperability across the data engineering lifecycle.
4.	Implement effective data storage solutions by applying knowledge of distributed storage, partitioning, data retention, and modern storage architectures such as data lakehouses and stream-to-batch systems.
5.	Build and manage data ingestion pipelines that handle both bounded and unbounded data while ensuring scalability, data integrity, and reliability through techniques like ETL, change data capture, and message queues.

Unit-1

Data Engineering Described : Definition, Lifecycle, Evolution of the Data Engineer, Data Engineering and Data Science , Data Engineering Skills and Activities, Data Maturity and the Data Engineer, The Background and Skills of a Data Engineer, Business Responsibilities , Technical Responsibilities, The Continuum of Data Engineering Roles, from A to B , Data Engineers Inside an Organization, Internal-Facing Versus External-Facing Data Engineers, Data Engineers and Other Technical Roles, Data Engineers and Business Leadership.

The Data Engineering Lifecycle: The Data Lifecycle Versus the Data Engineering Lifecycle, Generation: Source Systems, Storage, Ingestion, Transformation, Serving Data, Major Undercurrents Across the Data Engineering Lifecycle, Security, Data Management, DataOps, Data Architecture, Orchestration, Software Engineering.

Unit- 2

Designing Good Data Architecture: What Is Data Architecture, Enterprise Architecture Defined, Data Architecture Defined, "Good" Data Architecture, Principles of Good Data Architecture, Major Architecture Concepts.

Domains and Services, Distributed Systems, Scalability, and Designing for Failure, Tight Versus Loose Coupling: Tiers, Monoliths, and Microservices, **User Access**: Single Versus Multitenant, Event-Driven Architecture,

Brownfield Versus Greenfield Projects, Examples and Types of Data Architecture, Data Warehouse, Data Lake, Convergence, Next-Generation Data Lakes, and the Data Platform, Modern Data Stack, Lambda Architecture, Kappa Architecture, The Dataflow Model and Unified Batch and Streaming, Architecture for IoT, Data Mesh, Other Data Architecture Examples, Who's Involved with Designing a Data Architecture.

Unit- 3

Choosing Technologies Across the Data Engineering Lifecycle: Team Size and Capabilities, Speed to Market, Interoperability, Cost Optimization and Business Value, Total Cost of Ownership, Total Opportunity Cost of Ownership, FinOps, Optimization, Performance, and the Benchmark Wars.

Data Generation in Source Systems: Sources of Data: How Is Data Created? Source Systems: Main Ideas, Files and Unstructured Data, APIs, Application Databases (OLTP Systems), Online Analytical Processing System, Change Data Capture, Logs, Database Logs, CRUD, Insert-Only.

Messages and Streams, Types of Time, Source System Practical Details, Databases, APIs, Data Sharing, Third-Party Data Sources, Message Queues and Event-Streaming Platforms, Undercurrents and Their Impact on Source Systems, Security, Data Management, DataOps, Data Architecture, Orchestration, Software Engineering.

Unit -4

Storage: Data Storage Systems, Single Machine Versus Distributed Storage, Eventual Versus Strong Consistency, Cache and Memory-Based Storage Systems, The Hadoop Distributed File System, Streaming Storage, Indexes, Partitioning, and Clustering, Data Engineering Storage Abstractions, The Data Warehouse, The Data Lake, The Data Lakehouse, Data Platforms.

Stream-to-Batch Storage Architecture, Big Ideas and Trends in Storage, Data Catalog, Data Sharing, Schema, Separation of Compute from Storage, Data Storage Lifecycle and Data Retention, Single-Tenant Versus Multitenant Storage

Unit- 5
Ingestion: What Is Data Ingestion, Key Engineering Considerations for the Ingestion Phase, Bounded Versus Unbounded Data, Frequency, Synchronous Versus Asynchronous Ingestion, Serialization and Deserialization, Throughput and Scalability, Reliability and Durability, Payload, Push Versus Pull Versus Poll Patterns, Batch Ingestion Considerations, Snapshot or Differential Extraction, File-Based Export and Ingestion, ETL Versus ELT Inserts, Updates, and Batch Size, Data Migration, Message and Stream Ingestion Considerations, Schema Evolution, Late-Arriving Data, Ordering and Multiple Delivery, Replay, Time to Live, Message Size, Error Handling and Dead-Letter Queues, Consumer Pull and Push, Location, Ways to Ingest Data, Direct Database Connection, Change Data Capture, APIs, Message Queues and Event-Streaming Platforms, Managed Data Connectors, Moving Data with Object Storage, EDI, Databases and File Export.

Suggested Readings:
1. Joe Reis and Matt Housley, *Fundamentals of Data Engineering Plan and Build Robust Data Systems*, Published by O'Reilly Media, Inc., July 2022: First Edition.
2. Paul Crickard, *Data Engineering with Python: Work with massive datasets to design data models and automate data pipelines using Python*, Kindle Edition Packt Publishers.

HR601CS	**Complexity Theory**					
Prerequisites	Discrete Mathematics		**L**	**T**	**P**	**C**
			3	**0**	**0**	**3**
Evaluation	**CIE**	**40 Marks**	**SEE**		**60 Marks**	

Course Objectives	
1.	The aim is to help the student to understand the modern and advanced concepts in computational complexity theory.
2.	The course will explain measures of the complexity of problems and of algorithms, based on time and space used on abstract models.
3.	Important complexity classes will be defined, and the notion of completeness established through a thorough study of NP-completeness.
4.	Applications to cryptography will be considered.

Course Outcomes - At the end of the course students should be able to	
1.	Analyse practical problems and classify them according to their complexity;
2.	Familiar with the phenomenon of NP-completeness, and be able to identify problems that are NP-complete;
3.	Aware of a variety of complexity classes and their interrelationships;
4.	Understand the role of complexity theory in cryptography and quantum computing

Unit- 1
Algorithms and problems. Complexity of algorithms and of problems. Lower and upper bounds. Examples: sorting and travelling salesman. Time and space. Models of computation and measures of complexity. Time and space complexity on a Turing machine. Decidability and complexity

.

Unit -2
Time complexity. Time complexity classes. Polynomial time problems and algorithms. Problems on numbers, graphs and formulas. Non-determinism. Non-deterministic machines. The complexity class NP and its various characterizations. Non-deterministic algorithms for satisfiability and other problems in NP.

Unit -3
NP-completeness. Reductions and completeness. NP-completeness of satisfiability. More NP-complete problems. Graph-theoretic problems. Independent set, clique and 3-colourability. More NP-complete problems. Sets, numbers and scheduling. Matching, set covering and knapsack.

Unit- 4
coNP. Validity of boolean formulae and its completeness. NP ∩ coNP. Primality and factorisation. Cryptographic complexity. One-way functions. The class UP.

Unit -5
Space complexity. Deterministic and non-deterministic space complexity classes. The reachability method. Savitch's theorem. Hierarchy. The time and space hierarchy theorems and complete problems. Quantum Complexity. The classes BQP and QMA

.

Suggested Reading:	
1.	*Computational Complexity: A Modern Approach* by Arora and Barak
2.	*Computational Complexity: A Conceptual Perspective* by Oded Goldreich
3.	*Mathematics and Computation* by Avi Wigderson

HR602CS	Formal Methods and Program Analysis					
Prerequisites	Theory of Computation		**L**	**T**	**P**	**C**
			3	**0**	**0**	**3**
Evaluation	**CIE**	**40 Marks**	**SEE**		**60 Marks**	

Course Objectives	
1.	Understand and apply formal logic and proof techniques to reason about the correctness and properties of software systems, including methods for verification and validation.
2.	Analyze and model computational systems using formal methods such as automata theory, state machines, and process algebra to ensure system reliability and correctness.
3.	Implement and evaluate software verification techniques using formal specification languages and model checking to detect and address potential errors in software systems.
4.	Explore and apply program analysis techniques including static and dynamic analysis, data flow analysis, and symbolic execution to improve software quality and performance.

Course Outcomes - At the end of the course students should be able to	
1.	Explain the basic concepts of formal methods and logic, and use formal proof techniques to reason about software correctness.
2.	Develop formal specifications for software systems and use verification techniques to ensure these systems meet their specifications.
3.	Apply model checking tools to automatically verify the correctness of software systems and detect potential errors.
4.	Perform static and dynamic analysis on software to identify potential issues and optimize performance.
5.	Explore and apply advanced formal methods and program analysis techniques to complex systems, including security and reliability concerns.

Unit-1

Overview of Formal Methods: Definitions, importance, and applications in software engineering.
Mathematical Foundations: Logic, set theory, and proof techniques.
Specification Languages: Introduction to specification languages such as Z, B, and VDM.
Formal Verification: Basics of formal verification, theorem proving, and model checking.
Case Studies: Real-world examples where formal methods have been successfully applied.

Unit-2

Propositional Logic: Syntax, semantics, and inference rules.
Predicate Logic: Quantifiers, theories, and inference rules.
Proof Techniques: Direct proof, proof by contradiction, induction, and the use of automated theorem provers.
Formal Proof Systems: Natural deduction, sequent calculus, and tableaux methods.
Applications in Program Verification: Using logic for reasoning about program

correctness.

Unit-3
Finite State Machines (FSMs): Definition, types, and state transition diagrams. **Petri Nets:** Basic concepts, modeling, and analysis. **Automata Theory:** Deterministic and non-deterministic automata, and their applications in verification. **Temporal Logic:** Linear Temporal Logic (LTL) and Computational Tree Logic (CTL). **Model Checking:** Techniques and tools for verifying finite-state systems.

Unit-4
Static Analysis: Techniques for analyzing code without executing it, including abstract interpretation and data flow analysis. **Dynamic Analysis:** Techniques for analyzing code during execution, including runtime monitoring and debugging. **Formal Verification of Programs:** Using formal methods to prove the correctness of algorithms and software. **Symbolic Execution:** Techniques for exploring program paths and detecting errors. **Applications and Tools:** Introduction to tools and frameworks for program analysis and verification.

Unit-5
Concurrency and Parallelism: Formal methods for reasoning about concurrent and parallel systems. **Security and Privacy:** Using formal methods to analyze and ensure software security and privacy. **Hybrid Systems:** Formal methods for systems with both discrete and continuous components. **Quantitative Verification:** Techniques for analyzing systems with quantitative properties like performance and reliability. **Emerging Trends:** Recent developments and future directions in formal methods and program analysis.

Suggested Reading:
1. C. A. R. Hoare and He Jifeng, Formal Methods: An Introduction, 1st Edition, Prentice Hall
2. Michael Sipser, Introduction to the Theory of Computation, 3rd Edition, Cengage Learning
3. N. Shroff and R. D. D. Souza, Software Verification and Analysis: Engineering for Qualities, 1st Edition, Springer
4. Christel Baier and Joost-Pieter Katoen, Principles of Model Checking, 1st Edition, MIT Press

HR701CS	Advanced Operating Systems					
Prerequisites	Operating Systems		**L**	**T**	**P**	**C**
			3	**0**	**0**	**3**
Evaluation	**CIE**	**40 Marks**	**SEE**		**60 Marks**	

Course Objectives	
1.	Understand global view of distributed operating systems and provides theoretical foundation for distributed systems.
2.	Study the characteristics of OS for Multiprocessor and Multicomputer.
3.	Learn the issues related to designing OS.
4.	Understand Security & protection in computer systems and mechanisms used in building multiprocessor operating systems.
5.	Explore management of different resources in distributed systems.

Course Outcomes - At the end of the course students should be able to	
1.	Understand the concept of distributed system and foundations.
2.	Familiarize with advanced paradigms, architectures & protocols necessary in solve the challenges in design of advanced operating systems.
3.	Analysis of efficiency and proofs of correctness for multiple aspects in design of Advanced Operating Systems

Unit-1

Architecture of Distributed Systems: Types, Distributed Operating System, Issues in Distributed Operating Systems, Theoretical Foundations: Global Clock, Lamport's Logical Clock, Vector Clocks, Global State, and Termination Detection.

Unit -2

Distributed Mutual Exclusion: Classification, requirement, performance, non-token-based algorithms, Lamport's algorithm, the Richart-Agarwala algorithm, token-based algorithm Suzuki liasamil's broadcast algorithm, Singhals heuristic algorithm.

Deadlock Detection: Resource Vs Communication deadlock, A graph- theoretic model, prevention, avoidance, detection, control organization, centralized deadlock-detection algorithm, the completely centralized algorithm, the HO-Ramamoorthy algorithm. Distributed deadlock detection algorithm - path - pushing, edge-chasing, hierarchical deadlock detection algorithm, menace-muntz and Ho-Ramamoorthy algorithm. Agreement Protocols: The system model, the Byzantine agreement, and the consensus problem.

Unit -3

Distributed File System: Mechanisms, Design Issues - Andrew File System. Design and implementation of a log structured file system.
Distributed Shared Memory: Algorithms for Implementing DSM, Memory Coherence, Coherence Protocols, Design Issues.
Distributed Scheduling: Issues in Load Distribution, Components of Algorithm, Stability

Load Distributing Algorithm, Performance.

Unit -4
Failure Recovery: Backward, Forward Error Recovery in Concurrent Systems, Consistent Set of Check Points, Synchronous and Asynchronous Check Pointing and Recovery. **Fault Tolerance:** Commit Protocols, Non-Blocking Commit Protocols, Voting Protocols. **Protection and Security:** Access Matrix, Private Key, Public key, and Kerberos System.

Unit -5
Multiprocessor Operating Systems: Motivation, Basic Multiprocessor System Architecture, Interconnection Networks for Multiprocessor Systems, Caching, Hypercube Architecture. Threads, Process Synchronization, Processor Scheduling, and Memory Management. **Database Operating System:** Concurrence Control, Distributed Databases, and Concurrency Control Algorithms.

Suggested Reading:
1. Singhal M, Shivaratri N.G, *Advanced Concepts in Operating Systems*, McGraw-Hill Intl., 1994.
2. Pradeep K Sinha, *Distributed Operating Systems Concepts and Design*, PHI, First Edition, 2002.
3. Andrew S. Tanenbaum, *Distributed Operating Systems*, Pearson Education India, First Edition, 2011.

HR702CS	**Systems Design**					
Prerequisites	Data Structures and Algorithms		**L**	**T**	**P**	**C**
			3	**0**	**0**	**3**
Evaluation	**CIE**	**40 Marks**	**SEE**		**60 Marks**	

Course Objectives	
1.	Develop a comprehensive understanding of system design principles to create scalable, reliable, and maintainable architectures for modern computing environments.
2.	Apply design patterns and architectural strategies to address complex system requirements, including fault tolerance, high availability, and performance optimization.
3.	Analyze and evaluate various system components such as databases, distributed systems, and cloud infrastructure, to make informed decisions on technology selection and system integration.
4.	Design and implement effective solutions for real-world problems in system design, incorporating best practices in scalability, security, and performance.

Course Outcomes - At the end of the course students should be able to	
1.	Understand the core principles and methodologies of system design to create scalable and efficient architectures.
2.	Apply design patterns and principles to solve common system design problems and improve code maintainability.
3.	Design and manage relational and non-relational databases to optimize performance and ensure data integrity.
4.	Develop strategies for building and managing distributed systems to enhance scalability and fault tolerance.
5.	Implement cloud-based solutions and modern architectural patterns to address current challenges in system design and deployment.

Unit-1

Overview of System Design: Importance, goals, and challenges in designing scalable and efficient systems.

Design Principles: SOLID Principles, Modularization, abstraction, separation of concerns, and design trade-offs.

System Architecture: Client-server architecture, microservices, service-oriented architecture (SOA), and monolithic systems.

Design Patterns: Common design patterns such as Singleton, Factory, Observer, and MVC.

Software Development Lifecycle: Requirements gathering, design, development, and testing.

Case Studies: Design analysis of popular systems like Facebook, Google, and Amazon.

Unit-2
Scalability Concepts: Horizontal vs. vertical scaling, load balancing, partitioning, and sharding. **Performance Optimization:** Latency, throughput, and optimizing response time. **Caching Strategies:** Client-side, server-side, distributed caching, and cache invalidation policies. **Database Scalability:** NoSQL vs. SQL databases, database partitioning, and indexing. **Message Queuing and Event-Driven Architecture:** Use of message brokers (RabbitMQ, Kafka) for decoupling components. **Content Delivery Networks (CDN):** Using CDNs for optimizing content distribution.

Unit-3
Introduction to Distributed Systems: Characteristics and challenges, CAP theorem, consistency models. **Data Replication and Synchronization:** Replication strategies, eventual consistency, and conflict resolution. **Consensus Algorithms:** Paxos, Raft, and Byzantine Fault Tolerance. **Distributed Databases:** Understanding distributed databases (Cassandra, MongoDB), and their trade-offs. **Fault Tolerance and High Availability:** Techniques for achieving high availability, failover, and redundancy. **Case Studies:** Distributed systems design in companies like Google (Bigtable, Spanner) and Amazon (DynamoDB).

Unit-4
Security Principles in System Design: Confidentiality, integrity, availability, and authentication methods. **Data Security:** Encryption (AES, RSA), data privacy, and handling sensitive information. **Authentication and Authorization:** OAuth, JWT, SSO, and RBAC (Role-Based Access Control). **Reliability and Redundancy:** Designing reliable systems, backup strategies, disaster recovery, and replication. **Monitoring and Observability:** Log management, distributed tracing, and health checks. **Security Design Case Studies:** Security challenges and solutions in real-world systems (e.g., Payment gateways, secure messaging apps).

Unit-5
Real-World System Design Case Studies: Design analysis of large-scale systems like Netflix, Uber, and Twitter. **Advanced Design Patterns:** Event sourcing, CQRS, Circuit Breaker, and Bulkhead patterns. **Designing for DevOps:** Continuous integration (CI), continuous deployment (CD), infrastructure as code, and containerization (Docker, Kubernetes). **Cloud-Native System Design:** Designing systems for cloud platforms (AWS, Azure, GCP), serverless architectures. **Emerging Trends:** Edge computing, 5G, and the impact on system design. **Ethical and Sustainable System Design:** Designing energy-efficient systems and understanding ethical considerations in system design.

Suggested Reading:
1. Clean Architecture: A Craftsman's Guide to Software Structure and Design Robert C. Martin, 2017
2. Martin Kleppmann, Designing Data-Intensive Applications, 1st Edition, O'Reilly Media
3. Martin L. Abbott, Michael T. Fisher, The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise, 2nd Edition, Addison-Wesley Professional
4. Martin Fowler, Patterns of Enterprise Application Architecture, 1st Edition, Addison-Wesley Professional
5. Niall Richard Murphy, Betsy Beyer, Chris Jones, and Jennifer Petoff, Site Reliability Engineering: How Google Runs Production Systems, 1st Edition, O'Reilly Media

HR801CS	**Applied Cryptography**					
Prerequisites	Discrete Mathematics		**L**	**T**	**P**	**C**
			3	**0**	**0**	**3**
Evaluation	**CIE**	**40 Marks**	**SEE**		**60 Marks**	

Course Objectives	
1.	Develop a strong understanding of cryptographic principles and their mathematical foundations
2.	Master symmetric and asymmetric cryptography methods
3.	Learn to apply cryptographic hash functions and authentication protocols
4.	Explore advanced cryptography topics and emerging trends

Course Outcomes - At the end of the course students should be able to	
1.	Understand cryptography's principles, history, and foundational mathematics used in modern encryption.
2.	Analyze and implement symmetric encryption methods, considering operational modes and security practices.
3.	Gain knowledge of public key cryptography, including key exchange and digital signatures for secure communication.
4.	Apply hash functions and authentication techniques to ensure data integrity and secure verification.
5.	Explore advanced cryptographic methods, including post-quantum encryption, blockchain, and IoT security solutions.

Unit-1

Introduction to Cryptography: History and evolution of cryptography, Cryptographic goals: Confidentiality, Integrity, Authentication, Non-repudiation, Types of cryptography: Symmetric and Asymmetric Cryptography, Cryptographic protocols and their applications

Classical Cryptography: Caesar Cipher, Vigenère Cipher, Playfair Cipher, Substitution and Transposition Techniques, Limitations of classical cryptography

Mathematical Foundations: Modular arithmetic, GCD, Fermat's Theorem, Euler's Theorem, Primes, Factorization, and Discrete Logarithms, Finite fields and algebraic structures used in cryptography

Unit-2
Block Ciphers: DES (Data Encryption Standard): Structure, Strengths, and Weaknesses, Triple DES, Advanced Encryption Standard (AES), Modes of operation: ECB, CBC, CFB, OFB, and CTR **Stream Ciphers:**RC4, Salsa20, and ChaCha stream ciphers, Overview of LFSRs (Linear Feedback Shift Registers) in stream ciphers **Symmetric Key Algorithms and Security:** Key Distribution and Management, Random Number Generators and their significance in cryptography, Attacks on symmetric cryptosystems: Brute Force, Differential, and Linear Cryptanalysis.

Unit-3
Introduction to Public Key Cryptography: Principles of asymmetric encryption, RSA Algorithm: Key generation, encryption, and decryption, Diffie-Hellman Key Exchange Protocol **Elliptic Curve Cryptography (ECC):** Basics of elliptic curves and their application in cryptography, Elliptic Curve Digital Signature Algorithm (ECDSA) **Digital Signatures and Certificates:** Digital signatures: DSA (Digital Signature Algorithm), RSA-based signatures, Certificate Authorities (CA) and Public Key Infrastructure (PKI), X.509 certificates and certificate validation.

Unit-4
Hash Functions: Properties of cryptographic hash functions: Collision resistance, Pre-image resistance, and Second pre-image resistance, MD5, SHA-1, SHA-256, SHA-3 algorithms, HMAC (Hashed Message Authentication Code) **Message Integrity and Authentication:** MAC (Message Authentication Codes) and CMAC (Cipher-based MAC), Authentication protocols: Challenge-Response, Zero-Knowledge Proofs **Password-based Cryptography:** Salted Hashes, Key Stretching (PBKDF2, bcrypt, Argon2), Password Authentication Mechanisms.

Unit-5
Post-Quantum Cryptography: Threats posed by quantum computing to classical cryptosystems, Overview of quantum-resistant algorithms: Lattice-based, Hash-based, Multivariate polynomial cryptography **Blockchain and Cryptography:** Cryptographic concepts in blockchain: Hashing, Digital signatures, Merkle trees, Bitcoin and Ethereum cryptographic foundations **Cryptanalysis and Security Protocols:** Side-channel attacks: Timing attacks, Power analysis, Attacks on cryptographic algorithms: Man-in-the-Middle, Replay, and Chosen Ciphertext Attacks **Current Trends and Applications:** Homomorphic encryption, Secure Multi-Party Computation (SMPC), Zero-Knowledge Proofs in privacy-preserving applications, Lightweight cryptography for IoT devices

Suggested Reading:
1. Cryptography and Network Security: Principles and Practice by William Stallings, 7th Edition, Pearson Pub.
2. Applied Cryptography: Protocols, Algorithms, and Source Code in C by Bruce Schneier, 2nd Edition, Wiley Pub.
3. Introduction to Modern Cryptography: Principles and Protocols by Jonathan Katz and Yehuda Lindell, 2nd Edition, CRC Press
4. Understanding Cryptography: A Textbook for Students and Practitioners by Christof Paar and Jan Pelzl, 1st Edition, Springer