

INFORMATION SECURITY

Principles, Cryptography, Cyber Defence,
Practical Implementation and Real-World Security

AUTHOR

Professor Muhammad Umar Naheed

COURSE CODE: CSIS2022

PROGRAMME: Associate Degree Programme in Computer Science (ADP-CS)

Third Semester | 2 + 1 (3) Credit Hours | 16 Weeks, 32 Sessions

INFORMATION SECURITY

Principles, Cryptography, Cyber Defence, Practical Implementation and Real-World Security

A university-level textbook with practical laboratories, visual learning, real-world applications and examination preparation

Professor Muhammad Umar Naheed

Department of Computer Science

Course Code: CSIS2022

Associate Degree Programme in Computer Science (ADP-CS)

Third Semester | Credit Hours 2 + 1 (3) | 16 Weeks, 32 Sessions

Understand the Concept. Visualise the Process. Practise the Technique. Apply the Knowledge. Remember It for Life.

Copyright and Publication Information

Information Security: Principles, Cryptography, Cyber Defence, Practical Implementation and Real-World Security

Author: Professor Muhammad Umar Naheed. Course: CSIS2022 Information Security, Associate Degree Programme in Computer Science (ADP-CS), Third Semester. Language: English.

Copyright © Professor Muhammad Umar Naheed. All rights reserved. This book is an original educational work. No part may be reproduced or distributed without the author's permission, except for brief quotation with attribution for teaching, review or scholarship.

Publication status

This edition carries no ISBN, publisher imprint, university endorsement or official publication approval. It is a course textbook prepared for classroom use.

Educational use and safety

All practical exercises must be performed only on systems you own or are explicitly authorised to test, preferably in local virtual machines or isolated laboratory networks. The author and publisher accept no responsibility for misuse. Legal references are general and are not legal advice.

Verification and lab files

The numerical examples, cryptographic worked examples and Python/C programs in this book were executed and cross-checked while the book was prepared (for example DES against the standard test vector, AES against FIPS 197, RC4 and TOTP against published vectors). Commands that need a database server, a firewall, a graphical analyser or Internet access are clearly marked as reference procedures to be run in your own lab. Please verify bibliographic details (editions, pages) against your library copies. A companion archive of the lab programs accompanies this book.

Dedication

To every student who asks "why?" before "what?", and to everyone who protects other people's information quietly and responsibly.

Author's Preface

Information security is not a subject for specialists alone. Every student, teacher, nurse, shopkeeper and developer now depends on digital information, and every one of them benefits from understanding how that information is protected, how it is attacked and what responsible people do about it.

This book was written for ADP Computer Science students meeting information security for the first time. It assumes curiosity, not prior knowledge. Each topic starts from an everyday idea, introduces the formal definition afterwards, shows how the mechanism works with a diagram, and then lets you practise with a calculation, a program or a checklist.

The book follows the official course structure of CSIS2022 and its sixteen modules. It treats historical ciphers as training for the mind and modern algorithms as tools for real systems, and it is careful to say which is which. Every attack is paired with detection, prevention and mitigation, and every practical is designed for a safe, local environment.

My hope is simple: that you finish a module and say, "Now I understand," and then open a terminal and prove it to yourself.

Professor Muhammad Umar Naheed

Message to Students

Do not memorise definitions you do not understand. Read the simple explanation first, find the everyday analogy, then read the formal definition and notice how it says the same thing precisely.

Do every worked example with a pencil before you run the program. When your answer matches the program, you have learned something; when it does not, you have found the most valuable moment in the lesson.

Security is also about character. The skills in this book can protect a hospital or harm one. Use them only with permission, report what you find responsibly, and treat other people's data as you would want yours treated.

Study rhythm
Concept (read) → Process (study the figure) → Technique (do the worked example) → Application (run the practical) → Remember (revision sheet and questions).

About the Author

Professor Muhammad Umar Naheed is a computer science educator, cybersecurity and technology professional, entrepreneur, and academic content developer whose work focuses on making complex technological concepts understandable, practical and relevant to real-world challenges.

His professional interests span Information Security, Cybersecurity, Security Operations Centre (SOC) concepts, security monitoring, incident response, cloud security, network defence, ethical hacking, secure software development, artificial intelligence, automation and technology education.

He completed his Associate Degree in Computer Science at the University of Central Punjab, Bhalwal, during 2022-2024 with a Gold Medal. He subsequently pursued a BS Information Technology degree through the Post-ADP programme at the University of Education, Lahore, during 2024-2026, where he achieved a top position in his class.

Alongside his studies he has developed an extensive interest in cybersecurity, software engineering and intelligent technology solutions. His professional development includes CompTIA Security+ (SY0-601), the Google Cybersecurity Professional Certificate, Google IT Automation with Python, Cisco cybersecurity-related training and credentials, and AWS security-focused professional training. His professional profile reports more than 80 international-level credentials and learning achievements; these comprise a mixture of certifications, professional certificates, training programmes and course-completion credentials, and are not all equivalent professional certifications.

Academic and teaching experience

Professor Muhammad Umar Naheed is actively involved in computer science and technology education. His teaching and academic activities include work associated with Punjab Group of Colleges, Bhalwal, and university-level teaching responsibilities involving University of Sargodha programmes. His teaching interests include Computer Science, Information Security, cybersecurity fundamentals, Information and Communication Technology, Operating Systems, Software Project Management, digital and professional technology skills, and Artificial Intelligence and automation.

His approach emphasises conceptual clarity, practical demonstrations, student participation and problem-solving. A student should not merely memorise a definition; the student should understand the reason behind it, recognise it in a real situation and apply it independently.

Entrepreneurship and technology ventures

RaarTech (established November 2022): he is Founder and Head of Digital Solutions. Its work is associated with web and mobile application development, cybersecurity-focused solutions, software development, automation, Python-based solutions and digital transformation.

Vertex Institute of Technology & Skills (established January 2025): he is Founder, CEO and Lead Instructor. The institute focuses on practical, career-oriented education in cybersecurity, artificial intelligence, software and web development, programming, automation and professional technology skills, through industry-relevant learning and practical projects.

Cybersecurity and technical projects

CyberGuard360 is a cybersecurity-focused web project developed with the Fusion Minds team using React, TypeScript, Vite, Tailwind CSS and Supabase/Lovable Cloud technologies, including security-related functionality, detection-oriented features, application architecture, testing and security workflow design. **Parco ScanTrack** is a QR-based inventory and tracking Progressive Web Application demonstrating digital tracking and application-based workflows.

Areas of expertise

- Information Security and Cybersecurity; SOC concepts, security monitoring and incident response.
- Network security and defensive security; cloud security including AWS security services.
- Ethical hacking and security awareness; secure software and web application development.
- Python programming and automation; Artificial Intelligence and technology automation.
- Linux-based environments and cybersecurity tools; information systems and access control.
- Technology education and academic content development.

Professional philosophy

Cybersecurity education should not be restricted to complex terminology, theoretical definitions or examination preparation. Students should understand how information is protected, how attacks occur, why vulnerabilities matter, how security mechanisms operate and how responsible professionals identify and address security problems.

His educational objective is simple: make difficult concepts understandable, technical processes practical and knowledge valuable beyond the classroom.

Author's Teaching Philosophy

The whole book follows one principle: **Understand the Concept. Visualise the Process. Practise the Technique. Apply the Knowledge. Remember It for Life.**

Table 55. The five questions answered for every topic

Question	Where it is answered
What is it?	"In simple words" and "Formal definition"
Why does it matter?	Module introduction and real-world relevance
How does it work?	"How it works", figures and worked examples
Where is it used in real life?	Applications, case studies and real-world sections
How can I practise it?	Practical laboratories, questions and revision sheets

How to Use This Textbook

- Read the module introduction, objectives and prerequisites.
- For each topic read the simple explanation and analogy, study the figure, then read the formal definition.
- Work every worked example by hand before running the program.
- Run each practical in a safe environment. Boxes labelled "verified" were executed by the author; boxes labelled "reference" must be run in your own lab.
- Use the revision section and question bank at the end of every module; check answers in the answer-key appendix.
- Use the back matter (formula sheet, command reference, glossary, revision guide) for examination preparation.
- Long code lines are wrapped in print and each continuation line is marked with an arrow (↪). For runnable code, always use the companion lab archive rather than retyping from the page.

Table 56. Reading guide

Symbol	Meaning
Blue box	Information or definition
Teal box	Key idea to remember
Red box	Warning or common misconception
Amber box	Tip
Dark code panel	Program, command or SQL listing

Course Introduction

Information Security introduces students to the managerial and technological aspects of protecting information and information systems. The course examines security incidents, threats, vulnerabilities, attacks, prevention, detection, response, cryptographic techniques, authentication, system security, database protection, network defence and web application security. Students develop the ability to recognise security problems, analyse risks, understand appropriate countermeasures and communicate security-related decisions.

Table 57. Official course specification

Field	Information
Course Title	Information Security
Course Code	CSIS2022
Programme	ADP Computer Science
Semester	Third Semester
Credit Hours	2 + 1 (3)
Duration	16 Weeks, 32 Sessions
Author	Professor Muhammad Umar Naheed

Course objectives

- Describe information security threats and countermeasures.
- Assess security issues and develop appropriate defences.
- Explain how to respond to information security events.
- Explain common-key and public-key cryptographic techniques.
- Describe mechanisms that safeguard data privacy, integrity and accuracy.

Course learning outcomes

- Explain the CIA Triad.
- Identify assets, threats, vulnerabilities, exploits and risks.
- Perform introductory risk assessment.
- Understand security policies and controls.
- Explain classical and modern cryptographic methods.
- Understand encryption, decryption and cryptographic keys.
- Describe security incident detection and response.
- Communicate security issues clearly.
- Apply appropriate measures to protect information assets.
- Understand basic compliance and ethical responsibilities.
- Explain system, database, network and web security.
- Recognise common cybersecurity attacks and their countermeasures.

Course-to-module mapping

Table 58. Course objectives mapped to modules

Course objective	Modules
1. Threats and countermeasures	1, 2, 13, 15, 16
2. Assess security issues and develop defences	1, 8, 12, 14, 15, 16
3. Respond to security events	2, 15 (incident response, logs, IDS), 16 (ethics)
4. Common-key and public-key cryptography	3 to 7, 9, 10
5. Privacy, integrity and accuracy	2, 9, 11, 12, 14

Official course calendar

Table 59. Course calendar (as supplied)

Week	Topics and assessments
1	Foundations of Information Security
2	Security attacks, services and classical cipher introduction
3	Polyalphabetic ciphers and Assignment 1
4	Hill cipher, cryptanalysis and Quiz 1
5	RC4 and introduction to DES
6	DES and Assignment 2
7	Modular arithmetic, risk management and Quiz 2
8	Revision
9	Midterm
10	AES, public-key cryptography, RSA, hashes, digital signatures and PKI
11	Authentication and Assignment 3
12	Software and database security, buffer overflow, GRANT and REVOKE, Quiz 3
13	Operating system security, access control, network security, honeypots and firewalls
14	Host-based intrusion detection and Assignment 4
15	Web security, SQL injection, XSS, legal and ethical issues, Quiz 4
16	Presentation and revision
Final	Follow the officially announced academic schedule

Calendar note

The supplied course information refers to 16 teaching weeks but also lists the final term as Week 17. The calendar above is kept as supplied. The final-examination date, marks and assessment weightings are not specified here and must be taken from the institution's confirmed academic schedule.

Table of Contents

Copyright and Publication Information 3

Dedication 4

Author's Preface 5

Message to Students 6

About the Author 7

Author's Teaching Philosophy 9

How to Use This Textbook 10

Course Introduction 11

Table of Contents 13

List of Figures 18

List of Tables 19

List of Practical Exercises 21

List of Abbreviations 22

Module 01: Foundations of Information Security 24

1.1 What is Information Security? . . . 24
1.2 Data versus Information . . . 25
1.3 Information Assets and Objectives of Security . . . 26
1.4 Basic Security Terminology . . . 27
1.5 System Security and Common Weaknesses . . . 27
1.6 Social Engineering . . . 28
1.7 Malware and Computer Viruses . . . 29
1.8 Prevention and Protection . . . 30
Module 1 Revision . . . 33
Module 1 Question Bank . . . 34

Module 02: Security Attacks and Security Services 37

2.1 Security Attacks: Passive and Active . . . 37

2.2 Security Services . . . 38
2.3 Security Mechanisms . . . 39
2.4 Solutions to CIA-Related Issues . . . 40
Module 2 Revision . . . 42
Module 2 Question Bank . . . 42

Module 03: Introduction to Cryptography and Classical Ciphers 46

3.1 Cryptography, Cryptology and Cryptanalysis . . . 46
3.2 Symmetric and Asymmetric Cryptography . . . 47
3.3 Classical Cryptographic Techniques . . . 48
3.4 Mathematical Foundations: Alphabet Positions and Modular Arithmetic . . . 48
Module 3 Revision . . . 50
Module 3 Question Bank . . . 50

Module 04: Monoalphabetic Ciphers 54

4.1 Caesar Cipher . . . 54
4.2 General Substitution Cipher and Key Space . . . 55
4.3 Playfair Cipher . . . 55
Module 4 Revision . . . 61
Module 4 Question Bank . . . 61

Module 05: Polyalphabetic and Transposition Ciphers 65

5.1 Vigenere Cipher . . . 65
5.2 Autokey System . . . 66
5.3 Rail Fence Cipher . . . 67
5.4 Strengths and Weaknesses of Classical Ciphers . . . 68
Module 5 Revision . . . 70
Module 5 Question Bank . . . 71

Module 06: Hill Cipher and Cryptanalysis 74

6.1 Hill Cipher . . . 74
6.2 Introduction to Cryptanalysis . . . 75
Module 6 Revision . . . 78
Module 6 Question Bank . . . 79

Module 07: Stream Ciphers and the Data Encryption Standard 82

7.1 Stream and Block Cipher Fundamentals . . . 82
7.2 RC4 . . . 83
7.3 Data Encryption Standard (DES) . . . 84
Module 7 Revision . . . 87
Module 7 Question Bank . . . 88

Module 08: Modular Arithmetic and Risk Management **91**

8.1 Modular Arithmetic and Congruence . . . 91
8.2 GCD, Euclidean and Extended Euclidean Algorithms . . . 92
8.3 Risk Management . . . 93
Module 8 Revision . . . 97
Module 8 Question Bank . . . 98

Module 09: Advanced Encryption Standard **101**

9.1 Introduction and Structure of AES . . . 101
9.2 Keys, IVs, Nonces and Modes of Operation . . . 103
Module 9 Revision . . . 105
Module 9 Question Bank . . . 106

Module 10: Public-Key Cryptography and RSA **109**

10.1 Public-Key Cryptography . . . 109
10.2 RSA Algorithm . . . 110
Module 10 Revision . . . 112
Module 10 Question Bank . . . 113

Module 11: Hash Functions, Digital Signatures and PKI **116**

11.1 Cryptographic Hash Functions . . . 116
11.2 Digital Signatures . . . 117
11.3 Public Key Infrastructure (PKI) . . . 118
Module 11 Revision . . . 122
Module 11 Question Bank . . . 123

Module 12: Authentication and Access Management **126**

12.1 Identification, Authentication and Authorisation . . . 126
12.2 Password-Based Authentication and Storage . . . 127
12.3 Tokens, Factors and Multi-Factor Authentication . . . 128
Module 12 Revision . . . 131
Module 12 Question Bank . . . 132

Module 13: Software Security and Buffer Overflow **135**

13.1 Software Security and Secure Development . . . 135
13.2 Buffer Overflow . . . 136
Module 13 Revision . . . 139
Module 13 Question Bank . . . 140

Module 14: Database Security and Operating System Security **143**

Part A: Database Security . . . 143
14.1 Database Security Fundamentals . . . 143

Part B: Operating System Security . . . 145
14.2 Operating System Security and Access Control Models . . . 145
Module 14 Revision . . . 148
Module 14 Question Bank . . . 149

Module 15: Network Security, Firewalls and Intrusion Detection 152

15.1 Network Security Fundamentals and Common Attacks . . . 152
15.2 Firewalls . . . 153
15.3 Intrusion Detection and Prevention . . . 154
Module 15 Revision . . . 160
Module 15 Question Bank . . . 161

Module 16: Web Security, Application Attacks, Ethics and Projects 164

16.1 Web Security Fundamentals and Architecture . . . 164
16.2 SQL Injection . . . 165
16.3 Cross-Site Scripting (XSS) . . . 166
16.4 Legal Issues, Ethics and Professional Conduct . . . 167
16.5 Student Research Presentations and Security Project . . . 170
Module 16 Revision . . . 170
Module 16 Question Bank . . . 171

Back Matter A: Consolidated Practical Laboratory Index 174

Back Matter B: Important Command Reference 177

Back Matter C: Cryptography Formula Sheet 178

Back Matter D: Important Algorithms and Pseudocode 179

Back Matter E: Security Terminology Glossary 180

Back Matter F: Comprehensive Revision Guide 182

Back Matter G: Midterm Preparation 183

Back Matter H: Final Examination Preparation 184

Back Matter I: Comprehensive MCQ Bank 185

Back Matter J: Objective Answer Key and Explanations (Modules 1 to 16) 189

Back Matter K: Project and Presentation Guidelines 195

Back Matter L: Academic Integrity Statement 196

Back Matter M: Further Learning Resources **197**

Back Matter N: References and Bibliography **198**

Final Message from the Author **199**

Back Matter O: Subject Index **200**

List of Figures

Figure 1. Information security architecture: layered protection around assets 25
Figure 2. The CIA Triad 26
Figure 3. Relationship between threat, vulnerability, asset and risk 27
Figure 4. Social engineering lifecycle 28
Figure 5. Malware classification 29
Figure 6. Virus lifecycle 29
Figure 7. Passive and active attacks 38
Figure 8. Security services and the mechanisms that implement them 40
Figure 9. The encryption and decryption model 47
Figure 10. Caesar cipher with shift 3 54
Figure 11. Playfair matrix for the key MONARCHY and the three rules 56
Figure 12. Vigenere table (tabula recta) 66
Figure 13. Rail Fence zigzag with 3 rails 67
Figure 14. Hill cipher: encrypting HE with a 2x2 key 74
Figure 15. RC4 process: key scheduling, key stream and XOR 83
Figure 16. DES Feistel round structure 84
Figure 17. Modular arithmetic as a clock 92
Figure 18. Risk assessment lifecycle 94
Figure 19. AES round transformations 102
Figure 20. RSA key generation and use 110
Figure 21. Hashing: fixed-size, one-way digest 116
Figure 22. Digital signature: sign and verify 118
Figure 23. PKI certificate chain 119
Figure 24. Authentication, authorisation and factors 126
Figure 25. Stack layout and the direction of an overflow 136
Figure 26. Database access control flow 143
Figure 27. MAC, DAC and RBAC compared 145
Figure 28. Firewall architecture with DMZ 153
Figure 29. Network intrusion detection pipeline 155
Figure 30. Incident response lifecycle 156
Figure 31. SQL injection and its prevention 165
Figure 32. XSS and its prevention 166

List of Tables

Table 55. The five questions answered for every topic 9
Table 56. Reading guide 10
Table 57. Official course specification 11
Table 58. Course objectives mapped to modules 12
Table 59. Course calendar (as supplied) 12
Table 1. CIA classification examples 27
Table 2. Core security terminology 27
Table 3. Malware types compared 30
Table 4. Sample asset register 31
Table 5. CIA analysis exercise 31
Table 6. Phishing indicators 32
Table 7. Malware indicators 33
Table 8. Comparison for Module 1 33
Table 9. Interruption, interception, modification and fabrication 38
Table 10. Mapping attacks to services and mechanisms 40
Table 11. Attack classification worksheet 41
Table 12. Comparison for Module 2 42
Table 13. Symmetric versus asymmetric cryptography 48
Table 14. Alphabet-position table (A = 0 to Z = 25) 48
Table 15. Comparison for Module 3 50
Table 16. Comparison for Module 4 61
Table 17. Rail Fence results 67
Table 18. Classical cipher evaluation 68
Table 19. Comparison for Module 5 71
Table 20. Comparison for Module 6 78
Table 21. Stream versus block ciphers 83
Table 22. DES round trace (selected rounds) 85
Table 23. Comparison for Module 7 88
Table 24. Modular operations 92
Table 25. Square-and-multiply for 5^13 mod 23 92
Table 26. Extended Euclidean table for gcd(26, 7) 93
Table 27. Qualitative risk matrix (5 x 5 scoring) 94
Table 28. Sample risk register 97
Table 29. Comparison for Module 8 98
Table 30. AES variants 102
Table 31. AES-128 round 1 (verified) 103
Table 32. Modes of operation compared 103
Table 33. Comparison for Module 9 106

Table 34. Comparison for Module 10 113
Table 35. Hashing versus encryption 117
Table 36. Hash algorithms 117
Table 37. Comparison for Module 11 122
Table 38. Password protections 127
Table 39. Comparison for Module 12 132
Table 40. Buffer overflow defences 137
Table 41. Comparison for Module 13 140
Table 42. Comparison of MAC, DAC, ACL and RBAC 146
Table 43. Expected results (not executed in the authoring environment) 147
Table 44. OS security checklist 148
Table 45. Comparison for Module 14 149
Table 46. Common network attacks and defences 153
Table 47. Firewall types, basing and location 154
Table 48. IDS versus IPS 155
Table 49. Expected firewall behaviour (reference) 157
Table 50. Common Wireshark display filters 157
Table 51. Comparison for Module 15 160
Table 52. Principles of professional conduct 167
Table 53. Model assessment findings 170
Table 54. Comparison for Module 16 171
Table 60. Practical laboratory index 175
Table 61. Command quick reference 177
Table 62. Formula sheet 178
Table 63. Frequently confused concepts 182
Table 64. Midterm coverage 183
Table 65. Final examination emphasis 184
Table 66. Suggested practice tests 184

List of Practical Exercises

Practical 1: Identifying Information Assets	30
Practical 2: Analysing the CIA Triad	31
Practical 3: Recognising Phishing Emails	32
Practical 4: Identifying Malware Indicators	32
Practical 5: Classifying Security Attacks	41
Practical 33: Alphabet Mapping and Modular Arithmetic Warm-up	49
Practical 6: Caesar Cipher Encryption and Decryption	57
Practical 7: General Substitution Cipher	58
Practical 8: Playfair Cipher	59
Practical 9: Vigenere and Autokey Ciphers	68
Practical 10: Rail Fence Cipher	69
Practical 11: Hill Cipher Calculations	76
Practical 35: Caesar Cipher Cryptanalysis by Frequency Analysis	77
Practical 14: Educational RC4 Demonstration	86
Practical 15: DES Structure Simulation	87
Practical 12: Euclidean Algorithm	95
Practical 13: Modular Inverse	96
Practical 34: Risk Register for a Fictional Organisation	97
Practical 16: AES Encryption and Decryption (AES-256-GCM)	104
Practical 17: RSA Key Generation, Encryption and Decryption	111
Practical 18: SHA-256 Hashing and File Integrity	120
Practical 19: Digital Signature Creation and Verification	120
Practical 20: TLS Certificate Inspection	121
Practical 21: Password Security: Salted Hashing	129
Practical 22: Authentication and MFA: TOTP	130
Practical 36: Inspecting Authentication Logs	130
Practical 23: Safe Buffer Overflow Demonstration	138
Practical 24: Database GRANT and REVOKE	146
Practical 25: Operating System Permissions	147
Practical 26: Firewall Configuration	156
Practical 27: Wireshark Traffic Inspection	157
Practical 28: Security Log Analysis	158
Practical 29: Intrusion Detection Concepts and Honeypot Demonstration	159
Practical 30: SQL Injection Demonstration and Prevention	168
Practical 31: XSS Prevention with Output Encoding	168
Practical 32: Security Assessment of a Fictional University System	169

List of Abbreviations

Abbreviation	Meaning
ACL	Access Control List
ADP	Associate Degree Programme
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
APA	American Psychological Association
ASLR	Address Space Layout Randomisation
CA	Certificate Authority
CBC	Cipher Block Chaining
CIA	Confidentiality, Integrity, Availability
CRL	Certificate Revocation List
CSP	Content Security Policy
CTR	Counter mode
DAC	Discretionary Access Control
DDoS	Distributed Denial of Service
DEP	Data Execution Prevention
DES	Data Encryption Standard
DMZ	Demilitarised Zone
DoS	Denial of Service
ECB	Electronic Codebook
FIPS	Federal Information Processing Standard
GCD	Greatest Common Divisor
GCM	Galois/Counter Mode
HIDS	Host-based Intrusion Detection System
HIPS	Host-based Intrusion Prevention System
HMAC	Hash-based Message Authentication Code
HTTP(S)	Hypertext Transfer Protocol (Secure)
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
IV	Initialisation Vector
KDF	Key Derivation Function
KSA	Key Scheduling Algorithm
MAC	Mandatory Access Control / Message Authentication Code
MFA	Multi-Factor Authentication
MITM	Man-in-the-Middle
NIDS	Network Intrusion Detection System
NIST	National Institute of Standards and Technology
OCSP	Online Certificate Status Protocol
OWASP	Open Worldwide Application Security Project
PKI	Public Key Infrastructure

Abbreviation	Meaning
PRGA	Pseudo-Random Generation Algorithm
RBAC	Role-Based Access Control
RFC	Request for Comments
RPO / RTO	Recovery Point / Time Objective
RRL	Risk Reduction Leverage
RSA	Rivest-Shamir-Adleman
SHA	Secure Hash Algorithm
SIEM	Security Information and Event Management
SOC	Security Operations Centre
SQL	Structured Query Language
TLS	Transport Layer Security
TOTP	Time-based One-Time Password
XSS	Cross-Site Scripting

Module 01 Foundations of Information Security

Introduction. Every time you log in to a university portal, send a message, or pay a fee online, you rely on information being protected. This module builds the vocabulary and mental models on which the rest of the course stands: what we protect, what we protect it from, and how we describe the difference between a weakness and an actual loss.

Why this topic matters. Without a shared language, security discussions become vague. A student who can say "this is a vulnerability, that is a threat, and the combination creates this risk" can reason about any system, from a mobile app to a hospital network.

Real-world relevance. Online banking, student portals, hospital records, e-commerce checkouts and social media all depend on the principles in this module.

Learning objectives	Expected learning outcomes
• Define information security and explain why it matters. • Distinguish data from information, and assets from controls. • Explain confidentiality, integrity and availability (the CIA Triad). • Use the terms asset, threat, vulnerability, exploit, attack, risk and control correctly. • Describe social engineering and malware, including the virus lifecycle.	• Classify a situation as a confidentiality, integrity or availability failure. • Identify information assets in a university. • Recognise phishing messages and malware indicators. • Choose basic preventive measures.

Prerequisites	Estimated learning time
None. Basic computer use is enough.	About 6 hours: 2 sessions of teaching plus 2 hours of practicals.

1.1 What is Information Security?

In simple words. Information security means keeping information safe from people or events that could misuse, damage, change or block it.

Formal definition. Information security is the protection of information and information systems from unauthorised access, use, disclosure, disruption, modification or destruction in order to provide confidentiality, integrity and availability.

Everyday analogy. A school office keeps records in a locked cabinet (confidentiality), checks that nobody changes marks (integrity), and makes sure staff can open the cabinet during working hours (availability).

How it works. Security combines **people** (training and ethics), **process** (policies and procedures) and **technology** (encryption, firewalls, access control). A weakness in any one weakens the whole.

Figure 1 | Information security architecture: layered protection around assets

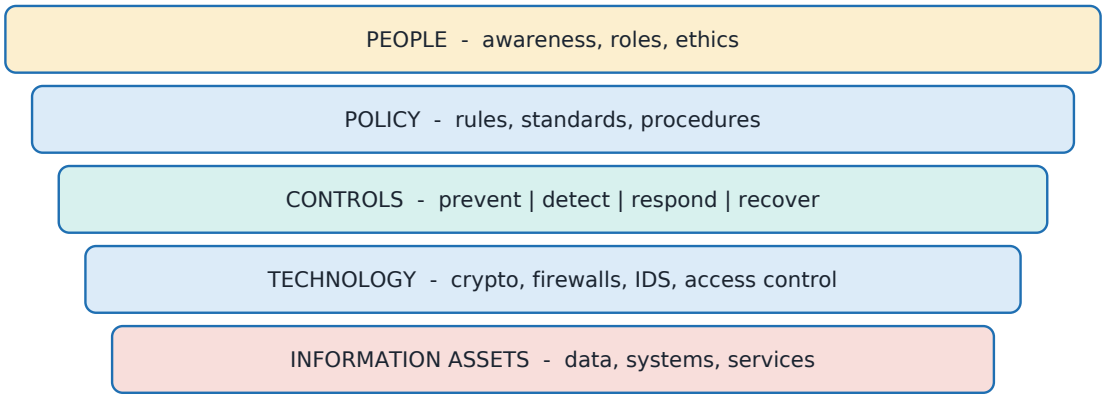


Figure 1. *Information security architecture: layered protection around assets*

Read the figure from top to bottom. Protection is built in layers so that if one fails, another still defends the asset.

Term	Meaning
Information security	Protecting information in any form: digital, paper or spoken.
Cybersecurity	Protecting systems, networks and data in cyberspace; a subset of information security.
Information system	Hardware, software, data, people and procedures that collect and process information.

Practical applications. Student information systems, electronic health records, exam-paper storage and tax databases.

> **Common misconception**
> Security is not only about hackers or only about technology. Many losses come from mistakes, lost devices and weak procedures.

Security implications. A single stolen exam paper can invalidate an entire examination session, so protecting information has real academic and financial cost.

1.2 Data versus Information

In simple words. Data are raw facts. Information is data that has been organised so that it means something.

Formal definition. Data are unprocessed symbols, numbers or text; information is data that has been processed, organised and interpreted to be meaningful in context (Stallings & Brown, 2018).

Everyday analogy. The numbers 85, 72, 91 are data. "Your average mark is 82.7" is information.

How it works. Systems collect data, process it with software, store it in databases and present information to users. Security must protect all three stages: data in storage, data in transit and data in use.

Term	Meaning
Data at rest	Stored on disks, databases or backups.
Data in transit	Moving across a network.
Data in use	Being processed in memory.

Practical applications. A bank stores transactions (data) and produces statements (information).

Security implications. A leaked list of numbers may be harmless, yet the same numbers with labels such as "salary" or "diagnosis" can seriously harm a person.

1.3 Information Assets and Objectives of Security

In simple words. An asset is anything of value that needs protection. Security has three main goals: keep secrets secret, keep data correct, keep systems usable.

Formal definition. An information asset is any data, device, system, service or person-held knowledge that has value to an organisation. The objectives of information security are to preserve confidentiality, integrity and availability and to support accountability and authenticity.

Everyday analogy. Think of a house: valuables (assets), locks (controls), the possibility of a burglar (threat), and an unlocked window (vulnerability).

How it works. List the assets, rank them by value and sensitivity, then decide which protections are justified. Security spending should match asset value.

Term	Meaning
Hardware asset	Servers, laptops, routers, phones.
Software asset	Operating systems, applications, source code.
Data asset	Student records, grades, financial data.
People and process	Staff knowledge, approval procedures.

The CIA Triad

Confidentiality prevents unauthorised disclosure (encryption, access control). **Integrity** prevents unauthorised or accidental alteration (hashes, digital signatures, version control). **Availability** ensures authorised users can reach systems and data when needed (backups, redundancy, protection against denial of service). The three goals can conflict: very strict confidentiality controls may reduce availability, so good design balances them against business needs.

Figure 2 | The CIA Triad

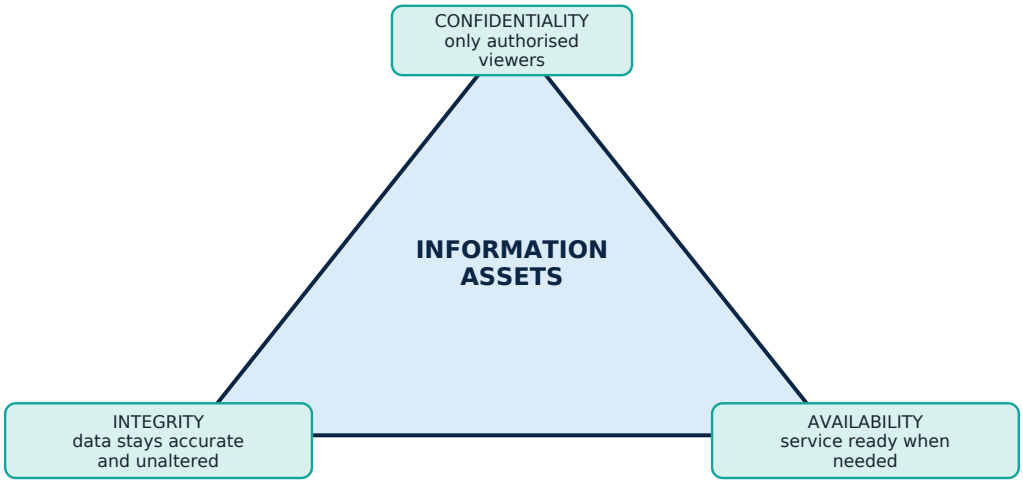


Figure 2. *The CIA Triad*

Each corner is a goal; the asset sits at the centre because every control exists to protect it.

Table 1. CIA classification examples

Scenario	Goal violated	Why
A student reads another student's results	Confidentiality	Disclosed to an unauthorised person
A clerk's typing error changes a mark from 78 to 18	Integrity	Data no longer accurate
Portal is down on result day because of a flood of requests	Availability	Authorised users cannot access service
A hacker changes a bank balance and steals a customer list	Integrity and confidentiality	Two goals broken at once

1.4 Basic Security Terminology

Table 2. Core security terminology

Term	Definition	University example
Asset	Anything valuable	Student database
Threat	Potential cause of harm	Criminal group, malware, flood, careless insider
Vulnerability	Weakness that can be exploited	Unpatched server, weak password
Exploit	Method or code that takes advantage of a vulnerability	Script that abuses the unpatched server
Attack	Deliberate attempt to harm or misuse	Using the exploit to read records
Risk	Likelihood of loss multiplied by its impact	High chance + high impact on the student database
Control (countermeasure)	Safeguard that reduces risk	Patching, MFA, backups, firewall

Figure 3 | Relationship between threat, vulnerability, asset and risk

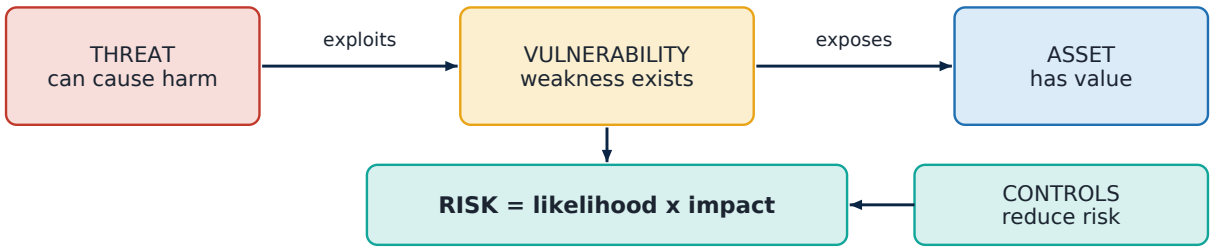


Figure 3. *Relationship between threat, vulnerability, asset and risk*

A threat only causes loss when it can exploit a vulnerability in a valuable asset. Controls reduce either the likelihood or the impact.

Remember
Risk exists only where all three meet: a valuable **asset**, a **vulnerability** and a **threat**. Remove any one, and risk falls.

1.5 System Security and Common Weaknesses

In simple words. System security means protecting the computer itself: its operating system, accounts, software and settings.

Formal definition. System security is the set of measures that protect the confidentiality, integrity and availability of a computer system and its resources, including account control, patching, malware defence, configuration management and logging.

Everyday analogy. It is like maintaining a car: regular service (patching), lock and alarm (access control), and a seat belt (backups) all matter.

How it works. Common weaknesses include default or weak passwords, missing updates, unnecessary services, excessive privileges, unencrypted storage and no backups. Hardening removes or reduces them.

Practical applications. Laboratories, hospital workstations and personal laptops.

Security implications. Most real-world breaches use known weaknesses that had simply not been fixed.

1.6 Social Engineering

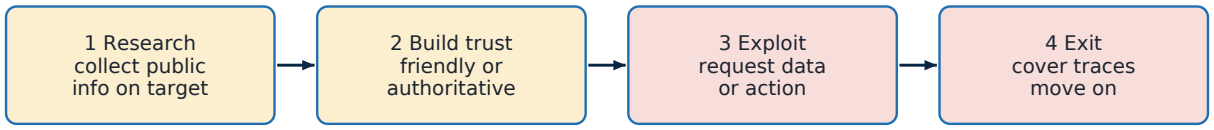
In simple words. Social engineering tricks people instead of computers. The attacker persuades someone to hand over information or do something unsafe.

Formal definition. Social engineering is the psychological manipulation of people into performing actions or divulging confidential information.

Everyday analogy. A stranger in a uniform says "I am here to fix the printer" and is let into the office. The uniform replaced the lock.

How it works. Attackers research the target, build trust or urgency, exploit the victim, then leave without leaving traces.

Figure 4 | Social engineering lifecycle



Defence: verify identity, never share secrets, report suspicious contact

Figure 4. *Social engineering lifecycle*

Every stage depends on trust or urgency. Verification through a second channel breaks the chain.

Term	Meaning
Phishing	Fraudulent e-mail or message that imitates a trusted sender.
Spear phishing	Phishing aimed at a specific person using personal details.
Vishing / smishing	Voice-call or SMS versions of phishing.
Pretexting	Inventing a believable story to obtain information.
Baiting	Leaving infected USB drives or offering free downloads.
Tailgating	Following an authorised person through a secured door.

Practical applications. Fake "account suspended" emails, fake fee-payment links, fake job offers.

Common misconception
Only careless people fall for it.

Security implications. Anyone can be tricked when rushed or frightened; defence is training, verification procedures and technical filtering.

1.7 Malware and Computer Viruses

In simple words. Malware is software built to harm, spy on or take control of a system. A virus is malware that attaches itself to other files and spreads when those files are run or shared.

Formal definition. Malware is any software intentionally designed to cause damage to, or unauthorised actions on, a computer system. A computer virus is malicious code that replicates by inserting copies of itself into other programs or files.

Everyday analogy. A biological virus needs a host cell. A computer virus needs a host file; when you open the file, it spreads.

How it works. A virus passes through four phases: dormant, propagation, triggering and execution. Defence includes updated antivirus, patching, least privilege, careful downloads, and backups.

Figure 5 | Malware classification

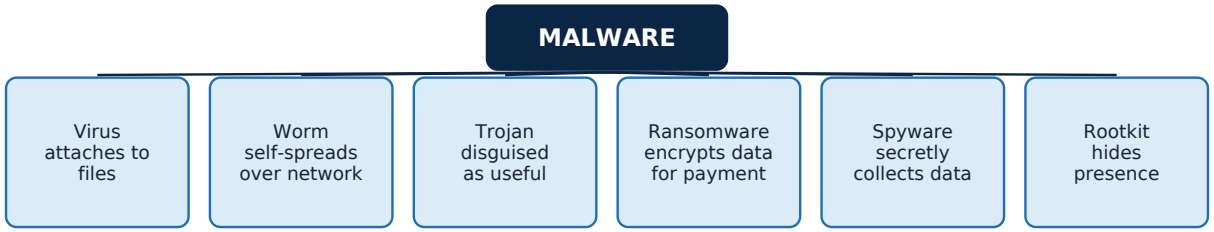
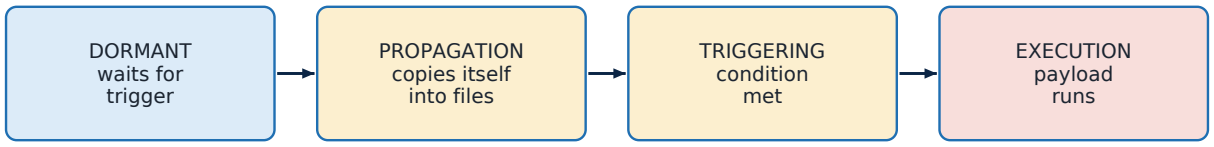


Figure 5. *Malware classification*

Figure 6 | Virus lifecycle



Antivirus can interrupt every phase: scan, detect signature/behaviour, quarantine

Figure 6. *Virus lifecycle*

Term	Meaning
Boot-sector virus	Infects the startup area of a disk.
File-infector virus	Attaches to executable files.
Macro virus	Hides in document macros.
Polymorphic virus	Changes its appearance to avoid signature detection.
Stealth virus	Hides modifications from the operating system.

Common misconception
Antivirus finds every threat.

Security implications. Signature-based antivirus misses new malware; behaviour monitoring, patching and user care are also required.

Table 3. Malware types compared

Type	Spreads by	Needs host file?	Main goal
Virus	User running infected file	Yes	Replicate, damage
Worm	Network, automatically	No	Spread, consume resources
Trojan	User installs it believing it is useful	No	Backdoor, theft
Ransomware	Phishing, vulnerabilities	No	Encrypt data for payment
Spyware	Bundled software	No	Collect information

1.8 Prevention and Protection

- Keep operating system and software updated (patch management).
- Use reputable antivirus with automatic updates.
- Apply least privilege: daily work with a standard account.
- Use strong unique passwords and multi-factor authentication.
- Take regular, tested, offline backups (3-2-1 rule: 3 copies, 2 media, 1 off-site).
- Never open unexpected attachments; verify sender by another channel.

Case Study: Fake fee-challan email at a university

Background	In the week before semester fee deadlines, many students receive an email about a "fee verification".
Organisation and environment	A university with a student portal, 12,000 students, a small IT team.
Information assets	Student login credentials, personal data, fee records.
Security problem	A look-alike website steals credentials.
Threats and vulnerabilities	Threat: criminal phishing group. Vulnerabilities: no MFA, students unaware, no email banner for external senders.
Potential impact	About 300 accounts compromised; fraudulent changes to payment details.
Detection	A student reports the email; the mail gateway shows a spike of clicks on one domain.
Investigation	IT checks headers, the link domain, and portal logs for logins from unusual countries.
Response	Block the domain, reset passwords, force MFA, notify students, restore altered records from backup.
Prevention	Mandatory MFA, mail filtering with external-sender warnings, awareness campaign, official fee links only on the portal.
Lessons learned	Technology plus human awareness plus quick reporting limited the damage. Fictional scenario for teaching.

Practical 1: Identifying Information Assets

Learning objective	Identify, classify and rank the information assets of a university.
Software and tools	Pen and paper or spreadsheet.
Environment	Classroom or home; no network needed.
Background	Asset inventory is the first step of every risk assessment.

Step-by-step procedure

1. List at least 12 assets across hardware, software, data, people and process.
2. Assign each a value (1 to 5) for confidentiality, integrity and availability.
3. Add the totals and rank the assets.
4. Mark the three most critical and suggest one control each.

Table 4. Sample asset register

Asset	Type	C	I	A	Total
Student grades database	Data	5	5	4	14
Wi-Fi access points	Hardware	2	3	4	9
Exam paper drafts	Data	5	5	2	12

Output interpretation	The highest totals are the assets that deserve the strongest controls first.
Verification procedure	Compare your ranking with a classmate; differences must be explained by reasons, not opinion.
Common errors	Listing only computers; forgetting data, people and reputation.
Troubleshooting	If everything scores 5, force a ranking by asking what hurts most if lost.
Security precautions	Use only your own institution's publicly known information; do not scan any network.
Practical conclusion	You have built the starting point for a risk assessment.
Student questions	1) Why does availability score low for exam drafts? 2) Which asset has the highest impact if altered? 3) Name two assets that are not computers.

Practical 2: Analysing the CIA Triad

Learning objective	Classify security incidents by the CIA goal they violate.
Software and tools	Pen and paper.
Environment	Classroom.
Background	Many incidents break more than one goal.

Step-by-step procedure

1. Read each incident below.
2. Write which goals are violated and why.
3. Propose one control for each.

Table 5. CIA analysis exercise

#	Incident	Model answer
1	Laptop with unencrypted student data is stolen	Confidentiality; control: disk encryption
2	Ransomware encrypts the library server	Availability (and possibly confidentiality); control: offline backups
3	A teacher's marks file is edited by an unknown user	Integrity; control: access control, audit logs, hashes
4	Website is unreachable during a traffic flood	Availability; control: rate limiting, CDN

Output interpretation	Matching each loss to a goal shows which control type is needed.
Verification procedure	Ask: who could see it, who could change it, who could not reach it?
Common errors	Choosing only one goal when two apply.
Troubleshooting	Re-read the incident for words such as "stolen", "altered", "unavailable".
Security precautions	Use fictional data only.
Practical conclusion	CIA is a diagnostic tool, not just a definition.
Student questions	1) Which goal is hardest to restore after loss? 2) Give an example where improving confidentiality harms availability. 3) Design a new incident and classify it.

Practical 3: Recognising Phishing Emails

Learning objective	Identify warning signs in suspicious messages.
Software and tools	Sample printed or on-screen emails (fictional).
Environment	Offline; do not click any real suspicious link.
Background	Phishing is the most common starting point of real breaches.

Step-by-step procedure

1. Read the sample message below.
2. Mark every warning sign.
3. Write the safe response.

Sample suspicious message (fictional)

```
From: fees-support@univ-portal-verify.example.net
Subject: URGENT!!! Your account will be closed in 24 hrs

Dear Customer, your fee record has errors. Click here to login
http://univ-portal.verify-fees.example.net/login and confirm your
password and ID card number.
```

Table 6. Phishing indicators

Warning sign	Explanation
Look-alike sender domain	Not the official university domain
Urgency and threat	Pressure prevents careful thinking
Generic greeting	Real offices use your name
Requests password and ID number	Legitimate staff never ask for passwords
Link text differs from true address	Hover to reveal the real destination

Output interpretation	Three or more indicators means treat as phishing.
Verification procedure	Check the sender domain, hover over links, and contact the office through its official number.
Common errors	Trusting a message because it uses the real logo.
Troubleshooting	Logos are easily copied; verify the address, not the appearance.
Security precautions	Never enter credentials from an email link; report the message.
Practical conclusion	Careful inspection stops most phishing.
Student questions	1) Why is urgency a warning sign? 2) What is the difference between phishing and spear phishing? 3) What should you do if you already clicked?

Practical 4: Identifying Malware Indicators

Learning objective	Recognise signs of infection and respond safely.
Software and tools	Checklist; optionally a lab virtual machine.
Environment	Use only an isolated virtual machine for any real file analysis.
Background	Early detection limits damage.

Step-by-step procedure

1. Study the indicator table.
2. Match each symptom to a likely cause and first response.

Table 7. Malware indicators

Symptom	Possible meaning	First response
Sudden slowness and fans always loud	Cryptominer or background malware	Check Task Manager; run full scan
Files renamed with strange extension and a note	Ransomware	Disconnect network; do not pay; restore from backup
Browser homepage changed	Adware or hijacker	Remove extension; scan
Antivirus switched off by itself	Malware tampering	Isolate machine; investigate

Output interpretation	Symptoms are clues, not proof; confirm with scans and logs.
Verification procedure	Compare the process list and startup entries against a clean machine.
Common errors	Panic deletion of files.
Troubleshooting	Isolate first, investigate second.
Security precautions	Never run unknown samples on your main computer.
Practical conclusion	Prompt isolation protects other systems.
Student questions	1) Why disconnect from the network first? 2) What is the 3-2-1 backup rule? 3) Why is paying a ransom risky?

Module 1 Revision

1. Module summary

Information security protects the confidentiality, integrity and availability of information assets. Risk arises when a threat exploits a vulnerability in an asset. Social engineering targets people; malware targets systems; layered controls reduce both.

2. Key definitions

Term	Definition
CIA Triad	Confidentiality, Integrity, Availability.
Threat	Potential cause of harm.
Vulnerability	Weakness that may be exploited.
Risk	Likelihood x impact of a loss.
Virus	Malware that replicates by attaching to files.

3. Important terminology

Asset, threat, vulnerability, exploit, attack, control, phishing, pretexting, tailgating, payload, quarantine.

4. Comparison table

Table 8. Comparison for Module 1

Concept	Virus	Worm	Trojan
Needs host	Yes	No	No
Self-spreads	With user action	Automatically	No
Deceives user	Sometimes	No	Yes

5. Quick revision notes

- Security = people + process + technology.

- Risk needs asset, threat and vulnerability.
- Data become information when organised and interpreted.

6. Memory techniques

- CIA: "Cats In Alleys" = Confidentiality, Integrity, Availability.
- Virus phases: DPTE = Dormant, Propagate, Trigger, Execute.

7. Important formulas

- Risk = Likelihood x Impact (qualitative or numeric).

8. Common examination mistakes

- Confusing a threat with a vulnerability.
- Saying antivirus gives complete protection.
- Mixing worm and virus.

Module 1 Question Bank

9. Short questions (15)

S1. Define information security.
S2. Differentiate data and information.
S3. What is an information asset? Give two examples.
S4. State the three goals of the CIA Triad.
S5. Define a vulnerability.
S6. Define a threat.
S7. What is the difference between an attack and an exploit?
S8. Define risk.
S9. What is a security control?
S10. Define social engineering.
S11. What is spear phishing?
S12. Define malware.
S13. What is a computer virus?
S14. Name the four phases of a virus lifecycle.
S15. What is the 3-2-1 backup rule?

10. Long questions (8)

L1. Explain the CIA Triad with a university example for each goal.
L2. Explain the relationship between threat, vulnerability, asset and risk with a diagram.
L3. Describe the layers of an information security architecture.
L4. Discuss social engineering techniques and their countermeasures.
L5. Compare viruses, worms and Trojans.
L6. Explain the virus lifecycle and how antivirus interrupts it.
L7. Differentiate data at rest, in transit and in use, with protection methods.
L8. Explain why security spending should be proportional to asset value.

11. Multiple-choice questions (10)

Q1. Which goal is violated when a student reads another student's marks?

A) Availability
B) Integrity
C) Authenticity
D) Confidentiality

Q2. A weakness that can be exploited is a:

A) Control
B) Vulnerability
C) Threat
D) Asset

Q3. Which malware does not need a host file?

A) Worm
B) Boot virus
C) Macro virus
D) Virus

Q4. Risk is best described as:

A) Vulnerability only
B) Threat only
C) Likelihood multiplied by impact
D) Number of assets

Q5. Tailgating is:

A) Password guessing
B) A network scan
C) Following someone through a secure door
D) Email spoofing

Q6. Which is an availability control?

A) Hashing
B) Backups and redundancy
C) Digital signature
D) Encryption

Q7. Phishing mainly exploits:

A) Software bugs
B) Hardware faults
C) Network cables
D) Human trust

Q8. The final phase of the virus lifecycle is:

A) Propagation
B) Dormant
C) Execution
D) Triggering

Q9. Which is NOT a CIA goal?

A) Popularity
B) Availability
C) Integrity
D) Confidentiality

Q10. Using standard accounts for daily work applies:

A) Obscurity
B) Least privilege
C) Open design
D) Redundancy

12. Scenario-based questions (5)

C1. A clerk accidentally changes the passing mark of 40 students. Which goal failed and what control would help?

C2. A caller claims to be from IT and asks for your password. Describe your response.

C3. A clinic loses access to patient files because of ransomware. Identify goals affected and recovery steps.

C4. A USB drive found in the car park is plugged into a lab PC. What attack is this and how do you prevent it?

C5. A website is unreachable during results. Which goal is affected and what could cause it?

13. Practical and problem-solving exercises (3)

X1. Build an asset register of 10 items for your college library and rank them.

X2. Create a one-page awareness poster on spotting phishing.

X3. Classify 8 incidents of your choice by CIA goal.

Model answers (selected short questions)

S5. A vulnerability is a weakness in an asset or control, such as an unpatched server, that a threat can exploit.

S8. Risk is the combination of how likely a loss is and how severe its impact would be.

S15. Keep 3 copies of data, on 2 different media, with 1 copy stored off-site.

Detailed model answers (selected long questions)

L1. The CIA Triad consists of confidentiality, integrity and availability. In a university, confidentiality means only authorised staff and the student see grades; integrity means grades cannot be changed without authority and any change is logged; availability means the portal is reachable during result announcement. Controls respectively include access control and encryption, hashing and audit logs, and redundancy with backups. The goals can conflict, so security design balances them.

14. References

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

Stallings, W., & Brown, L. (2018). *Computer security: Principles and practice* (4th ed.). Pearson.

Easttom, W. (2019). *Computer security fundamentals* (4th ed.). Pearson.

Pfleeger, C. P., Pfleeger, S. L., & Margulies, J. (2015). *Security in computing* (5th ed.). Prentice Hall.

Module 02
Security Attacks and Security Services

Introduction. Module 1 gave us vocabulary. Here we study how attacks are classified and which services and mechanisms answer them. The framework used is the OSI Security Architecture (ITU-T X.800), which organises security into attacks, services and mechanisms.

Why this topic matters. A defender who can name the attack type immediately knows which service is needed and which mechanism can provide it.

Real-world relevance. Online banking replay protection, hospital audit trails, and exam-portal denial-of-service defence all use ideas from this module.

Learning objectives	Expected learning outcomes
• Classify attacks as passive or active. • Describe interruption, interception, modification and fabrication. • Explain the six security services. • Describe specific and pervasive security mechanisms. • Map attacks to CIA impact and suitable controls.	• Identify the attack in a scenario. • Choose the correct security service. • Match services to mechanisms. • Analyse a fictional university incident.

Prerequisites	Estimated learning time
Module 1 (CIA Triad, threat and vulnerability).	About 6 hours including practicals.

2.1 Security Attacks: Passive and Active

In simple words. A passive attacker only watches. An active attacker changes, injects or blocks.

Formal definition. A passive attack attempts to learn or use information from a system without affecting system resources. An active attack attempts to alter system resources or affect their operation (Stallings, 2017).

Everyday analogy. A passive attacker is like someone listening at the door; an active attacker opens the envelope, rewrites the letter and posts it again.

How it works. Passive attacks are hard to detect but can be prevented, mainly by encryption. Active attacks are easier to detect but harder to prevent completely, so detection and recovery are emphasised.

Figure 7 | Passive and active attacks

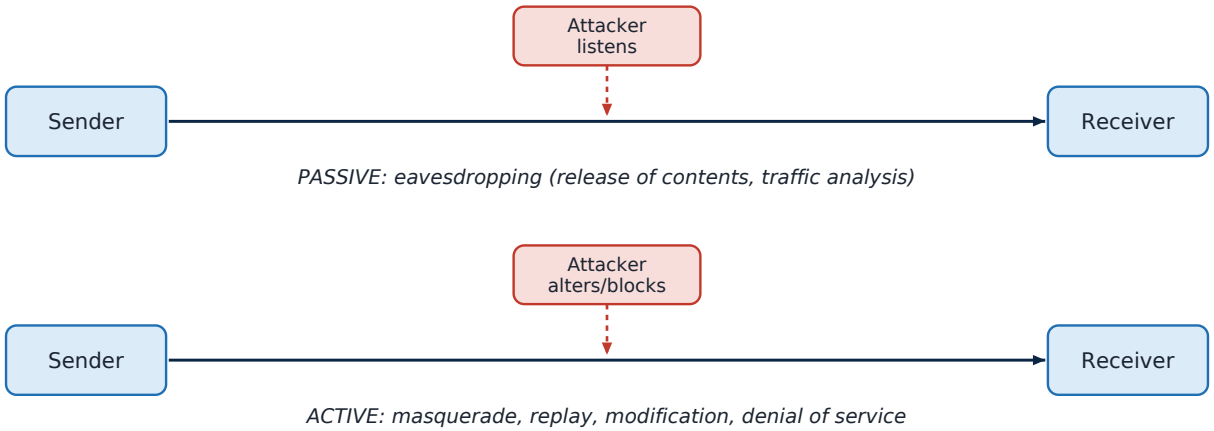


Figure 7. *Passive and active attacks*

Top: the attacker only observes. Bottom: the attacker interferes with the flow.

Term	Meaning
Release of message contents	Reading sensitive data such as an email or file in transit.
Traffic analysis	Observing who talks to whom, when and how much, even if content is encrypted.
Masquerade	One entity pretends to be another.
Replay	Capturing a valid message and resending it later.
Message modification	Altering part of a legitimate message.
Denial of Service	Preventing normal use of a service.

Practical applications. Wi-Fi eavesdropping at cafes, replaying a captured payment request, flooding a result website.

> **Common misconception**
> Encrypted traffic reveals nothing.

Security implications. Encryption hides contents, but metadata (who, when, how much) still leaks through traffic analysis.

Four categories of attack on a system

Table 9. Interruption, interception, modification and fabrication

Category	Meaning	Violates	Example
Interruption	Asset is destroyed or made unavailable	Availability	Cutting a cable, flooding a server
Interception	Unauthorised party gains access	Confidentiality	Packet sniffing, wiretapping
Modification	Unauthorised party alters an asset	Integrity	Changing database values
Fabrication	Unauthorised party inserts counterfeit objects	Authenticity	Adding a fake record or forged message

2.2 Security Services

In simple words. A security service is a protection we want, such as "prove who you are" or "keep this secret".

Formal definition. A security service is a processing or communication service that enhances the security of data processing systems and information transfers, making use of one or more security mechanisms (ITU-T X.800).

Everyday analogy. A service is the promise on a bank counter ("your money is safe"); a mechanism is the vault, camera or lock that keeps the promise.

How it works. Services are selected from the risk analysis; each service is implemented by one or more mechanisms.

Term	Meaning
Authentication	Assurance that a communicating entity or data origin is who it claims to be.
Access control	Prevention of unauthorised use of a resource.
Data confidentiality	Protection of data from unauthorised disclosure.
Data integrity	Assurance data is received exactly as sent; detects modification, insertion, deletion, replay.
Non-repudiation	Prevents sender or receiver from denying having sent or received a message.
Availability	System is accessible and usable on demand by authorised entities.

Practical applications. Banks use all six: login (authentication), account permissions (access control), TLS (confidentiality), checksums and signatures (integrity), signed transaction logs (non-repudiation), and redundant servers (availability).

2.3 Security Mechanisms

In simple words. Mechanisms are the tools and techniques that deliver the services.

Formal definition. A security mechanism is a process, device or technique designed to detect, prevent or recover from a security attack. X.800 separates specific mechanisms (tied to one service) from pervasive mechanisms (not tied to any one service).

Everyday analogy. Specific mechanisms are individual locks; pervasive mechanisms are the alarm system and the guard patrol that help everything.

How it works. Each mechanism is chosen for a service; pervasive mechanisms add overall trust and recovery.

Term	Meaning
Encipherment	Using cryptography to make data unreadable.
Digital signature	Data appended to or derived from a message to prove origin and integrity.
Access control mechanism	Passwords, permissions, tokens that enforce rights.
Data integrity mechanism	Checksum, hash or MAC to detect change.
Authentication exchange	Protocol where entities prove identity to each other.
Traffic padding	Insert bogus data to hide traffic patterns.
Routing control	Select physically secure routes or avoid risky ones.
Notarisation	Trusted third party assures a property of an exchange.
Trusted functionality	Hardware or software assumed correct (for example a secure OS kernel).
Security label	Marking resources with sensitivity such as Confidential.
Event detection	Detecting security-relevant events such as repeated logins.
Security audit trail	Independent record of system events for review.
Security recovery	Actions to restore state after a breach (backups, failover).

Figure 8 | Security services and the mechanisms that implement them

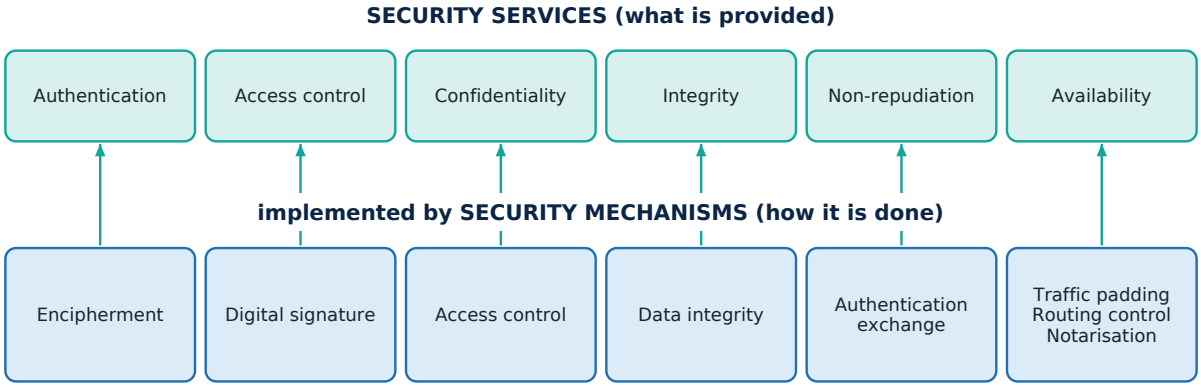


Figure 8. *Security services and the mechanisms that implement them*

Arrows show which mechanism group commonly supports each service. One service may need several mechanisms.

Table 10. Mapping attacks to services and mechanisms

Attack	Service that counters it	Typical mechanism
Eavesdropping	Confidentiality	Encipherment (TLS)
Masquerade	Authentication	Authentication exchange, MFA
Replay	Integrity / authentication	Nonces, timestamps, sequence numbers
Modification	Integrity	Hash, MAC, digital signature
Denial of service	Availability	Redundancy, rate limiting, event detection
Denial of having sent	Non-repudiation	Digital signature, notarisation, audit trail
Unauthorised access	Access control	Permissions, labels, trusted functionality

2.4 Solutions to CIA-Related Issues

Confidentiality problems are addressed by encryption, access control and classification. Integrity problems are addressed by hashes, digital signatures, version control and least privilege. Availability problems are addressed by backups, redundancy, load balancing, patching and denial-of-service protection. Always combine **prevention**, **detection** and **recovery**: no prevention is perfect.

Case Study: Replay and masquerade at a campus payment kiosk

Background	Students pay small fees at self-service kiosks on campus.
Organisation and environment	University finance office, kiosks connected to a payment server over the campus LAN.
Information assets	Payment requests, student IDs, transaction logs.
Security problem	A captured payment-confirmation message is re-sent, crediting an unpaid account.
Threats and vulnerabilities	Threat: insider with sniffing tool. Vulnerabilities: no timestamps or nonces, no integrity protection.
Potential impact	Free credits for several students; revenue loss; loss of trust.
Detection	Daily reconciliation shows credits without matching bank receipts.
Investigation	Packet capture reveals identical confirmation messages repeated.
Response	Disable affected accounts, correct ledgers, add message authentication.
Prevention	Add nonces and timestamps, MAC or TLS with server authentication, and alert on duplicate transaction IDs.
Lessons learned	Defences against active attacks need integrity and freshness, not just secrecy. Fictional scenario.

Practical 5: Classifying Security Attacks

Learning objective	Classify attacks and map them to CIA impact and services.
Software and tools	Pen and paper.
Environment	Classroom.
Background	Naming an attack leads directly to the right defence.

Step-by-step procedure

1. Read each scenario.
2. Classify as passive or active, then name the category.
3. State the violated goal and the service needed.

Table 11. Attack classification worksheet

Scenario	Class / type	Goal	Service
Student reads a colleague's email on shared Wi-Fi	Passive, interception	Confidentiality	Confidentiality
Someone logs in using a stolen password	Active, masquerade	Authentication	Authentication, MFA
Fee confirmation message sent twice	Active, replay	Integrity	Integrity, freshness
Portal fails under massive requests	Active, DoS (interruption)	Availability	Availability
A lecturer denies sending a notice	Repudiation	Non-repudiation	Digital signature

Output interpretation	Each row follows the same decision path: observe or change? what goal? what service?
Verification procedure	Check that each answer names one class, one goal and one service.
Common errors	Confusing replay with modification.
Troubleshooting	Replay resends an unchanged message; modification changes it.
Security precautions	Use fictional scenarios only.
Practical conclusion	Classification is the first step of incident analysis.
Student questions	1) Why is traffic analysis passive? 2) Which mechanism supports non-repudiation? 3) Why are active attacks harder to prevent?

Module 2 Revision

1. Module summary

Attacks are passive (observe) or active (alter). Services (authentication, access control, confidentiality, integrity, non-repudiation, availability) are delivered by specific and pervasive mechanisms. Good defence combines prevention, detection and recovery.

2. Key definitions

Term	Definition
Passive attack	Monitoring without altering.
Active attack	Altering or disrupting.
Security service	Protection goal.
Security mechanism	Tool implementing a service.
Non-repudiation	Prevents denial of an action.

3. Important terminology

Masquerade, replay, traffic analysis, notarisation, padding, audit trail, security label.

4. Comparison table

Table 12. Comparison for Module 2

	Passive	Active
Effect on system	None	Altered/disrupted
Detection	Hard	Easier
Defence emphasis	Prevention	Detection and recovery

5. Quick revision notes

- Service = what; mechanism = how.
- Interruption=availability, interception=confidentiality, modification=integrity, fabrication=authenticity.

6. Memory techniques

- IMIF = Interruption, Interception, Modification, Fabrication.
- SAC-IND = Services: Authentication, Access control, Confidentiality, Integrity, Non-repudiation, Denial (availability).

7. Important formulas

- None in this module; see CIA mapping table.

8. Common examination mistakes

- Treating encryption as the solution to every attack.
- Confusing service with mechanism.

Module 2 Question Bank

9. Short questions (15)

S1. Define passive attack.

S2. Define active attack.

S3. What is traffic analysis?

S4. What is a masquerade attack?

S5. What is a replay attack?

S6. Define denial of service.
S7. Differentiate interruption and interception.
S8. Define fabrication.
S9. List the six security services.
S10. Define non-repudiation.
S11. What is a security mechanism?
S12. What is notarisation?
S13. What is a security audit trail?
S14. What is traffic padding?
S15. What is security recovery?

10. Long questions (8)

L1. Differentiate passive and active attacks with examples.
L2. Explain interruption, interception, modification and fabrication with CIA impact.
L3. Describe the six security services.
L4. Explain specific security mechanisms.
L5. Explain pervasive mechanisms with examples.
L6. Map five attacks to services and mechanisms.
L7. Explain how replay attacks are prevented.
L8. Discuss prevention, detection and recovery as a combined strategy.

11. Multiple-choice questions (10)

Q1. Reading another user's message on the network is:
A) Interception
B) Active attack
C) Fabrication
D) Interruption

Q2. Replay attacks are best countered with:
A) Nonces/timestamps
B) Bigger monitors
C) Colour coding
D) Passwords only

Q3. Which is a pervasive mechanism?
A) Encipherment
B) Security audit trail
C) Traffic padding
D) Digital signature

Q4. Non-repudiation is mainly supported by:
A) Padding
B) Backups
C) Digital signatures
D) Firewalls

Q5. Traffic analysis is:
A) Fabrication
B) Modification
C) Active
D) Passive

Q6. Flooding a server mainly violates:

A) Integrity

B) Confidentiality

C) Authenticity

D) Availability

Q7. Inserting a fake record is:

A) Interruption

B) Fabrication

C) Modification

D) Interception

Q8. Security labels are used for:

A) Encryption

B) Routing

C) Marking sensitivity levels

D) Hashing

Q9. Which attack is hardest to detect?

A) Modification

B) DoS

C) Active

D) Passive

Q10. A mechanism is:

A) A promise

B) An attack

C) The tool implementing a service

D) A threat

12. Scenario-based questions (5)

C1. An attacker repeats a valid "transfer 5000" message. Name the attack and defence.

C2. Encrypted messages are intercepted but patterns reveal busy hours. What attack and which mechanism?

C3. A professor denies sending a notice. Which service was missing?

C4. A server fails under flood traffic. List three mitigations.

C5. A user claims they never made an online purchase. How can the shop prove otherwise?

13. Practical and problem-solving exercises (3)

X1. Create a table of 6 attacks, each with service and mechanism.

X2. Draw the attack categories diagram from memory.

X3. Write a one-page analysis of the kiosk case.

Model answers (selected short questions)

S3. Traffic analysis studies communication patterns (who, when, how much) even when the content is encrypted.

S10. Non-repudiation prevents a party from falsely denying that it sent or received a message.

Detailed model answers (selected long questions)

L3. The six services are authentication (verifies identity), access control (limits use of resources), data confidentiality (protects against disclosure), data integrity (detects alteration), non-repudiation (prevents denial of actions) and availability (keeps service usable). In an online bank: login with MFA, account permissions, TLS, MACs on transactions, signed logs, and redundant servers.

14. References

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

Stallings, W., & Brown, L. (2018). *Computer security: Principles and practice* (4th ed.). Pearson.

Pfleeger, C. P., Pfleeger, S. L., & Margulies, J. (2015). *Security in computing* (5th ed.). Prentice Hall.

Easttom, W. (2019). *Computer security fundamentals* (4th ed.). Pearson.

Module 03
Introduction to Cryptography and Classical Ciphers

Introduction. Cryptography is the science of protecting information by transforming it. We begin with the historical ciphers because they teach the vocabulary and the logic of encryption in a form you can compute with a pencil.

Why this topic matters. Every modern secure system, from Wi-Fi to online banking, is built from the same ideas: plaintext, ciphertext, algorithm and key.

Real-world relevance. The same terminology appears in TLS, messaging apps, password storage and digital signatures.

Learning objectives	Expected learning outcomes
• Define cryptography, cryptology and cryptanalysis. • Use the terms plaintext, ciphertext, encryption, decryption and key. • Distinguish symmetric and asymmetric cryptography. • Explain substitution and transposition. • Convert letters to numbers and apply modular arithmetic.	• Encrypt and decrypt a message by hand. • Explain why the key must be secret. • Map the alphabet to positions 0 to 25.

Prerequisites	Estimated learning time
Module 1 and basic arithmetic.	About 5 hours including practicals.

3.1 Cryptography, Cryptology and Cryptanalysis

In simple words. Cryptography hides information. Cryptanalysis tries to uncover it without permission. Together they are cryptology.

Formal definition. Cryptography is the study of techniques for secure communication in the presence of adversaries. Cryptanalysis is the study of breaking cryptographic systems. Cryptology is the combined field.

Everyday analogy. Cryptography is making a lock; cryptanalysis is picking it; cryptology is the study of both.

How it works. A sender applies an encryption algorithm with a key to plaintext, producing ciphertext. The receiver applies decryption with the matching key. An attacker who sees ciphertext but not the key should learn nothing useful.

Figure 9 | The encryption and decryption model

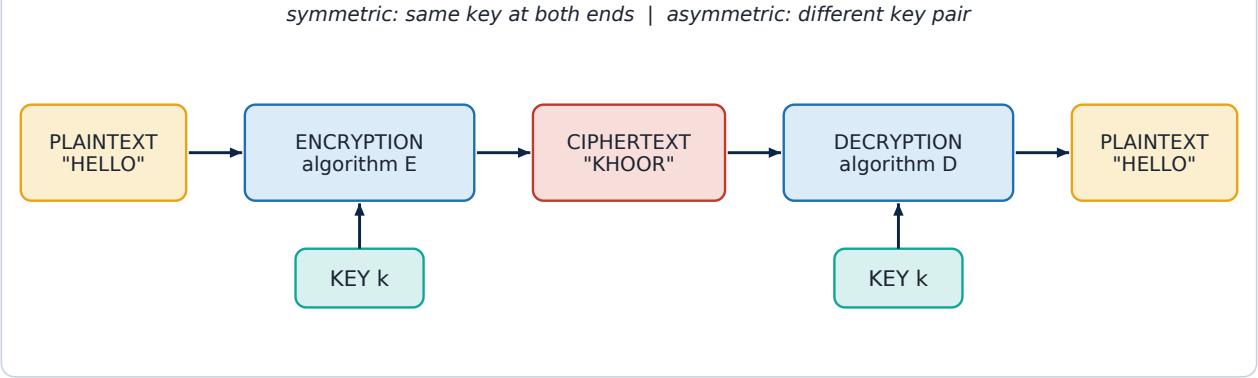


Figure 9. *The encryption and decryption model*

Both parties need the key. The attacker sees only the ciphertext.

Term	Meaning
Plaintext	The original readable message.
Ciphertext	The scrambled output.
Encryption	Plaintext to ciphertext, C = E(K, P).
Decryption	Ciphertext to plaintext, P = D(K, C).
Key	Secret value controlling the transformation.
Cipher	The encryption and decryption algorithms.
Cryptanalyst	Person attempting to break a cipher.

Practical applications. Messaging apps, VPNs, secure web browsing, digital payment cards.

Common misconception
A secret algorithm makes a system secure.

Security implications. Kerckhoffs's principle: security must depend on the secrecy of the key, not the algorithm.

3.2 Symmetric and Asymmetric Cryptography

In simple words. Symmetric: one shared secret key. Asymmetric: a public key to lock, a private key to unlock.

Formal definition. Symmetric cryptography uses the same key for encryption and decryption. Asymmetric (public-key) cryptography uses a mathematically related key pair, where the public key may be shared openly.

Everyday analogy. Symmetric: both friends hold identical house keys. Asymmetric: a letterbox anyone can drop letters into, but only the owner can open.

How it works. Symmetric ciphers are fast and used for bulk data, but share keys securely. Asymmetric ciphers solve key distribution and enable signatures but are slower. Real systems combine both.

Term	Meaning
Key distribution problem	How to share a secret key safely.
Hybrid encryption	Public-key crypto exchanges a symmetric session key.

Security implications. Classical ciphers in this module are all symmetric.

Table 13. Symmetric versus asymmetric cryptography

Feature	Symmetric	Asymmetric
Keys	One shared key	Public/private pair
Speed	Fast	Slower
Key distribution	Difficult	Easier
Examples	Caesar, DES, AES	RSA, ECC
Typical use	Bulk data	Key exchange, signatures

3.3 Classical Cryptographic Techniques

In simple words. Old ciphers either swap letters for other letters (substitution) or shuffle their positions (transposition).

Formal definition. Substitution replaces plaintext units with other units; transposition rearranges the positions of plaintext units without changing them. Product ciphers combine both.

Everyday analogy. Substitution is replacing each word with a code word; transposition is scrambling the order of words in a sentence.

How it works. Caesar, Playfair and Vigenere are substitution ciphers; Rail Fence is a transposition cipher. Modern ciphers such as AES repeat substitution and permutation many times.

Term	Meaning
Monoalphabetic	One fixed substitution for the whole message.
Polyalphabetic	Several substitutions used in rotation.
Diffusion	Spreading plaintext influence across ciphertext (transposition-like).
Confusion	Hiding the key-ciphertext relationship (substitution-like).

3.4 Mathematical Foundations: Alphabet Positions and Modular Arithmetic

We convert letters to numbers so that encryption becomes arithmetic. The alphabet is numbered 0 to 25. After 25 we wrap around, which is exactly what the **mod 26** operation does: $a \bmod 26$ is the remainder after dividing by 26.

Table 14. Alphabet-position table (A = 0 to Z = 25)

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Worked example

Encrypt letter **X** (23) by shifting +5: $(23 + 5) \bmod 26 = 28 \bmod 26 =$ **2**, which is letter **C**. Decrypt C: $(2 - 5) \bmod 26 = -3 \bmod 26 =$ **23**, which is X. Negative values are brought back into range by adding 26.

Case Study: Why a secret cipher failed at a small online shop

Background	A small shop designs its own "secret" scrambling method for customer order codes.
Organisation and environment	Online shop, 2 developers, a shared hosting server.
Information assets	Order data and customer details.
Security problem	Order codes are scrambled with a home-made substitution.
Threats and vulnerabilities	Threat: curious customer. Vulnerability: reliance on secret algorithm; patterns visible.
Potential impact	Order numbers predictable; customers view others' orders.
Detection	A customer notices repeated patterns in codes.
Investigation	Developers analyse frequencies and confirm the pattern is reversible.
Response	Replace with random identifiers, authorisation checks and standard encryption.
Prevention	Use proven algorithms and keep keys, not designs, secret.
Lessons learned	Never invent your own cryptography. Fictional scenario.

Practical 33: Alphabet Mapping and Modular Arithmetic Warm-up

Learning objective	Convert text to numbers and practise mod 26.
Software and tools	Pen and paper; Python optional.
Environment	Any.
Background	All classical ciphers use these conversions.

Step-by-step procedure

1. Convert HELLO to numbers.
2. Add 7 to each and reduce mod 26.
3. Convert back to letters.
4. Reduce -4, 30 and 52 mod 26.

Worked solution

```
H E L L O  ->  7 4 11 11 14
+7 -> 14 11 18 18 21 -> O L S S V
-4 mod 26 = 22   30 mod 26 = 4   52 mod 26 = 0
```

Output interpretation	HELLO becomes OLSSV; the wrap rule gives 22, 4, 0.
Verification procedure	Decrypt OLSSV by subtracting 7 to recover HELLO.
Common errors	Forgetting that A is 0 not 1.
Troubleshooting	Write the table first.
Security precautions	None.
Practical conclusion	Arithmetic with letters is the key skill for Modules 4 to 6.
Student questions	1) What is 27 mod 26? 2) What is -1 mod 26? 3) Why must results stay between 0 and 25?

Module 3 Revision

1. Module summary

Cryptography transforms plaintext into ciphertext with a key. Symmetric ciphers share one key; asymmetric ciphers use key pairs. Classical ciphers use substitution and transposition and rely on modular arithmetic.

2. Key definitions

Term	Definition
Cryptography	Science of secure communication.
Cryptanalysis	Breaking ciphers.
Key	Secret parameter of a cipher.
Substitution	Replace symbols.
Transposition	Rearrange positions.

3. Important terminology

Kerckhoffs's principle, confusion, diffusion, product cipher.

4. Comparison table

Table 15. Comparison for Module 3

	Substitution	Transposition
Changes	Letter identity	Letter position
Example	Caesar	Rail Fence

5. Quick revision notes

- C = E(K,P), P = D(K,C).
- Letters map A=0 to Z=25.

6. Memory techniques

- Substitution = Switch letters; Transposition = Travel (move) letters.

7. Important formulas

- C = E(K, P); P = D(K, C); x mod 26 in range 0..25

8. Common examination mistakes

- Using A = 1.
- Forgetting to wrap around.

Module 3 Question Bank

9. Short questions (15)

S1. Define cryptography.
S2. Define cryptanalysis.
S3. What is cryptology?
S4. Differentiate plaintext and ciphertext.
S5. Define a cipher key.
S6. What is symmetric cryptography?
S7. What is asymmetric cryptography?
S8. State Kerckhoffs's principle.
S9. Define substitution cipher.

S10. Define transposition cipher.
S11. What is a product cipher?
S12. What is diffusion?
S13. What is confusion?
S14. Compute 29 mod 26.
S15. Compute -5 mod 26.

10. Long questions (8)

L1. Explain the encryption model with a diagram.
L2. Compare symmetric and asymmetric cryptography.
L3. Explain classical techniques with examples.
L4. Explain mod 26 arithmetic with examples.
L5. Discuss why algorithm secrecy is not security.
L6. Explain hybrid encryption.
L7. Differentiate monoalphabetic and polyalphabetic substitution.
L8. Explain confusion and diffusion.

11. Multiple-choice questions (10)

Q1. Which statement matches Kerckhoffs's principle?
A) Never publish papers
B) Use short keys
C) Only the key must be secret
D) Hide the algorithm

Q2. Position of letter K (A=0) is:
A) 9
B) 10
C) 11
D) 12

Q3. Caesar cipher is:
A) Substitution
B) Hash
C) Transposition
D) Asymmetric

Q4. RSA is:
A) Symmetric
B) Asymmetric
C) Transposition
D) Classical

Q5. (-3) mod 26 equals:
A) 26
B) 3
C) 23
D) -23

Q6. Rail Fence is a:

A) Transposition cipher

B) Hash

C) Substitution cipher

D) Stream cipher

Q7. Cryptanalysis is:

A) Storing keys

B) Making ciphers

C) Breaking ciphers

D) Sending mail

Q8. Symmetric ciphers are generally:

A) Always secure

B) Slower than asymmetric

C) Unable to encrypt

D) Faster than asymmetric

Q9. The ciphertext is produced by:

A) Hashing only

B) Decryption

C) Signing only

D) Encryption

Q10. Hybrid encryption uses public-key crypto to:

A) Delete keys

B) Exchange a symmetric key

C) Compress files

D) Encrypt every file

12. Scenario-based questions (5)

C1. A company keeps its own secret algorithm. Evaluate.

C2. Two friends want private chat but cannot meet. Which approach?

C3. Encrypt "SAFE" with shift 4.

C4. A message was shifted by 7. How to decrypt?

C5. Why do systems combine symmetric and asymmetric crypto?

13. Practical and problem-solving exercises (3)

X1. Make an alphabet-position wheel.

X2. Encrypt your name with shift 5 and decrypt it.

X3. Compute ten mod-26 reductions including negatives.

Model answers (selected short questions)

S8. Security of a cipher must depend only on secrecy of the key, even if the algorithm is public.

S14. 29 mod 26 = 3.

Detailed model answers (selected long questions)

L2. Symmetric cryptography uses one shared key and is fast, but key distribution is hard. Asymmetric uses a public/private key pair, simplifying key exchange and enabling signatures but is slower. Real systems, such as TLS, use asymmetric methods to establish a key and symmetric encryption for the data.

14. References

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

Paar, C., & Pelzl, J. (2010). *Understanding cryptography: A textbook for students and practitioners*. Springer.

Pfleeger, C. P., Pfleeger, S. L., & Margulies, J. (2015). *Security in computing* (5th ed.). Prentice Hall.

Python Software Foundation. (n.d.). *Python 3 documentation*. https://docs.python.org/3/

Module 04
Monoalphabetic Ciphers

Introduction. A monoalphabetic cipher replaces each plaintext letter by one fixed ciphertext letter. We study the simplest case (Caesar), the general case (any permutation of the alphabet) and a cleverer variant that encrypts pairs of letters (Playfair).

Why this topic matters. These ciphers are easy to compute and show clearly why key size and letter statistics matter.

Real-world relevance. Understanding why simple substitution fails prepares you for the design of modern ciphers.

Learning objectives	Expected learning outcomes
• Explain monoalphabetic substitution. • Encrypt and decrypt with the Caesar cipher using C = (P + k) mod 26. • Calculate the key space of a general substitution cipher. • Explain frequency analysis. • Build a Playfair matrix and encrypt and decrypt digraphs.	• Complete Caesar and Playfair calculations by hand and by program. • Judge cipher strength from key space and statistical leakage.

Prerequisites	Estimated learning time
Module 3.	About 6 hours including practicals.

4.1 Caesar Cipher

In simple words. Move every letter a fixed number of places along the alphabet.

Formal definition. The Caesar cipher is a substitution cipher in which each plaintext letter is replaced by the letter k positions further along the alphabet, with wrap-around: C = (P + k) mod 26 and P = (C − k) mod 26.

Everyday analogy. It is like turning a dial lock by the same number of clicks for every letter.

How it works. Convert letters to numbers (A=0), add the key, reduce mod 26, convert back. For decryption subtract the key. Only 25 non-trivial keys exist.

Figure 10 | Caesar cipher with shift 3

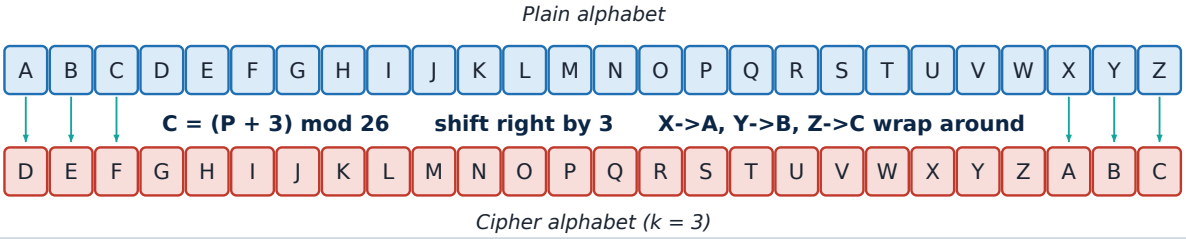


Figure 10. *Caesar cipher with shift 3*

The lower strip is the plain alphabet shifted left; X, Y, Z wrap around to A, B, C.

Term	Meaning
k	Shift key, 1 to 25.

Term	Meaning
P, C	Plaintext and ciphertext letter positions.
ROT13	Caesar with k = 13; its own inverse.

Practical applications. Historically used by Julius Caesar; today only as a teaching tool and puzzle.

Common misconception
Longer messages are safer.

Security implications. With only 25 keys, an attacker simply tries all of them (brute force).

Worked example (verified by program)
Encrypt **CRYPTO** with k = 5. C(2)+5=7 H, R(17)+5=22 W, Y(24)+5=29 mod 26=3 D, P(15)+5=20 U, T(19)+5=24 Y, O(14)+5=19 T. Ciphertext = **HWDUYT**. Decrypting HWDUYT with k = 5 gives **CRYPTO**.

4.2 General Substitution Cipher and Key Space

In simple words. Instead of shifting, shuffle the whole alphabet and use the shuffled alphabet as the key.

Formal definition. A general monoalphabetic substitution cipher uses an arbitrary permutation of the 26 letters as the key, giving 26! possible keys (about 4.03×10^{26}, roughly 2^{88}).

Everyday analogy. A secret decoder ring with the letters scrambled randomly.

How it works. The key is a table such as A to Q, B to W and so on. Encryption looks up each letter; decryption uses the reverse table. Brute force is infeasible, but statistics defeat the cipher.

Term	Meaning
Key space	Number of possible keys.
Keyword alphabet	Alphabet built from a keyword followed by unused letters.

Common misconception
A huge key space guarantees security.

Security implications. Letter frequencies survive substitution, so frequency analysis breaks it with a few hundred letters.

Frequency analysis

In English, the most common letters are approximately E (12.7%), T (9.1%), A (8.2%), O (7.5%), I (7.0%) and N (6.7%). Because a monoalphabetic cipher maps every E to the same ciphertext letter, the commonest ciphertext letter is probably the image of E. Attackers then use common words (THE, AND), digraphs (TH, HE, IN) and double letters to complete the key.

4.3 Playfair Cipher

In simple words. Playfair encrypts letters two at a time using a 5 x 5 grid, which hides single-letter frequencies.

Formal definition. The Playfair cipher encrypts digraphs (pairs of letters) using a 5 x 5 matrix built from a keyword, with I and J sharing one cell.

Everyday analogy. Like looking up two letters together on a chessboard instead of one by one.

How it works. Build matrix, split text into pairs, apply the three rules, output pairs.

Figure 11 | Playfair matrix for the key MONARCHY and the three rules

Key: MONARCHY (I/J share a cell)

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

RULES for each digraph

Same row: take letter to the RIGHT

Same column: take letter BELOW

Rectangle: swap columns (same row)

Double letters: insert filler X

Odd length: append X

```
BA -> IB
LX -> SU
LO -> PM
ON -> NA
```

Figure 11. *Playfair matrix for the key MONARCHY and the three rules*

Term	Meaning
Digraph	Pair of letters.
Filler letter	Usually X, inserted between double letters and to complete the last pair.
I/J combination	J is treated as I.

Practical applications. Used in World War I field communications.

Common misconception
Playfair is unbreakable by hand.

Security implications. It has 600 digraph frequencies instead of 26 letters, which is harder but still crackable with enough text.

Worked example: key SECURITY, plaintext INFORMATION

Matrix (I/J merged):

```
S E C U R
I T Y A B
D F G H K
L M N O P
Q V W X Z
```

Digraphs: **IN FO RM AT IO NX**. Encrypted pairs: **YL HM EP BY AL OW**. Ciphertext: **YLHMEPBYALOW**. Decrypting gives **INFORMATIONX**; the filler X (if any) is removed or understood from context.

Rules recap
Same row: take letter to the right (wrap). **Same column**: take letter below (wrap). **Rectangle**: swap column positions. Decrypt: left, above, and the same rectangle rule.

Case Study: Weak student-ID obfuscation

Background	A club hides member IDs on printed lists using a letter shift.
Organisation and environment	A university society with 200 members.
Information assets	Member list with contact details.
Security problem	List posted on a public notice board.
Threats and vulnerabilities	Threat: casual curiosity. Vulnerability: 25 possible keys.
Potential impact	Privacy loss for all members.
Detection	A student decodes the list in minutes.
Investigation	Brute-force of all shifts reveals readable text.
Response	Withdraw the list, issue neutral IDs held separately.
Prevention	Use access-controlled digital lists with proper encryption.
Lessons learned	Obfuscation is not security. Fictional scenario.

Practical 6: Caesar Cipher Encryption and Decryption

Learning objective	Encrypt, decrypt and brute-force a Caesar cipher.
Software and tools	Python 3.
Environment	Any computer; terminal or IDLE.
Background	Programs make large-scale checking easy.

Step-by-step procedure

1. Save the program as caesar.py.
2. Run: python3 caesar.py
3. Read ciphertext and the brute-force table.
4. Change the message and key and repeat.

Program: caesar.py

```
# Caesar cipher: encrypt, decrypt and brute-force (educational only)
def shift(text, k):
    out = ""
    for ch in text.upper():
        if ch.isalpha():
            out += chr((ord(ch) - 65 + k) % 26 + 65)   # C = (P + k) mod 26
        else:
            out += ch
    return out

msg = "MEET ME AT THE LIBRARY"
ct = shift(msg, 3)
print("Ciphertext :", ct)
print("Decrypted  :", shift(ct, -3))
print("--- brute force on the ciphertext ---")
for k in range(26):
    print(f"k={k:2d}  {shift(ct, -k)}")
```

Expected output

```
Ciphertext : PHHW PH DW WKH OLEUDUB
Decrypted  : MEET ME AT THE LIBRARY
--- brute force on the ciphertext ---
k= 0  PHHW PH DW WKH OLEUDUB
k= 1  OGGV OG CV VJG NKDTCTA
k= 2  NFFU NF BU UIF MJCSBSZ
k= 3  MEET ME AT THE LIBRARY
k= 4  LDDS LD ZS SGD KHAQZQX
k= 5  KCCR KC YR RFC JGZPYPW
k= 6  JBBQ JB XQ QEB IFYOXOV
k= 7  IAAP IA WP PDA HEXNWNU
k= 8  HZZO HZ VO OCZ GDWMVMT
k= 9  GYYN GY UN NBY FCVLULS
k=10  FXXM FX TM MAX EBUKTKR
k=11  EWWL EW SL LZW DATJSJQ
k=12  DVVK DV RK KYV CZSIRIP
k=13  CUUJ CU QJ JXU BYRHQHO
k=14  BTTI BT PI IWT AXQGPGN
k=15  ASSH AS OH HVS ZWPFOFM
k=16  ZRRG ZR NG GUR YVOENEL
k=17  YQQF YQ MF FTQ XUNDMDK
k=18  XPPE XP LE ESP WTMCLCJ
k=19  WOOD WO KD DRO VSLBKBI
k=20  VNNC VN JC CQN URKAJAH
k=21  UMMB UM IB BPM TQJZIZG
k=22  TLLA TL HA AOL SPIYHYF
k=23  SKKZ SK GZ ZNK ROHXGXE
k=24  RJJY RJ FY YMJ QNGWFWD
k=25  QIIX QI EX XLI PMFVEVC
```

Output interpretation	The line with k = 3 shows readable text, which reveals the key.
Verification procedure	Decrypt manually using the formula and compare.
Common errors	Using lowercase letters (the program uppercases first).
Troubleshooting	Check Python version with python3 --version.
Security precautions	Use only your own text.
Practical conclusion	Small key spaces fall to brute force.
Student questions	1) How many keys must an attacker try? 2) What does ROT13 decrypt with? 3) Why is the output readable only at one key?

Practical 7: General Substitution Cipher

Learning objective	Build a keyword-based substitution alphabet and count the key space.
Software and tools	Python 3.
Environment	Terminal.
Background	The key is the whole cipher alphabet.

Step-by-step procedure

1. Save substitution.py.
2. Run it.
3. Change the keyword and compare alphabets.

Program: substitution.py

```
# General monoalphabetic substitution cipher with a keyword-based alphabet
import string
A = string.ascii_uppercase
def make_key(keyword):
    seen = []
    for c in keyword.upper() + A:
        if c.isalpha() and c not in seen:
            seen.append(c)
    return "".join(seen)

key = make_key("CYBERSECURITY")
print("Plain  :", A)
print("Cipher :", key)
enc = str.maketrans(A, key)
dec = str.maketrans(key, A)
ct = "INFORMATION SECURITY".translate(enc)
print("Encrypted:", ct)
print("Decrypted:", ct.translate(dec))
import math
print("Key space = 26! = %d  (about 2^%.1f)" % (math.factorial(26), math.log2(math.factorial(26))))
```

Output

```
Plain  : ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher : CYBERSUITADFGHJKLMNOPQVWXZ
Encrypted: THSJMGCOTJH NRBPMTOX
Decrypted: INFORMATION SECURITY
Key space = 26! = 403291461126605635584000000  (about 2^88.4)
```

Output interpretation	The key space is 26! (about 2^88), too large for brute force, but frequency analysis remains.
Verification procedure	Decrypt the output and compare with the original.
Common errors	Repeated letters in the keyword.
Troubleshooting	The program removes duplicates automatically.
Security precautions	Use sample text only.
Practical conclusion	Large key space is not enough.
Student questions	1) Why does frequency analysis still work? 2) How many bits is 26! roughly? 3) Why remove duplicates from the keyword?

Practical 8: Playfair Cipher

Learning objective	Build a Playfair matrix and encrypt and decrypt digraphs.
Software and tools	Python 3.
Environment	Terminal.
Background	Playfair works on pairs of letters.

Step-by-step procedure

1. Save playfair.py.
2. Run it.
3. Test other keywords and plaintexts.

Program: playfair.py

```
# Playfair cipher (I/J combined, filler X)
import string
def matrix(keyword):
    seq = []
    for c in (keyword.upper() + string.ascii_uppercase).replace("J", "I"):
        if c.isalpha() and c not in seq:
            seq.append(c)
    return [seq[i:i + 5] for i in range(0, 25, 5)]
def find(M, c):
    for r in range(5):
        if c in M[r]:
            return r, M[r].index(c)
def digraphs(text):
    t = "".join(c for c in text.upper().replace("J", "I") if c.isalpha())
    out, i = [], 0
    while i < len(t):
        a = t[i]
        b = t[i + 1] if i + 1 < len(t) else "X"
        if a == b:
            out.append(a + "X"); i += 1
        else:
            out.append(a + b); i += 2
    return out
def crypt(M, pairs, d):            # d = +1 encrypt, -1 decrypt
    res = []
    for a, b in pairs:
        (r1, c1), (r2, c2) = find(M, a), find(M, b)
        if r1 == r2:   res.append(M[r1][(c1 + d) % 5] + M[r2][(c2 + d) % 5])
        elif c1 == c2: res.append(M[(r1 + d) % 5][c1] + M[(r2 + d) % 5][c2])
        else:          res.append(M[r1][c2] + M[r2][c1])
    return res
M = matrix("MONARCHY")
for row in M: print(" ".join(row))
p = digraphs("BALLOON")
print("Digraphs  :", p)
c = crypt(M, p, +1)
print("Ciphertext:", "".join(c))
print("Decrypted :", "".join(crypt(M, c, -1)), "(filler X remains)")
```

Output

```
M O N A R
C H Y B D
E F G I K
L P Q S T
U V W X Z
Digraphs  : ['BA', 'LX', 'LO', 'ON']
Ciphertext: IBSUPMNA
Decrypted : BALXLOON (filler X remains)
```

Output interpretation	The matrix, digraphs and ciphertext appear in order; BALLOON becomes BA LX LO ON.
Verification procedure	Decrypt and confirm the inserted X.
Common errors	Forgetting the I/J merge.
Troubleshooting	Check double letters were split.
Security precautions	Sample text only.
Practical conclusion	Digraph ciphers improve on single-letter ones.
Student questions	1) Why insert X between double letters? 2) How many possible digraphs exist? 3) What happens when both letters share a row?

Module 4 Revision

1. Module summary

Caesar shifts letters by k; the key space is only 25. General substitution has 26! keys but leaks letter frequencies. Playfair encrypts digraphs using a 5x5 matrix and is harder but still breakable.

2. Key definitions

Term	Definition
Monoalphabetic	One fixed mapping.
Key space	Number of possible keys.
Frequency analysis	Using letter statistics to break ciphers.
Digraph	Two-letter unit.

3. Important terminology

Caesar, ROT13, keyword alphabet, digraph, filler.

4. Comparison table

Table 16. Comparison for Module 4

Cipher	Key space	Weakness
Caesar	25	Brute force
General substitution	26!	Frequency analysis
Playfair	25! matrices (about 2^84)	Digraph analysis with enough text

5. Quick revision notes

- C = (P+k) mod 26; P = (C-k) mod 26.
- Playfair: row right, column down, rectangle swap.

6. Memory techniques

- Playfair: "Row Right, Column Down, Rectangle Swap".

7. Important formulas

- C = (P + k) mod 26
- P = (C - k) mod 26
- Key space = 26!

8. Common examination mistakes

- Using A = 1.
- Forgetting to wrap.
- Forgetting I/J merge.

Module 4 Question Bank

9. Short questions (15)

S1. Define monoalphabetic cipher.

S2. Write the Caesar encryption formula.

S3. Write the Caesar decryption formula.

S4. How many Caesar keys exist?

S5. What is the key space of general substitution?

S6. What is frequency analysis?
S7. Which letter is most common in English?
S8. What is ROT13?
S9. What is a digraph?
S10. Why is I/J merged in Playfair?
S11. When is a filler letter used?
S12. State the Playfair same-row rule.
S13. State the Playfair rectangle rule.
S14. Is Playfair secure today?
S15. What is a keyword alphabet?

10. Long questions (8)

L1. Explain the Caesar cipher with a worked example.
L2. Explain why a key space of 25 makes Caesar vulnerable to exhaustive search.
L3. Calculate and explain the key space of general substitution.
L4. Explain frequency analysis step by step.
L5. Construct a Playfair matrix for key MONARCHY.
L6. Encrypt a message with Playfair, showing every digraph.
L7. Decrypt a Playfair ciphertext.
L8. Compare Caesar, substitution and Playfair.

11. Multiple-choice questions (10)

Q1. Caesar key space is:
A) 26
B) 25
C) 26!
D) 2^128

Q2. Encrypt D with k=3:
A) F
B) H
C) A
D) G

Q3. General substitution key space:
A) 26!
B) 52
C) 26
D) 2^26

Q4. Frequency analysis exploits:
A) Letter statistics
B) Short keys only
C) Side channels
D) Hash weaknesses

Q5. Playfair works on:
A) Words
B) Bits
C) Digraphs
D) Single letters

Q6. Same row in Playfair encrypts by taking:

A) Diagonal

B) Above

C) Right

D) Left

Q7. Encrypt X with k=5:

A) B

B) E

C) D

D) C

Q8. ROT13 applied twice gives:

A) Random text

B) Error

C) Same ciphertext

D) Original text

Q9. Most common English letter:

A) E

B) O

C) A

D) T

Q10. Why does BALLOON become BA LX LO ON?

A) Matrix rule

B) Doubles are split with X

C) Random

D) Key needs it

12. Scenario-based questions (5)

C1. An attacker finds the commonest ciphertext letter is H. Estimate the Caesar key.

C2. Is a 26!-key cipher safe? Argue.

C3. You need a quick hand-computable cipher for a game. Choose and justify.

C4. Decrypt "KHOOR" by brute force.

C5. Why would a club list encoded with Caesar be unsafe?

13. Practical and problem-solving exercises (3)

X1. Encrypt MEET AT DAWN with k=9.

X2. Build a Playfair matrix for SECURITY and encrypt SAFE.

X3. Write a Python function that tries all Caesar keys.

Model answers (selected short questions)

S4. 25 non-trivial keys.

S7. E is the most frequent letter in English text.

Detailed model answers (selected long questions)

L1. Convert each letter to its position (A=0), add k, reduce mod 26 and convert back. For k=5, CRYPTO becomes HWDUYT. Decryption subtracts k mod 26. Only 25 keys exist, so brute force is trivial.

14. References

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

Paar, C., & Pelzl, J. (2010). *Understanding cryptography: A textbook for students and practitioners*. Springer.
Pfleeger, C. P., Pfleeger, S. L., & Margulies, J. (2015). *Security in computing* (5th ed.). Prentice Hall.
Python Software Foundation. (n.d.). *Python 3 documentation*. https://docs.python.org/3/

Module 05 Polyalphabetic and Transposition Ciphers

Introduction. Monoalphabetic ciphers fail because one plaintext letter always gives the same ciphertext letter. Polyalphabetic ciphers use several substitution alphabets in rotation, flattening letter frequencies. We also study a classical transposition cipher, Rail Fence.

Why this topic matters. These ciphers show two design ideas still used in modern cryptography: varying the transformation (key-dependent) and rearranging positions (permutation).

Real-world relevance. The idea of key-stream generation in Vigenere leads directly to stream ciphers (Module 7).

Learning objectives	Expected learning outcomes
• Explain polyalphabetic substitution. • Encrypt and decrypt with the Vigenere cipher. • Explain the autokey system and distinguish plaintext from ciphertext autokey. • Apply the Rail Fence cipher with different rail counts. • Evaluate strengths and weaknesses of classical ciphers.	• Complete Vigenere, autokey and Rail Fence problems manually and by program.

Prerequisites	Estimated learning time
Modules 3 and 4.	About 6 hours including practicals.

5.1 Vigenere Cipher

In simple words. Use a keyword; each letter of the keyword gives a different Caesar shift for the matching letter of the message.

Formal definition. The Vigenere cipher is a polyalphabetic substitution cipher in which the ith plaintext letter is shifted by the value of the ith key letter (repeated cyclically): $C_i = (P_i + K_{i \bmod m}) \bmod 26$.

Everyday analogy. Imagine a combination lock whose dial setting changes with every letter according to a repeating password.

How it works. Write the key repeated under the plaintext, add letter values mod 26. Decrypt by subtracting.

Figure 12 | Vigenere table (tabula recta)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
L	L	M	N	O	**P**	Q	R	S	T	U	V	W	X	Y
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A

Vigenere table (first 14 rows/columns)

Plain letter picks the column

Key letter picks the row

Intersection = cipher letter

Example: P = E, K = L -> C = P

(row L, column E)

Figure 12. *Vigenere table (tabula recta)*

Term	Meaning
Key stream	Repeated key aligned with the message.
Tabula recta	26x26 table of shifted alphabets.
Key length m	Number of letters in the repeating key.

Practical applications. Used for centuries and once called "le chiffre indechiffrable".

> **Common misconception**
> Vigenere is unbreakable.

Security implications. Kasiski examination and index of coincidence reveal the key length, then each position is a Caesar cipher.

Worked example (verified). Plaintext **ATTACKATDAWN**, key **LEMON**. Aligned key: **LEMONLEMONLE**. Encryption by letters: A+L = L, T+E = X, T+M = F, A+O = O, C+N = P, K+L = V, ... Ciphertext = **LXFOPVEFRNHR**. Decryption subtracts the same key stream and returns ATTACKATDAWN.

5.2 Autokey System

In simple words. Autokey avoids the repeating key: after the short starter key, the message itself becomes the key.

Formal definition. In an autokey cipher the key stream consists of an initial primer followed by the plaintext (plaintext autokey) or by the ciphertext (ciphertext autokey).

Everyday analogy. A password that continues by copying your own message, so it never repeats.

How it works. Plaintext autokey: key stream = primer + plaintext. The receiver decrypts letter by letter and appends each recovered plaintext letter to the key stream. Ciphertext autokey: key stream = primer + ciphertext, so the eavesdropper also sees the key stream.

Term	Meaning
Primer	Short initial key.
Plaintext autokey	Key stream continues with plaintext.
Ciphertext autokey	Key stream continues with ciphertext.

> **Common misconception**
> Autokey is secure because the key never repeats.

Security implications. The key stream is derived from known-language text, so statistical attacks and guessing the primer still work.

Worked example (verified). Plaintext ATTACKATDAWN, primer **QUEENLY**. Plaintext autokey ciphertext: **QNXEPVYTWTWP**. Ciphertext autokey (primer then ciphertext) gives **QNXEPVYJQXAC**. They agree for the first seven letters (same primer) and differ afterwards because the key streams diverge.

5.3 Rail Fence Cipher

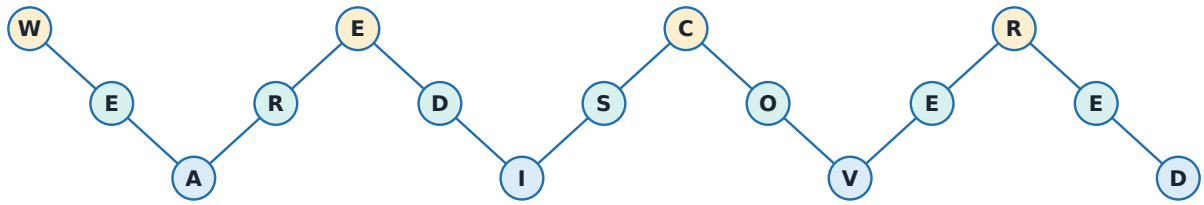
In simple words. Write the message in a zigzag across several rows, then read the rows one after another.

Formal definition. The Rail Fence cipher is a transposition cipher in which plaintext is written diagonally on a number of rails (rows) in a zigzag pattern and the ciphertext is read row by row.

Everyday analogy. Weaving a thread up and down a fence, then unravelling it row by row.

How it works. Choose n rails; write letters going down then up; read each rail from left to right. The key is the number of rails. Decryption recreates the zigzag pattern, fills it row by row, and reads it diagonally.

Figure 13 | Rail Fence zigzag with 3 rails



Rows read in order: WECR + ERDSOEE + AIVD = WECRERDSOEEAIVD (first 15 letters)

Figure 13. *Rail Fence zigzag with 3 rails*

Term	Meaning
Rail	One row of the zigzag.
Period	Pattern repeats every 2(n-1) letters.

Common misconception
More rails always mean strong security.

Security implications. There are very few possible keys (at most one per length), so brute force finishes at once.

Worked example (verified). Plaintext WEAREDISCOVEREDFLEEATONCE.

Table 17. Rail Fence results

Rails	Ciphertext
2	WAEICVRDLETNEERDSOEEFEAOC
3	WECRLTEERDSOEEFEAOCAIVDEN
4	WIREEEDSEEEACAECVDLTNROFO

5.4 Strengths and Weaknesses of Classical Ciphers

Table 18. Classical cipher evaluation

Cipher	Strength	Weakness
Caesar	Simple	25 keys
Substitution	Large key space	Frequency analysis
Playfair	Hides single letters	Digraph statistics
Vigenere	Flattens frequencies	Key-length attacks
Autokey	Non-repeating key	Known-text attacks
Rail Fence	Easy to use	Few keys; letters unchanged

Case Study: Historical message found in a school archive

Background	A school archivist finds an old coded note.
Organisation and environment	School archive with limited digital tools.
Information assets	The note itself and its historical value.
Security problem	Meaning unknown.
Threats and vulnerabilities	No threat; analysis task. Vulnerability of the cipher: repeating key.
Potential impact	Historical interest only.
Detection	Letter repetition suggests Vigenere with a short key.
Investigation	Repeated trigrams at distance 15 suggest key length 5 or 3.
Response	Split text into 5 columns and treat each as a Caesar cipher.
Prevention	Teaches why repeating keys are dangerous.
Lessons learned	Key length discovery turns Vigenere into several Caesar ciphers. Fictional scenario.

Practical 9: Vigenere and Autokey Ciphers

Learning objective	Encrypt and decrypt with Vigenere and autokey.
Software and tools	Python 3.
Environment	Terminal.
Background	Programs reduce arithmetic errors.

Step-by-step procedure

1. Save vigenere.py.
2. Run it.
3. Compute ATTACKATDAWN by hand and compare.
4. Try a different key.

Program: vigenere.py

```
# Vigenere and Autokey ciphers
def vig(text, key, d=1):
    out = ""
    for i, ch in enumerate(text):
        k = ord(key[i % len(key)]) - 65
        out += chr((ord(ch) - 65 + d * k) % 26 + 65)
    return out
def autokey_enc(text, key):
    stream = (key + text)[:len(text)]            # key followed by the plaintext itself
    return "".join(chr((ord(p) + ord(k) - 130) % 26 + 65) for p, k in zip(text, stream))
def autokey_dec(ct, key):
    stream, out = key, ""
    for i, c in enumerate(ct):
        p = chr((ord(c) - ord(stream[i])) % 26 + 65)
        out += p; stream += p
    return out
pt = "ATTACKATDAWN"
c1 = vig(pt, "LEMON");        print("Vigenere ", c1, vig(c1, "LEMON", -1))
c2 = autokey_enc(pt, "QUEENLY"); print("Autokey  ", c2, autokey_dec(c2, "QUEENLY"))
```

Output

```
Vigenere  LXFOPVEFRNHR ATTACKATDAWN
Autokey   QNXEPVYTWTWP ATTACKATDAWN
```

Output interpretation	Both ciphers decrypt back to ATTACKATDAWN; the outputs differ after the primer length.
Verification procedure	Match the program output against the hand calculation.
Common errors	Spaces or lowercase in input.
Troubleshooting	Remove spaces and use uppercase.
Security precautions	Sample text only.
Practical conclusion	Key-stream design determines strength.
Student questions	1) Why does Vigenere resist simple frequency analysis? 2) What is the key stream in autokey? 3) Why is a short repeating key a weakness?

Practical 10: Rail Fence Cipher

Learning objective	Encrypt and decrypt with 2, 3 and 4 rails.
Software and tools	Python 3.
Environment	Terminal.
Background	Transposition changes positions only.

Step-by-step procedure

1. Save railfence.py.
2. Run it.
3. Draw the 3-rail zigzag on paper and compare.

Program: railfence.py

```
# Rail Fence (zigzag) transposition cipher
def pattern(rails, n):
    r, d, res = 0, 1, []
    for _ in range(n):
        res.append(r)
        if rails > 1:
            if r == 0: d = 1
            elif r == rails - 1: d = -1
            r += d
    return res
def encrypt(text, rails):
    pat = pattern(rails, len(text))
    return "".join(text[i] for i in sorted(range(len(text)), key=lambda i: (pat[i], i)))
def decrypt(ct, rails):
    pat = pattern(rails, len(ct))
    order = sorted(range(len(ct)), key=lambda i: (pat[i], i))
    out = [""] * len(ct)
    for k, i in enumerate(order): out[i] = ct[k]
    return "".join(out)
pt = "WEAREDISCOVEREDFLEEATONCE"
for rails in (2, 3, 4):
    c = encrypt(pt, rails); print(rails, "rails:", c, decrypt(c, rails) == pt)
```

Output

```
2 rails: WAEICVRDLETNEERDSOEEFEAOC True
3 rails: WECRLTEERDSOEEFEAOCAIVDEN True
4 rails: WIREEEDSEEEACAECVDLTNROFO True
```

Output interpretation	The ciphertext letters are the same letters as the plaintext in a new order.
Verification procedure	The program prints True when decryption restores the plaintext.
Common errors	Mixing up row reading order.
Troubleshooting	Draw the pattern.
Security precautions	Sample text only.
Practical conclusion	Rails act as the key.
Student questions	1) Why is the letter frequency unchanged? 2) How many rails make sense for 10 letters? 3) How would you break it?

Module 5 Revision

1. Module summary

Vigenere uses repeating key letters as different Caesar shifts. Autokey extends the key with the message. Rail Fence is a transposition cipher whose key is the number of rails. All remain breakable.

2. Key definitions

Term	Definition
Polyalphabetic	Several substitution alphabets.
Primer	Initial autokey key.
Transposition	Rearranging positions.

3. Important terminology

Tabula recta, Kasiski, key stream, rail.

4. Comparison table

Table 19. Comparison for Module 5

	Vigenere	Autokey	Rail Fence
Type	Substitution	Substitution	Transposition
Key	Repeating word	Primer plus text	Rails count

5. Quick revision notes

- C = (P + K) mod 26 with K cycling.
- Autokey key stream = primer + text.

6. Memory techniques

- Rail Fence: "Zig, zag, read the rows".

7. Important formulas

- Vigenere: C = (P + K) mod 26
- Decrypt: P = (C - K) mod 26

8. Common examination mistakes

- Key not repeated correctly.
- Reading rails in wrong order.

Module 5 Question Bank

9. Short questions (15)

S1. Define polyalphabetic cipher.
S2. Write the Vigenere formula.
S3. What is the tabula recta?
S4. What is a key stream?
S5. What is the autokey cipher?
S6. Differentiate plaintext and ciphertext autokey.
S7. What is a primer?
S8. What is Rail Fence?
S9. What is the key of Rail Fence?
S10. What is Kasiski examination?
S11. Why is Vigenere stronger than Caesar?
S12. Name one weakness of Vigenere.
S13. Does Rail Fence change letter frequencies?
S14. What is a period in Rail Fence?
S15. Which is stronger: Caesar or Vigenere?

10. Long questions (8)

L1. Explain Vigenere with a worked example.
L2. Explain how Vigenere is cryptanalysed.
L3. Explain autokey encryption and decryption.
L4. Compare the two autokey variants.
L5. Explain Rail Fence encryption and decryption.
L6. Discuss strengths and weaknesses of classical ciphers.
L7. Explain why transposition and substitution combine well.
L8. Explain why classical ciphers fail today.

11. Multiple-choice questions (10)

Q1. Vigenere with key length 5 is made of:

A) Five keys of 26!
B) Five Caesar ciphers
C) One Caesar
D) Five hashes

Q2. Encrypt A with key L:

A) M
B) L
C) A
D) K

Q3. Rail Fence is:

A) Transposition
B) Block cipher
C) Substitution
D) Hash

Q4. Autokey key stream is:

A) Constant
B) Hash
C) Random bits
D) Primer plus text

Q5. Kasiski helps find:

A) Padding
B) Key length
C) Hash
D) The message

Q6. Rail Fence key is:

A) Matrix
B) Prime
C) Keyword
D) Number of rails

Q7. Which keeps letter frequencies unchanged?

A) Autokey
B) Vigenere
C) Rail Fence
D) Caesar

Q8. Polyalphabetic ciphers aim to:

A) Flatten frequencies
B) Compress
C) Shorten text
D) Hash data

Q9. Decrypting autokey requires:

A) No key
B) Public key
C) Primer
D) Hash

Q10. Vigenere is:

A) Historical

B) A hash

C) Modern secure

D) Asymmetric

12. Scenario-based questions (5)

C1. Intercepted text shows repeats at distance 12. What does this suggest?

C2. Why does Rail Fence leave the frequency profile intact?

C3. Design a cipher that combines Vigenere and Rail Fence. Is it secure?

C4. Why is autokey weaker if the primer is guessable?

C5. A company wants Vigenere for passwords. Advise.

13. Practical and problem-solving exercises (3)

X1. Encrypt WEAREDISCOVERED with Vigenere key SAFE.

X2. Encrypt HELLOWORLD with 3 rails.

X3. Write Python to produce the key stream for autokey.

Model answers (selected short questions)

S2. Ci = (Pi + Ki) mod 26 where K repeats cyclically.

S6. Plaintext autokey appends plaintext to the primer; ciphertext autokey appends ciphertext.

Detailed model answers (selected long questions)

L1. Write the key under the plaintext repeated; ATTACKATDAWN with LEMON gives LXFOPVEFRNHR by adding letter positions mod 26. Decryption subtracts.

14. References

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

Paar, C., & Pelzl, J. (2010). *Understanding cryptography: A textbook for students and practitioners*. Springer.

Pfleeger, C. P., Pfleeger, S. L., & Margulies, J. (2015). *Security in computing* (5th ed.). Prentice Hall.

Python Software Foundation. (n.d.). *Python 3 documentation*. https://docs.python.org/3/

Module 06
Hill Cipher and Cryptanalysis

Introduction. The Hill cipher is the first cipher in this course that treats several letters at once as a vector and uses linear algebra. We then study how classical ciphers are attacked, which explains why modern ciphers are designed so differently.

Why this topic matters. Matrix multiplication and modular inverses appear again in AES and RSA. Cryptanalysis teaches us to think like an attacker, which is the best way to judge any cipher.

Real-world relevance. Hill introduces matrix-based diffusion, an idea used in AES MixColumns.

Learning objectives	Expected learning outcomes
• Explain Hill cipher encryption and decryption. • Multiply matrices mod 26 and find modular inverses. • Check whether a key matrix is invertible. • Perform brute-force and frequency cryptanalysis on Caesar. • Explain why classical ciphers are unsuitable for modern high-security systems.	• Encrypt and decrypt with a 2x2 Hill key. • Break a Caesar ciphertext.

Prerequisites	Estimated learning time
Modules 3 to 5 and basic matrices.	About 6 hours.

6.1 Hill Cipher

In simple words. Group the message into small blocks, treat each block as a column of numbers and multiply by a secret matrix.

Formal definition. The Hill cipher encrypts m letters at once as C = K P mod 26, where K is an m x m invertible key matrix and P is the plaintext vector. Decryption uses K^{-1} mod 26.

Everyday analogy. A kitchen blender: each output letter is a recipe mixing all input letters.

How it works. Take pairs of letters, convert to numbers, multiply by K, reduce mod 26. For decryption compute $K^{-1} = (\det K)^{-1}$ x adj(K) mod 26.

Figure 14 | Hill cipher: encrypting HE with a 2x2 key

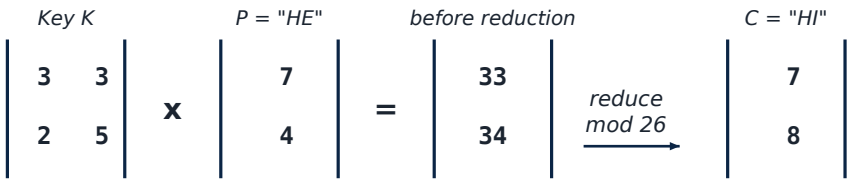


Figure 14. *Hill cipher: encrypting HE with a 2x2 key*

Term	Meaning
Key matrix K	Secret m x m matrix of integers mod 26.

Term	Meaning
Determinant	det K = ad - bc for a 2x2 matrix.
Invertible mod 26	Exists K^{-1} only if gcd(det K, 26) = 1.

Practical applications. Hill (1929) was the first practical cipher to operate on more than three letters at once.

Common misconception
Any matrix will work as a key.

Security implications. If gcd(det K, 26) is not 1 the matrix has no inverse and the message cannot be decrypted uniquely.

Worked example (verified)

Key K = [[3, 3], [2, 5]]. Plaintext HELP gives vectors HE = (7, 4) and LP = (11, 15).

HE: 3x7 + 3x4 = 33 = 7 (mod 26) = H; 2x7 + 5x4 = 34 = 8 (mod 26) = I. So HE becomes **HI**.

LP: 3x11 + 3x15 = 78 = 0 (mod 26) = A; 2x11 + 5x15 = 97 = 19 (mod 26) = T. So LP becomes **AT**. Ciphertext = **HIAT**.

Decryption: det K = 3x5 - 3x2 = 9. The inverse of 9 mod 26 is 3 because 9x3 = 27 = 1 (mod 26). adj K = [[5, -3], [-2, 3]]. Multiply by 3 and reduce: K^{-1} = [[15, 17], [20, 9]] (matches the program: [[15, 17], [20, 9]]). Applying it to HIAT returns **HELP**.

Invertibility check
Key [[2, 4], [1, 2]]: det = 0, impossible. Key [[2, 3], [4, 5]]: det = -2 = 24 mod 26; gcd(24, 26) = 2, not 1, so it is rejected. Key [[3, 3], [2, 5]]: det = 9, gcd(9, 26) = 1, accepted.

6.2 Introduction to Cryptanalysis

In simple words. Cryptanalysis tries to recover plaintext or keys without being given the key.

Formal definition. Cryptanalysis is the study of methods for obtaining the meaning of encrypted information without access to the secret key.

Everyday analogy. A locksmith testing which keys fit, listening to the lock, or noticing worn buttons.

How it works. Attack models depend on what the attacker knows.

Term	Meaning
Ciphertext-only	Attacker has ciphertext only.
Known-plaintext	Attacker has some plaintext-ciphertext pairs.
Chosen-plaintext	Attacker can encrypt chosen text.
Chosen-ciphertext	Attacker can decrypt chosen ciphertext.
Brute force	Try every key.
Frequency analysis	Use letter statistics.

Security implications. Hill is vulnerable to known-plaintext attacks: with enough pairs the key matrix is found by solving linear equations.

Caesar cipher cryptanalysis

Brute force: try all 25 keys and read the output; only one produces meaningful text. **Frequency analysis**: assume the most common ciphertext letter represents E, compute k = (position of that letter - 4) mod 26 and test. Both methods succeed instantly, demonstrating that a key space of 25 offers no security.

Why classical ciphers are not suitable for modern systems

- Small or structured key spaces and exploitable statistics.
- They preserve language patterns (frequency, digraphs, repeated words).
- Linear structure (Hill) falls to known-plaintext attacks.
- No integrity protection: an attacker can alter ciphertext undetected.
- Modern computers test billions of keys per second.

Case Study: Breaking a student-made cipher in a lab competition

Background	An ADP class runs a cryptography challenge.
Organisation and environment	University lab with student teams.
Information assets	Challenge messages and scoring server.
Security problem	Team A publishes a Hill-encrypted message.
Threats and vulnerabilities	Weakness: known header "DEAR" at message start.
Potential impact	Key recovered by other teams.
Detection	A team notices the standard greeting.
Investigation	They solve linear equations using the known pair.
Response	Team A changes to a modern cipher library.
Prevention	Never assume the attacker lacks known text.
Lessons learned	Predictable text enables known-plaintext attacks. Fictional scenario.

Practical 11: Hill Cipher Calculations

Learning objective	Encrypt and decrypt with a 2x2 Hill key and verify invertibility.
Software and tools	Python 3.
Environment	Terminal.
Background	Programs prevent arithmetic mistakes.

Step-by-step procedure

1. Save hill.py.
2. Run it.
3. Change the key to [[5,8],[17,3]] and test.
4. Try a non-invertible key and read the error.

Program: hill.py

```
# Hill cipher (2x2) with modular inverse of the key matrix
import math
def inv_mod(a, m):
    for x in range(1, m):
        if (a * x) % m == 1: return x
    raise ValueError("matrix not invertible mod 26")
def hill(text, K, decrypt=False):
    det = (K[0][0] * K[1][1] - K[0][1] * K[1][0]) % 26
    assert math.gcd(det, 26) == 1, "key matrix not invertible"
    if decrypt:
        di = inv_mod(det, 26)
        K = [[K[1][1] * di % 26, -K[0][1] * di % 26], [-K[1][0] * di % 26, K[0][0] * di % 26]]
        print("Inverse key matrix:", K)
    out = ""
    for i in range(0, len(text), 2):
        v = [ord(text[i]) - 65, ord(text[i + 1]) - 65]
        out += "".join(chr((K[r][0] * v[0] + K[r][1] * v[1]) % 26 + 65) for r in range(2))
    return out
K = [[3, 3], [2, 5]]
c = hill("HELP", K); print("Cipher:", c)
print("Plain :", hill(c, K, decrypt=True))
```

Output

```
Cipher: HIAT
Inverse key matrix: [[15, 17], [20, 9]]
Plain : HELP
```

Output interpretation	Encryption gives HIAT; the inverse key matrix is [[15,17],[20,9]].
Verification procedure	Multiply K by K^-1 mod 26 to obtain the identity.
Common errors	Odd-length plaintext.
Troubleshooting	Pad with X.
Security precautions	Sample text only.
Practical conclusion	Matrix algebra can encrypt blocks.
Student questions	1) Why must gcd(det,26)=1? 2) What is the inverse of 9 mod 26? 3) Which attack breaks Hill?

Practical 35: Caesar Cipher Cryptanalysis by Frequency Analysis

Learning objective	Recover a Caesar key by frequency analysis.
Software and tools	Python 3.
Environment	Terminal.
Background	Statistics reveal keys.

Step-by-step procedure

1. Save freq_lab.py.
2. Run it.
3. Replace the message with another sentence.

Program: freq_lab.py

```
# Frequency analysis: recover a Caesar key by assuming the commonest letter is E
from collections import Counter
def shift(t, k): return "".join(chr((ord(c) - 65 + k) % 26 + 65) if c.isalpha() else c for c in t)
secret = shift("THE SECURITY OF INFORMATION DEPENDS ON THE SECRECY OF THE KEY AND NOT THE
  ↪ ALGORITHM", 11)
print("Ciphertext:", secret)
counts = Counter(c for c in secret if c.isalpha())
print("Top letters:", counts.most_common(4))
guess = (ord(counts.most_common(1)[0][0]) - ord("E")) % 26
print("Guessed key:", guess)
print("Recovered  :", shift(secret, -guess))
```

Output

```
Ciphertext: ESP DPNFCTEJ ZQ TYQZCXLETZY OPAPYOD ZY ESP DPNCPNJ ZQ ESP VPJ LYO YZE ESP LWRZCTESX
Top letters: [('P', 10), ('E', 8), ('Z', 7), ('Y', 6)]
Guessed key: 11
Recovered  : THE SECURITY OF INFORMATION DEPENDS ON THE SECRECY OF THE KEY AND NOT THE ALGORITHM
```

Output interpretation	The commonest ciphertext letter P maps to E, giving key 11.
Verification procedure	Decrypt with the recovered key.
Common errors	Short texts give wrong guesses.
Troubleshooting	Try the next most common letter.
Security precautions	Sample text only.
Practical conclusion	Statistical leakage is fatal for classical ciphers.
Student questions	1) Why does the guess fail with short text? 2) How would you extend this to Vigenere? 3) What defence do modern ciphers use?

Module 6 Revision

1. Module summary

Hill encrypts blocks with C = K P mod 26 and requires an invertible key matrix. Cryptanalysis uses brute force, statistics and known plaintext. Classical ciphers do not meet modern security needs.

2. Key definitions

Term	Definition
Hill cipher	Matrix-based block cipher.
Known-plaintext attack	Attack using known plaintext-ciphertext pairs.
Brute force	Exhaustive key search.

3. Important terminology

Determinant, adjugate, modular inverse, attack model.

4. Comparison table

Table 20. Comparison for Module 6

Attack	Needs	Breaks
Brute force	Small key space	Caesar
Frequency analysis	Enough text	Substitution
Known plaintext	Pairs	Hill

5. Quick revision notes

- Invertible if gcd(det,26)=1.
- K^-1 = det^-1 x adj(K) mod 26.

6. Memory techniques

- Hill = "Matrix Mixes Message".

7. Important formulas

- C = K P mod 26
- P = K^-1 C mod 26
- det = ad - bc

8. Common examination mistakes

- Forgetting mod 26 on negatives.
- Using non-invertible key.

Module 6 Question Bank

9. Short questions (15)

S1. Define Hill cipher.
S2. What is the encryption formula?
S3. What is the decryption formula?
S4. State the invertibility condition.
S5. Compute det of [[3,3],[2,5]].
S6. What is 9^-1 mod 26?
S7. What is a known-plaintext attack?
S8. What is brute force?
S9. Why does frequency analysis work against Caesar ciphertext?
S10. Why is Hill vulnerable to known plaintext?
S11. Why pad with X?
S12. What is an adjugate matrix?
S13. What does mod 26 do to negatives?
S14. Name two ciphertext-only attacks.
S15. Why are classical ciphers not used today?

10. Long questions (8)

L1. Explain the Hill cipher with an example.
L2. Explain how to find a Hill inverse key.
L3. Discuss when a key matrix is valid.
L4. Explain brute-force cryptanalysis of Caesar.
L5. Explain frequency analysis of Caesar.
L6. Describe four attack models.
L7. Explain why Hill is vulnerable to known plaintext.
L8. Explain why classical ciphers are insecure.

11. Multiple-choice questions (10)

Q1. Hill cipher key must be:
A) Symmetric
B) Any matrix
C) Identity only
D) Invertible mod 26

Q2. det [[3,3],[2,5]] =
A) 15
B) 6
C) 9
D) 21

Q3. 9^-1 mod 26 =
A) 13
B) 3
C) 5
D) 9

Q4. Known-plaintext attack uses:
A) Plaintext/ciphertext pairs
B) Hash
C) Only ciphertext
D) Public key

Q5. Caesar brute force tries:
A) 100 keys
B) 26! keys
C) 25 keys
D) 2^56 keys

Q6. Hill encrypts:
A) Blocks of letters
B) Words
C) Single letters
D) Bits only

Q7. A matrix with gcd(det,26)=2:
A) Symmetric
B) Invalid
C) Faster
D) Valid

Q8. Frequency analysis assumes common letter is:
A) X
B) Z
C) E
D) Q

Q9. Classical ciphers lack:
A) Alphabet
B) Letters
C) Keys
D) Integrity and strong structure

Q10. Ciphertext-only attack requires:

A) Oracle

B) Key

C) Plaintext

D) Only ciphertext

12. Scenario-based questions (5)

C1. Choose a key matrix: [[2,4],[1,2]] or [[3,3],[2,5]]. Why?

C2. Messages always begin DEAR. Which attack is possible?

C3. Why does Hill diffusion not save it?

C4. A team brute-forces Caesar: how many tries at most?

C5. Is Hill with 3x3 keys safe against known-plaintext?

13. Practical and problem-solving exercises (3)

X1. Encrypt SECURITY with key [[3,3],[2,5]] and decrypt.

X2. Compute K^-1 for [[5,8],[17,3]].

X3. Write a program to test all 25 shifts.

Model answers (selected short questions)

S4. gcd(det K, 26) = 1.

S6. 9 x 3 = 27 = 1 (mod 26) so the inverse is 3.

Detailed model answers (selected long questions)

L2. For K = [[a,b],[c,d]], compute det = ad-bc mod 26, find its modular inverse, form adj = [[d,-b],[-c,a]], multiply by the inverse and reduce mod 26. For [[3,3],[2,5]] the result is [[15,17],[20,9]].

14. References

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

Paar, C., & Pelzl, J. (2010). *Understanding cryptography: A textbook for students and practitioners*. Springer.

Pfleeger, C. P., Pfleeger, S. L., & Margulies, J. (2015). *Security in computing* (5th ed.). Prentice Hall.

Python Software Foundation. (n.d.). *Python 3 documentation*. https://docs.python.org/3/

Module 07
Stream Ciphers and the Data Encryption Standard

Introduction. Modern symmetric ciphers work on bits, not letters. We meet two historic designs: RC4, a fast stream cipher that is now broken, and DES, the first public encryption standard. Both are studied for understanding and are not to be chosen for new systems.

Why this topic matters. RC4 teaches the key-stream idea; DES teaches the Feistel structure that influenced decades of cipher design.

Real-world relevance. RC4 was widely used in Wi-Fi (WEP) and web encryption; DES protected banking networks for decades.

Learning objectives	Expected learning outcomes
• Differentiate stream and block ciphers. • Explain RC4 key scheduling and key-stream generation. • State known RC4 weaknesses. • Describe the DES Feistel structure and its 16 rounds. • Explain DES key generation, decryption and limitations.	• Run an educational RC4 and DES demonstration and verify against published test vectors. • Explain why RC4 and DES must not be used in new production systems.

Prerequisites	Estimated learning time
Modules 3 to 6; binary and XOR.	About 8 hours.

7.1 Stream and Block Cipher Fundamentals

In simple words. A stream cipher encrypts bit by bit using a running stream of secret bits. A block cipher encrypts fixed-size blocks.

Formal definition. A stream cipher combines plaintext with a pseudorandom key stream, typically by XOR. A block cipher is a keyed permutation operating on fixed-length blocks, used with a mode of operation for longer messages.

Everyday analogy. Stream: a tap dripping coloured ink on a flowing paper roll. Block: stamping a pattern on each page of a book.

How it works. Stream ciphers: $C_i = P_i$ XOR K_i. The same operation decrypts. Key-stream **must never be reused** with the same key, because XORing two ciphertexts removes the key.

Term	Meaning
XOR	Bitwise exclusive OR: 1 if bits differ.
Key stream	Pseudorandom bytes generated from the key.
Block size	Number of bits per block (DES 64, AES 128).
Mode of operation	Method for processing many blocks (CBC, CTR, GCM).

Practical applications. Stream ciphers: mobile voice, Wi-Fi. Block ciphers: disk and file encryption, TLS.

Security implications. Unauthenticated encryption can be altered bit by bit; use authenticated encryption in practice.

Table 21. Stream versus block ciphers

Feature	Stream cipher	Block cipher
Unit	Bit or byte	Block (64 or 128 bits)
Speed	Very fast	Fast
Main risk	Key-stream reuse	Weak modes, short blocks
Examples	RC4, ChaCha20	DES, AES

7.2 RC4

In simple words. RC4 shuffles a list of 256 numbers using the key, then keeps shuffling to produce secret bytes that are XORed with the message.

Formal definition. RC4 is a variable-key-length stream cipher by Ron Rivest (1987) that maintains a 256-byte state array S and produces a pseudorandom key stream via two algorithms: KSA and PRGA.

Everyday analogy. Shuffling a deck of 256 numbered cards using the key as instructions, then drawing cards one at a time as secret numbers.

How it works. KSA: initialise S = 0..255, j = 0; for i = 0..255: j = (j + S[i] + key[i mod len]) mod 256; swap S[i], S[j]. PRGA: i = j = 0; repeat: i = (i+1) mod 256; j = (j + S[i]) mod 256; swap S[i], S[j]; output S[(S[i] + S[j]) mod 256].

Figure 15 | RC4 process: key scheduling, key stream and XOR

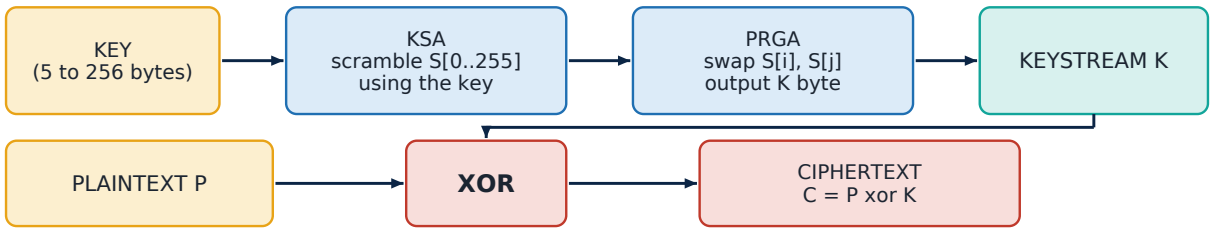


Figure 15. *RC4 process: key scheduling, key stream and XOR*

Term	Meaning
KSA	Key Scheduling Algorithm.
PRGA	Pseudo-Random Generation Algorithm.
State S	Permutation of 0..255.

Practical applications. WEP Wi-Fi, early SSL/TLS, some file formats.

> **Common misconception**
> RC4 became safe once longer keys were used.

Security implications. Statistical biases in the first key-stream bytes and key-related attacks (such as FMS) broke WEP; RFC 7465 prohibits RC4 in TLS.

Verified test vector. Key "Key", plaintext "Plaintext" gives ciphertext **BBF316E8D940AF0AD3** (published value BBF316E8D940AF0AD3). After KSA with key "Key", the first state entries are S[0..7] = [75, 51, 132, 157, 192, 200, 29, 168].

7.3 Data Encryption Standard (DES)

In simple words. DES takes 64 bits of data and a 56-bit key and scrambles the data through sixteen (16) rounds.

Formal definition. DES is a symmetric block cipher with a 64-bit block and 56 effective key bits (64 including 8 parity bits), built as a 16-round Feistel network (FIPS 46-3).

Everyday analogy. Sixteen coat-and-shuffle steps: each round swaps the two halves and mixes one half with the key.

How it works. Plaintext goes through Initial Permutation (IP); the block splits into L and R (32 bits each). Each round computes $L_i = R_{i-1}$, $R_i = L_{i-1}$ XOR $f(R_{i-1}, K_i)$. After round 16 the halves are swapped and the Final Permutation (IP^{-1}) is applied.

Figure 16 | DES Feistel round structure

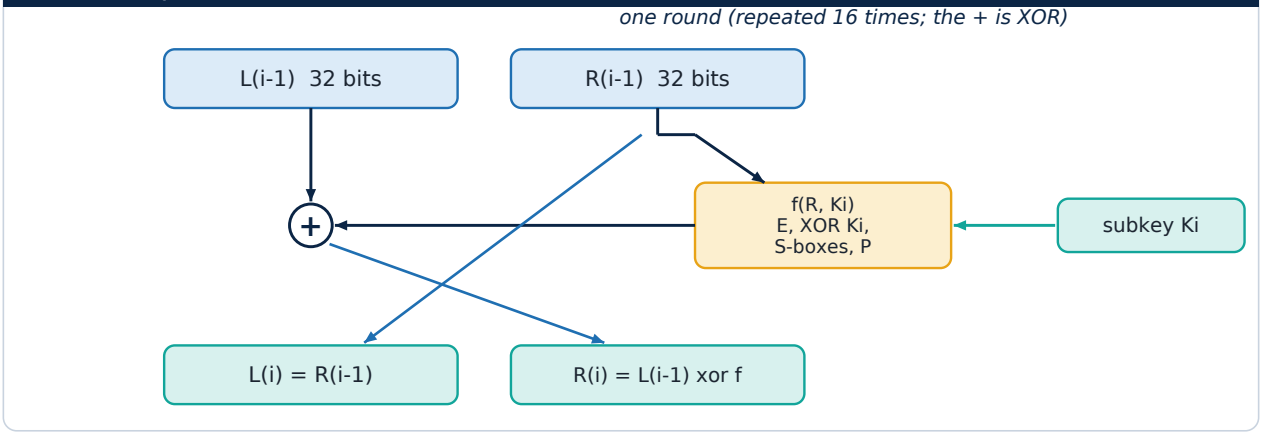


Figure 16. *DES Feistel round structure*

Term	Meaning
Feistel structure	Round structure that is invertible even if f is not.
IP / FP	Initial and final permutations.
Expansion E	Expands R from 32 to 48 bits.
S-boxes	Eight 6-to-4 bit substitution tables, the only non-linear step.
P-box	32-bit permutation after S-boxes.
Subkey Ki	48-bit round key.

Practical applications. Historic ATM networks and payment systems.

> **Common misconception**
> DES is broken because its structure is flawed.

Security implications. The structure is sound; the 56-bit key is simply too short for modern computing.

The round function f

f expands R to 48 bits (E), XORs with the 48-bit subkey, splits into eight 6-bit groups, replaces each by 4 bits using an S-box (the outer two bits select the row, inner four select the column), and finally permutes the 32 bits with P.

Key generation

The 64-bit key passes through Permuted Choice 1 (PC1), dropping parity bits to 56 bits split into C0 and D0 (28 bits each). In each of 16 rounds, C and D are rotated left by 1 or 2 bits (1 in rounds 1, 2, 9, 16), then Permuted Choice 2 (PC2) selects 48 bits as subkey K_i. **Verified example**: key 133457799BBCDFF1 gives K1 = `000110 110000 001011 101111 111111 000111 000001 110010`.

Decryption

Decryption uses the same circuit with subkeys in reverse order K16 ... K1. This is the elegance of Feistel networks.

Verified trace of the textbook example

Plaintext 0123456789ABCDEF, key 133457799BBCDFF1. After IP: L0 = CC00CCFF, R0 = F0AAF0AA.

Table 22. DES round trace (selected rounds)

Round	L	R
1	F0AAF0AA	EF4A6544
2	EF4A6544	CC017709
3	CC017709	A25C0BF4
15	C28C960D	43423234
16	43423234	0A4CD995

Final ciphertext: **85E813540F0AB405**, which matches the widely published value 85E813540F0AB405. Decrypting with reversed subkeys returns the plaintext.

Limitations of DES

- 56-bit key: a custom machine built by the Electronic Frontier Foundation (1998) found a DES key in a few days; today it takes hours or less with modern hardware.
- 64-bit block: vulnerable to collisions when large amounts of data are encrypted under one key (birthday bound).
- Triple DES (3DES) extended life but is slow and also deprecated; NIST withdrew DES (FIPS 46-3) in 2005.
- Use AES (Module 9) for all new systems.

Case Study: Legacy payment terminal using DES

Background	A small retail chain still uses old payment terminals.
Organisation and environment	Chain of 40 shops, terminals from 2005.
Information assets	Card data in transit to the processor.
Security problem	Terminals use single DES for encryption.
Threats and vulnerabilities	Threat: criminals with compute power. Vulnerability: short key, no authenticated encryption.
Potential impact	Card data exposed; fines and loss of trust.
Detection	Payment processor audit flags deprecated cryptography.
Investigation	Review shows firmware cannot support AES.
Response	Replace terminals with AES/TLS capable devices.
Prevention	Maintain a cryptographic inventory; retire obsolete algorithms on schedule.
Lessons learned	Old crypto is a hidden liability. Fictional scenario.

Practical 14: Educational RC4 Demonstration

Learning objective	Run RC4, verify the test vector and observe key-stream reuse.
Software and tools	Python 3.
Environment	Terminal, isolated learning environment.
Background	RC4 is shown only to teach stream ciphers.

Step-by-step procedure

1. Save rc4_demo.py.
2. Run it.
3. Compare with the published test vector.
4. Observe the keystream-reuse leak.

Program: rc4_demo.py

```
# RC4 educational demonstration. RC4 is BROKEN - never use it in real systems.
def ksa(key):
    S = list(range(256)); j = 0
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
        S[i], S[j] = S[j], S[i]
    return S
def prga(S, n):
    i = j = 0; out = []
    for _ in range(n):
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        out.append(S[(S[i] + S[j]) % 256])
    return out
def rc4(key, data):
    ks = prga(ksa(key), len(data))
    return bytes(d ^ k for d, k in zip(data, ks))
ct = rc4(b"Key", b"Plaintext")
print("Ciphertext:", ct.hex().upper())            # published test vector: BBF316E8D940AF0AD3
print("Decrypted :", rc4(b"Key", ct))
# Keystream reuse: XOR of two ciphertexts removes the keystream entirely
c1, c2 = rc4(b"K", b"ATTACK AT DAWN"), rc4(b"K", b"RETREAT AT TEN")
leak = bytes(a ^ b for a, b in zip(c1, c2))
print("c1 XOR c2 == p1 XOR p2 :", leak == bytes(a ^ b for a, b in zip(b"ATTACK AT DAWN", b"RETREAT
  ↪ AT TEN")))
```

Output

```
Ciphertext: BBF316E8D940AF0AD3
Decrypted : b'Plaintext'
c1 XOR c2 == p1 XOR p2 : True
```

Output interpretation	The ciphertext equals BBF316E8D940AF0AD3; XOR of two ciphertexts equals XOR of plaintexts.
Verification procedure	Decrypt and confirm the plaintext.
Common errors	Using strings instead of bytes.
Troubleshooting	Prefix literals with b.
Security precautions	Never use RC4 in real systems.
Practical conclusion	Key-stream reuse destroys secrecy.
Student questions	1) Why is key reuse fatal? 2) What do KSA and PRGA do? 3) Which standard prohibits RC4 in TLS?

Practical 15: DES Structure Simulation

Learning objective	Observe DES rounds, subkeys and decryption on the published test vector.
Software and tools	Python 3 and the supplied file des_lab.py (in the lab pack).
Environment	Terminal.
Background	DES is simulated for learning only.

Step-by-step procedure

1. Place des_lab.py in your working folder.
2. Run: python3 des_lab.py
3. Read the ciphertext, K1 and the first/last round values.
4. Decrypt and verify.

Output

```
85E813540F0AB405
0001101100000010111011111111110001110000011100 10
(1, 'F0AAF0AA', 'EF4A6544') (16, '43423234', '0A4CD995')
DES verified
```

Output interpretation	Ciphertext 85E813540F0AB405, K1 as shown and round 1 (F0AAF0AA, EF4A6544).
Verification procedure	Compare each value with the published worked example.
Common errors	Wrong table order.
Troubleshooting	Re-copy tables; the supplied file is verified.
Security precautions	Educational code, not for production.
Practical conclusion	Round-key explanation is clearer when you see real data.
Student questions	1) Why do decryption subkeys reverse? 2) Which step provides non-linearity? 3) Why is 56 bits too short?

Module 7 Revision

1. Module summary

Stream ciphers XOR a key stream with data; RC4 builds the key stream from a 256-byte state. DES is a 16-round Feistel cipher with 56-bit key. Both are obsolete; AES replaces them.

2. Key definitions

Term	Definition
RC4	Byte-oriented stream cipher.
DES	56-bit-key, 64-bit-block Feistel cipher.
Feistel	Round structure with swap and XOR.

3. Important terminology

KSA, PRGA, IP, S-box, subkey, bias.

4. Comparison table

Table 23. Comparison for Module 7

	RC4	DES	AES
Type	Stream	Block 64	Block 128
Key	40-2048 bits	56 bits	128/192/256
Status	Prohibited in TLS	Withdrawn	Current

5. Quick revision notes

- Never reuse a key stream.
- DES decrypt = reversed subkeys.

6. Memory techniques

- DES: "I Expand, Substitute, Permute: 16 times".

7. Important formulas

- Stream: C = P xor K
- Feistel: Li = Ri-1; Ri = Li-1 xor f(Ri-1, Ki)

8. Common examination mistakes

- Saying DES key is 64 effective bits.
- Confusing KSA with PRGA.

Module 7 Question Bank

9. Short questions (15)

S1. Define stream cipher.
S2. Define block cipher.
S3. What is XOR?
S4. What does KSA do?
S5. What does PRGA do?
S6. Name one RC4 weakness.
S7. What is a Feistel network?
S8. What is the DES block size?
S9. What is the effective DES key length?
S10. How many rounds does DES have?
S11. What is an S-box?
S12. What is IP in DES?
S13. Why does DES decryption reverse subkeys?
S14. What replaced DES?
S15. Why is key-stream reuse dangerous?

10. Long questions (8)

L1. Explain RC4 KSA and PRGA with steps.
L2. Discuss RC4 vulnerabilities.
L3. Explain DES architecture with a diagram.
L4. Explain the DES round function.
L5. Explain DES key generation.
L6. Explain DES decryption.
L7. Discuss limitations of DES.

L8. Compare stream and block ciphers.

11. Multiple-choice questions (10)

Q1. DES rounds:

A) 10
B) 14
C) 12
D) 16

Q2. DES effective key bits:

A) 64
B) 56
C) 40
D) 128

Q3. RC4 is a:

A) Signature
B) Hash
C) Block cipher
D) Stream cipher

Q4. Key-stream reuse lets attacker obtain:

A) Nothing
B) A hash
C) P1 xor P2
D) The key directly

Q5. The non-linear DES component is:

A) P-box
B) IP
C) Key rotate
D) S-boxes

Q6. DES block size:

A) 64
B) 56
C) 128
D) 32

Q7. RFC 7465 prohibits:

A) AES in TLS
B) RC4 in TLS
C) SHA-256
D) RSA

Q8. Feistel decryption uses:

A) Same subkey order
B) Random keys
C) Reverse subkey order
D) No keys

Q9. Which should a new system choose?

A) AES-GCM

B) DES

C) Caesar

D) RC4

Q10. Expansion E converts R to:

A) 32

B) 64

C) 48

D) 56

12. Scenario-based questions (5)

C1. Legacy system uses DES. Recommend actions.

C2. Two messages were encrypted with the same RC4 key. What is the risk?

C3. Why might a vendor still claim "3DES is fine"?

C4. Explain why DES design is considered sound despite insecurity.

C5. A WEP Wi-Fi network is found. Advise.

13. Practical and problem-solving exercises (3)

X1. Trace the first RC4 KSA swaps for key "Key" by hand (first 3).

X2. Compute the DES left rotations over 16 rounds.

X3. Write down the Feistel equations for rounds 1 and 2.

Model answers (selected short questions)

S7. A Feistel network splits the block, mixes one half with a keyed function and swaps halves each round.

S13. Feistel structure is symmetric, so reversing the subkey order undoes the encryption.

Detailed model answers (selected long questions)

L7. DES limitations: 56-bit key brute-forced in days in 1998 and hours today; 64-bit block gives collision risk for large data; the cipher is withdrawn by NIST. 3DES is slow and deprecated. Use AES instead.

14. References

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

Paar, C., & Pelzl, J. (2010). *Understanding cryptography: A textbook for students and practitioners*. Springer.

National Institute of Standards and Technology. (1999). *Data Encryption Standard (DES)* (FIPS PUB 46-3; withdrawn 2005).

Popov, A. (2015). *Prohibiting RC4 cipher suites* (RFC 7465). https://www.rfc-editor.org/info/rfc7465

Python Software Foundation. (n.d.). *Python 3 documentation*. https://docs.python.org/3/

Module 08
Modular Arithmetic and Risk Management

Introduction. This module has two parts. The first supplies the mathematics (congruence, greatest common divisor, inverses, modular exponentiation) needed for Hill, AES, RSA and signatures. The second introduces risk management: how organisations decide what to protect and how much to spend.

Why this topic matters. RSA cannot be understood without modular inverses; security investment cannot be justified without risk analysis.

Real-world relevance. Modular inverses are used in RSA key generation; risk registers are used in every security audit.

Learning objectives	Expected learning outcomes
• Compute with congruences and modular operations. • Apply the Euclidean and Extended Euclidean algorithms. • Find multiplicative inverses. • Identify, analyse and treat risks. • Calculate risk exposure and risk reduction leverage.	• Solve modular problems with full working. • Create a risk register for a fictional organisation.

Prerequisites	Estimated learning time
Modules 3 and 6.	About 7 hours.

8.1 Modular Arithmetic and Congruence

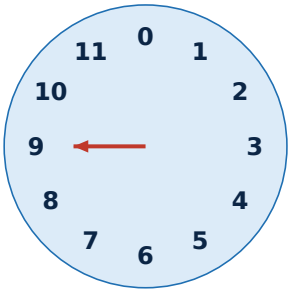
In simple words. Modular arithmetic is clock arithmetic: after reaching the modulus, we start again from 0.

Formal definition. Integers a and b are congruent modulo n, written $a \equiv b \pmod{n}$, if n divides (a - b). The notation a mod n denotes the remainder r, with $0 \leq r < n$, when a is divided by n.

Everyday analogy. A 12-hour clock: 9 hours after 5 o'clock is 2 o'clock, because $14 \bmod 12 = 2$.

How it works. Addition, subtraction and multiplication may be done before or after reduction: $(a + b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n$, and likewise for subtraction and multiplication. Division requires a multiplicative inverse.

Figure 17 | Modular arithmetic as a clock



Clock arithmetic: numbers wrap

(mod 12 shown; ciphers use mod 26)

9 + 5 = 14 = 2 (mod 12)

23 mod 26 = 23

27 mod 26 = 1

-3 mod 26 = 23

a = b (mod n) means n divides (a - b)

Figure 17. *Modular arithmetic as a clock*

Term	Meaning
Modulus n	The number at which values wrap around.
Residue	The remainder in 0..n-1.
Congruence	Same remainder modulo n.

Practical applications. Cryptography, hashing, check digits, calendars.

Common misconception
Modular division is ordinary division.

Security implications. Division means multiplying by an inverse, which exists only when gcd(a, n) = 1.

Table 24. Modular operations

Operation	Example	Result
Addition	(17 + 20) mod 26	11
Subtraction	(5 - 12) mod 26	19
Multiplication	(15 x 9) mod 26	5
Exponentiation	5^{13} mod 23	21
Congruence	38 ≡ 12 (mod 26)	True: 26 divides 26

Modular exponentiation: square and multiply

Computing a^e mod n directly produces enormous numbers. Instead process the bits of e from the left: square the running result, and multiply by a when the bit is 1. For 5^{13} mod 23, 13 = 1101 in binary.

Table 25. Square-and-multiply for 5^13 mod 23

Step	Exponent bit	Computation	Result
1	1	square then multiply by 5	5
2	1	square then multiply by 5	10
3	0	square then no multiply	8
4	1	square then multiply by 5	21

8.2 GCD, Euclidean and Extended Euclidean Algorithms

In simple words. The GCD is the largest number that divides both numbers. The Euclidean algorithm finds it by repeated division.

Formal definition. The greatest common divisor gcd(a, b) is the largest integer dividing both. Euclid's algorithm uses gcd(a, b) = gcd(b, a mod b) until the remainder is 0. The Extended Euclidean algorithm also finds integers x, y with ax + by = gcd(a, b).

Everyday analogy. Cutting the largest possible equal square tiles to cover a rectangular floor exactly.

How it works. Example gcd(1071, 462): 1071 = 2x462 + 147; 462 = 3x147 + 21; 147 = 7x21 + 0, so the gcd is **21**. Extended example: 240x(-9) + 46x47 = 2, hence gcd(240, 46) = 2 with x = -9, y = 47.

Term	Meaning
Coprime	gcd(a, b) = 1.
Bezout identity	ax + by = gcd(a, b).

Security implications. Euclid's algorithm is efficient even for numbers with hundreds of digits, which is why RSA key generation is practical.

Multiplicative inverse

The inverse of a modulo n is the number x with a x ≡ 1 (mod n). It exists exactly when gcd(a, n) = 1, and the Extended Euclidean algorithm provides it: if ax + ny = 1 then x mod n is the inverse.

Verified example: inverse of 7 modulo 26. Running the algorithm on (26, 7):

Table 26. Extended Euclidean table for gcd(26, 7)

q	r0	r1	r	s	t
3	26	7	5	1	0
1	7	5	2	0	1
2	5	2	1	1	-3
2	2	1	0	-1	4

The final non-zero remainder is 1; the Bezout coefficients give 7x(-11) + 26x3 = 1, so 7^{-1} = -11 mod 26 = **15**. Check: 7x15 = 105 = 4x26 + 1. Similarly, 17^{-1} mod 3120 = **2753** (used later in RSA).

No inverse
4 has no inverse modulo 26 because gcd(4, 26) = 2. This is why only some Hill key matrices and RSA exponents are valid.

8.3 Risk Management

In simple words. Risk management is a structured way to decide which security problems matter most and what to do about them.

Formal definition. Risk management is the coordinated process of identifying, analysing, evaluating and treating risks to an organisation's assets, followed by monitoring and review (NIST SP 800-30).

Everyday analogy. Checking the weather and traffic before a journey, then deciding to leave early, take another road, buy insurance, or accept a delay.

How it works. Identify assets and threats, analyse likelihood and impact, rank risks, choose a treatment, apply controls, and review continuously.

Figure 18 | Risk assessment lifecycle

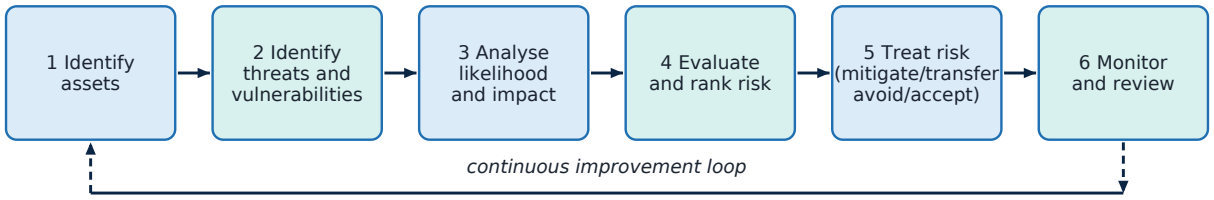


Figure 18. *Risk assessment lifecycle*

Term	Meaning
Risk exposure (RE)	RE = probability of loss x size of loss.
Risk reduction leverage (RRL)	RRL = (RE before - RE after) / cost of the control.
Mitigate	Reduce likelihood or impact with controls.
Transfer	Share the risk, for example insurance.
Avoid	Stop the risky activity.
Accept	Knowingly keep a low risk.
Residual risk	Risk remaining after controls.

Practical applications. Banks, hospitals and universities maintain risk registers reviewed by management.

Common misconception
Risk can be reduced to zero.

Security implications. Residual risk always remains; the aim is to reduce it to an acceptable level.

Worked example. A university database loss would cost PKR 5,000,000 with probability 0.20 per year, so RE = 0.20 x 5,000,000 = **PKR 1,000,000**. A replicated backup system costing PKR 200,000 reduces probability to 0.05, so RE after = 0.05 x 5,000,000 = PKR 250,000. RRL = (1,000,000 - 250,000) / 200,000 = **3.75**. An RRL above 1 means the control saves more than it costs.

Table 27. Qualitative risk matrix (5 x 5 scoring)

Level	Likelihood	Impact	Score (L x I)
Low	1 Rare	1 Minor	1-4
Medium	2-3	2-3	5-9
High	4-5	4-5	10-25

Case Study: Risk register for a university online examination system

Background	The university moves final exams online.
Organisation and environment	4 campuses, 6,000 students, cloud-hosted exam platform.
Information assets	Question bank, student answers, exam server.
Security problem	Unclear which risks deserve funding first.
Threats and vulnerabilities	Threats: DoS, cheating, insider leaks. Vulnerabilities: single server, weak passwords.
Potential impact	Exam cancellation, reputational damage.
Detection	Quarterly risk workshop with IT, exams office and academics.
Investigation	Assets listed; likelihood and impact scored 1-5.
Response	Top risk: single point of failure (score 20). Add failover and rate limiting.
Prevention	Review register each term; test failover before exams.
Lessons learned	Risk-based spending targets the biggest exposure first. Fictional scenario.

Practical 12: Euclidean Algorithm

Learning objective	Compute GCDs and Bezout coefficients.
Software and tools	Python 3.
Environment	Terminal.
Background	GCD is the first step of inverse finding.

Step-by-step procedure

1. Save euclid.py.
2. Run it.
3. Test gcd(252,105) by hand and by program.

Program: euclid.py

```
# Euclidean, Extended Euclidean algorithm and modular inverse
def gcd(a, b):
    while b:
        print(f"  {a} = {a // b} x {b} + {a % b}")
        a, b = b, a % b
    return a
def egcd(a, b):
    if b == 0: return a, 1, 0
    g, x, y = egcd(b, a % b)
    return g, y, x - (a // b) * y
def modinv(a, m):
    g, x, _ = egcd(a, m)
    if g != 1: raise ValueError("no inverse: gcd(a, m) != 1")
    return x % m
print("gcd(1071, 462) ="); print(" ->", gcd(1071, 462))
g, x, y = egcd(240, 46); print("egcd(240,46):", g, x, y, "check:", 240 * x + 46 * y)
print("7^-1 mod 26   =", modinv(7, 26), " check:", 7 * modinv(7, 26) % 26)
print("17^-1 mod 3120 =", modinv(17, 3120))
try: modinv(4, 26)
except ValueError as e: print("4 mod 26:", e)
```

Output

```
gcd(1071, 462) =
  1071 = 2 x 462 + 147
  462 = 3 x 147 + 21
  147 = 7 x 21 + 0
 -> 21
egcd(240,46): 2 -9 47 check: 2
7^-1 mod 26   = 15  check: 1
17^-1 mod 3120 = 2753
4 mod 26: no inverse: gcd(a, m) != 1
```

Output interpretation	gcd(1071,462)=21; egcd(240,46) gives 2 = 240x(-9)+46x47.
Verification procedure	Substitute x and y into ax+by.
Common errors	Recursion depth with huge numbers.
Troubleshooting	Use the iterative form.
Security precautions	None.
Practical conclusion	Euclid is the engine of modular inverses.
Student questions	1) What is gcd(252,105)? 2) What does egcd return? 3) Why does 4 have no inverse mod 26?

Practical 13: Modular Inverse

Learning objective	Compute inverses using both algorithms.
Software and tools	Python 3.
Environment	Terminal.
Background	Needed for Hill decryption and RSA.

Step-by-step procedure

1. Use the same program.
2. Verify 7x15 mod 26 = 1.
3. Compute 17 inverse mod 3120.

Output

```
egcd(240,46): 2 -9 47 check: 2
7^-1 mod 26   = 15  check: 1
17^-1 mod 3120 = 2753
4 mod 26: no inverse: gcd(a, m) != 1
```

Output interpretation	7 inverse = 15; 17 inverse = 2753; 4 has none.
Verification procedure	Multiply a by inverse modulo n.
Common errors	Negative results.
Troubleshooting	Add n until positive.
Security precautions	None.
Practical conclusion	Inverse needs gcd 1.
Student questions	1) Find 11 inverse mod 26. 2) Why can inverse be negative before reduction? 3) Where is it used in RSA?

Practical 34: Risk Register for a Fictional Organisation

Learning objective	Build and rank a risk register.
Software and tools	Spreadsheet.
Environment	Any.
Background	Risk registers support decisions.

Step-by-step procedure

1. List 8 assets and threats.
2. Score L and I from 1 to 5.
3. Compute score and rank.
4. Choose treatment and residual risk.

Table 28. Sample risk register

ID	Risk	L	I	Score	Treatment
R1	Single exam server fails	4	5	20	Mitigate: failover
R2	Phishing of staff	5	3	15	Mitigate: MFA, training
R3	Flood damages server room	2	5	10	Transfer: insurance, off-site backup
R4	Old lab PCs	3	2	6	Accept / plan replacement

Output interpretation	Highest scores deserve first attention.
Verification procedure	Review with a classmate.
Common errors	Scoring everything high.
Troubleshooting	Use relative ranking.
Security precautions	Fictional data.
Practical conclusion	Risk registers support defensible decisions.
Student questions	1) Which risk is highest? 2) What is residual risk? 3) When should the register be reviewed?

Module 8 Revision

1. Module summary

Modular arithmetic wraps numbers; GCD and the Extended Euclidean algorithm yield inverses. Risk management identifies, analyses and treats risks; RE and RRL support spending decisions.

2. Key definitions

Term	Definition
Congruence	a = b (mod n) if n | (a - b).
Inverse	x with ax = 1 mod n.
Risk exposure	Probability x loss.
RRL	(RE before - RE after)/cost.

3. Important terminology

Residue, coprime, Bezout, mitigate, transfer, avoid, accept.

4. Comparison table

Table 29. Comparison for Module 8

Treatment	Meaning	Example
Mitigate	Reduce	Backups
Transfer	Share	Insurance
Avoid	Stop	Disable feature
Accept	Keep	Low-risk glitch

5. Quick revision notes

- Inverse exists iff gcd = 1.
- RRL > 1 means worthwhile.

6. Memory techniques

- Treatment: MATA = Mitigate, Avoid, Transfer, Accept.

7. Important formulas

- gcd(a,b)=gcd(b,a mod b)
- ax+by=gcd
- RE = P x L
- RRL = (REb - REa)/C

8. Common examination mistakes

- Forgetting to reduce mod n.
- Mixing RE and RRL.

Module 8 Question Bank

9. Short questions (15)

S1. Define congruence.
S2. Compute 40 mod 26.
S3. Compute (-7) mod 26.
S4. Define gcd.
S5. State Euclid's recurrence.
S6. What does the extended algorithm add?
S7. Define multiplicative inverse.
S8. When does an inverse exist?
S9. What is 7 inverse mod 26?
S10. Define risk exposure.
S11. Define RRL.
S12. List four risk treatments.
S13. What is residual risk?
S14. What is a risk register?
S15. Why are risk matrices used?

10. Long questions (8)

L1. Explain modular arithmetic properties with examples.
L2. Explain Euclid's algorithm with an example.
L3. Explain extended Euclid and find an inverse.
L4. Explain square-and-multiply.

L5. Explain the risk management process.
L6. Explain RE and RRL with a worked example.
L7. Discuss risk treatment options.
L8. Describe how to build a risk register.

11. Multiple-choice questions (10)

Q1. gcd(1071,462)=
A) 42
B) 3
C) 147
D) 21

Q2. 7^-1 mod 26=
A) 15
B) 5
C) 11
D) 19

Q3. Inverse exists iff:
A) n prime only
B) a even
C) gcd(a,n)=1
D) a>n

Q4. RE =
A) L-P
B) P+L
C) P x L
D) P/L

Q5. Insurance is:
A) Accept
B) Avoid
C) Transfer
D) Ignore

Q6. (15x9) mod 26=
A) 9
B) 5
C) 11
D) 3

Q7. 5^13 mod 23=
A) 21
B) 8
C) 2
D) 22

Q8. RRL above 1 means:
A) Unknown
B) Illegal
C) Waste
D) Worthwhile

Q9. Residual risk is:

A) Threat

B) Remaining risk

C) Zero risk

D) Added risk

Q10. (-7) mod 26=

A) -19

B) 19

C) 26

D) 7

12. Scenario-based questions (5)

C1. Loss PKR 2,000,000, P=0.1, control reduces to P=0.02 at cost 100,000. Compute RRL.

C2. A system has two equal risks; budget covers one. How to decide?

C3. Explain why gcd(6,26)=2 blocks an RSA exponent choice analogue.

C4. A manager wants zero risk. Respond.

C5. Which treatment fits moving to a safer vendor?

13. Practical and problem-solving exercises (3)

X1. Find gcd(252,105) and Bezout coefficients.

X2. Find 11 inverse mod 26.

X3. Build a 5-row risk register.

Model answers (selected short questions)

S5. gcd(a,b)=gcd(b,a mod b), repeated until remainder is 0.

S9. 7 inverse mod 26 is 15.

Detailed model answers (selected long questions)

L6. RE=P x L. For P=0.2, L=5,000,000, RE=1,000,000. If a control reduces P to 0.05 at cost 200,000, RE after = 250,000 and RRL=(1,000,000-250,000)/200,000=3.75, so the control is worthwhile.

14. References

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

Paar, C., & Pelzl, J. (2010). *Understanding cryptography: A textbook for students and practitioners*. Springer.

National Institute of Standards and Technology. (2012). *Guide for conducting risk assessments* (NIST SP 800-30 Rev. 1). https://doi.org/10.6028/NIST.SP.800-30r1

Stallings, W., & Brown, L. (2018). *Computer security: Principles and practice* (4th ed.). Pearson.

Module 09
Advanced Encryption Standard

Introduction. AES is the symmetric cipher that protects most of the world's data today: Wi-Fi (WPA2/WPA3), HTTPS, disk encryption, messaging apps and cloud storage. We study its structure, then learn the practical rules (keys, IVs, nonces, modes) that decide whether a real system is safe.

Why this topic matters. A strong cipher used wrongly is still insecure. This module teaches both the cipher and the safe way to use it.

Real-world relevance. Online banking (TLS), mobile messaging, full-disk encryption and cloud storage encryption.

Learning objectives	Expected learning outcomes
• Describe AES block and key sizes and number of rounds. • Explain SubBytes, ShiftRows, MixColumns and AddRoundKey. • Explain key expansion and decryption. • Distinguish modes of operation and explain nonces and IVs. • Use AES-GCM safely with a trusted library.	• Trace one AES round on real data. • Encrypt and decrypt with AES-256-GCM and detect tampering.

Prerequisites	Estimated learning time
Modules 7 and 8 (XOR, Feistel, finite arithmetic).	About 8 hours.

9.1 Introduction and Structure of AES

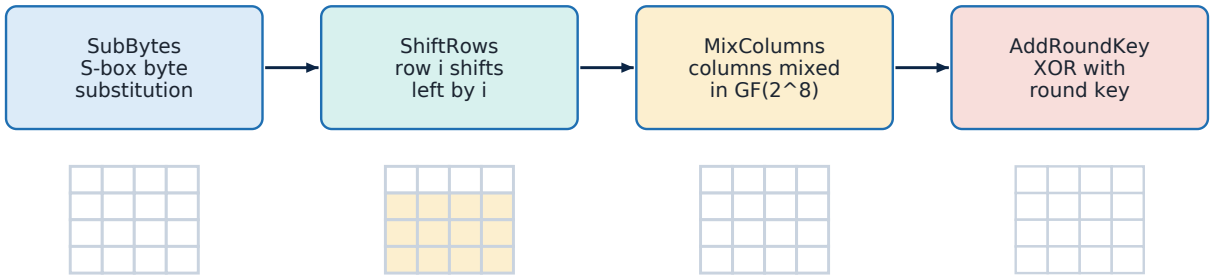
In simple words. AES scrambles 16 bytes at a time through 10, 12 or 14 rounds of four simple operations, using a secret key of 128, 192 or 256 bits.

Formal definition. AES is the Rijndael cipher selected by NIST in 2001 (FIPS 197): a symmetric block cipher with a 128-bit block and key sizes of 128, 192 or 256 bits, built as a substitution-permutation network of 10, 12 or 14 rounds.

Everyday analogy. A recipe repeated many times: replace each ingredient (SubBytes), rotate the rows of the tray (ShiftRows), blend each column (MixColumns), and sprinkle a secret spice (AddRoundKey).

How it works. The 16-byte block is arranged as a 4x4 **state** matrix (column by column). The state is XORed with the first round key, then each round applies SubBytes, ShiftRows, MixColumns and AddRoundKey. The final round omits MixColumns.

Figure 19 | AES round transformations



AES-128: initial AddRoundKey, 9 full rounds, then a final round without MixColumns (10 rounds)

Figure 19. *AES round transformations*

Term	Meaning
State	4x4 byte matrix being transformed.
Round key	128-bit key derived from the cipher key.
S-box	Non-linear byte substitution table.
GF(2^8)	Finite field in which MixColumns multiplies.

Practical applications. Wi-Fi WPA2/3, TLS, BitLocker/FileVault, 7-Zip, cloud storage.

Common misconception
AES-256 is always needed.

Security implications. AES-128 remains secure; correct mode and key handling matter more than key size.

Table 30. AES variants

Key size	Rounds	Block	Typical use
128 bits	10	128 bits	General purpose, TLS, Wi-Fi
192 bits	12	128 bits	Rare
256 bits	14	128 bits	High assurance, long-term protection

The four round operations

- **SubBytes**: each byte is replaced using the S-box (for example, 0x53 becomes 0xED). This is the only non-linear step and provides confusion.
- **ShiftRows**: row 0 unchanged, row 1 shifted left by 1, row 2 by 2, row 3 by 3. Provides diffusion across columns.
- **MixColumns**: each column is multiplied by a fixed matrix in GF(2^8) so every output byte depends on all four input bytes. Not applied in the last round.
- **AddRoundKey**: state XOR round key; the only step that uses the key.

Key expansion

The cipher key is expanded into 11 round keys (AES-128). Words are generated by XOR, with a rotation (RotWord), S-box substitution (SubWord) and a round constant (Rcon) applied to every fourth word, so each round key differs and the structure is not symmetric.

Decryption

AES decryption applies the inverse operations (InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns) with round keys in reverse order. Unlike DES, AES is not a Feistel cipher, so each operation needs an explicit inverse.

Verified example (FIPS 197 Appendix)

Key 000102...0f, plaintext 00112233445566778899aabbccddeeff. After the initial AddRoundKey the state is 00102030405060708090a0b0c0d0e0f0. In round 1:

Table 31. AES-128 round 1 (verified)

Step	State (hex)
After SubBytes	63cab7040953d051cd60e0e7ba70e18c
After ShiftRows	6353e08c0960e104cd70b751bacad0e7
After MixColumns	5f72641557f5bc92f7be3b291db9f91a
Round key 1	d6aa74fdd2af72fadaa678f1d6ab76fe
After AddRoundKey (start of round 2)	89d810e8855ace682d1843d8cb128fe4

After ten rounds the ciphertext is **69c4e0d86a7b0430d8cdb78070b4c55a**, exactly the published FIPS 197 result. This was also confirmed against the *cryptography* library.

9.2 Keys, IVs, Nonces and Modes of Operation

In simple words. A block cipher handles 16 bytes. Modes explain how to encrypt longer data safely; an IV or nonce makes each encryption unique.

Formal definition. A mode of operation defines how a block cipher processes messages longer than one block. An initialisation vector (IV) or nonce is a per-message value that must be unique (and sometimes unpredictable) for each encryption under the same key.

Everyday analogy. The same photocopier with a new random stamp each time, so two identical letters never look identical.

How it works. ECB encrypts blocks independently, so equal plaintext blocks give equal ciphertext blocks and patterns leak. CBC chains blocks with a random IV but gives no integrity. CTR turns AES into a stream cipher with a nonce and counter. GCM adds authentication (an integrity tag).

Term	Meaning
IV	Initial value for CBC, must be unpredictable.
Nonce	Number used once; must never repeat with the same key.
AEAD	Authenticated Encryption with Associated Data (AES-GCM, ChaCha20-Poly1305).
Tag	Authentication value that detects tampering.

Practical applications. TLS 1.3 uses AES-GCM or ChaCha20-Poly1305.

Common misconception
Encryption alone guarantees integrity.

Security implications. Without authentication, ciphertext can be altered; always use AEAD.

Table 32. Modes of operation compared

Mode	Needs	Hides patterns?	Integrity?	Recommendation
ECB	Nothing	No	No	Never use
CBC	Random IV, padding	Yes	No	Legacy only; add MAC
CTR	Unique nonce	Yes	No	Use with MAC
GCM	Unique 96-bit nonce	Yes	Yes	Preferred

Critical rule

> Never reuse a nonce with the same key in AES-GCM. A repeated nonce can reveal plaintext relationships and allow forgery of tags.

Safe key handling

- Generate keys with a cryptographically secure random generator (never from a typed password directly; use a key-derivation function).
- Never hard-code keys in source code or commit them to repositories.
- Store keys in a key vault or hardware module; separate keys from data.
- Rotate keys and keep a plan for revocation and recovery.
- Use a well-reviewed library, never your own AES implementation.

Case Study: Unencrypted student records in cloud storage

Background	A department stores exported grade sheets in a cloud folder.
Organisation and environment	University faculty, cloud storage, shared links.
Information assets	Student grade files.
Security problem	Files are stored unencrypted and links shared widely.
Threats and vulnerabilities	Threat: link guessing, insider. Vulnerability: no encryption or access control.
Potential impact	Privacy breach for 800 students.
Detection	Cloud provider alert shows unusual downloads.
Investigation	Access logs show downloads from unfamiliar addresses.
Response	Revoke links, restrict access, encrypt files with AES-256-GCM and store keys in a vault.
Prevention	Encrypt sensitive data at rest and in transit, use least privilege, and audit access.
Lessons learned	Encryption protects data when access control fails. Fictional scenario.

Practical 16: AES Encryption and Decryption (AES-256-GCM)

Learning objective	Encrypt, decrypt and detect tampering with AES-GCM.
Software and tools	Python 3, library *cryptography*.
Environment	pip install cryptography (use a virtual environment).
Background	AES-GCM provides both secrecy and integrity.

Step-by-step procedure

1. Install: python3 -m pip install cryptography
2. Save aes_gcm_lab.py.
3. Run it.
4. Change one byte of ciphertext and observe the error.

Program: aes_gcm_lab.py

```
# AES-256-GCM authenticated encryption with the 'cryptography' library
import os
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
key = AESGCM.generate_key(bit_length=256)      # random 256-bit key
aes = AESGCM(key)
nonce = os.urandom(12)                         # 96-bit nonce, NEVER reuse with the same key
header = b"student-record-v1"                  # authenticated but not encrypted
ct = aes.encrypt(nonce, b"GPA=3.85; ID=2024-CS-017", header)
print("Ciphertext+tag length:", len(ct), "bytes")
print("Decrypted:", aes.decrypt(nonce, ct, header))
tampered = bytearray(ct); tampered[0] ^= 1
try:
    aes.decrypt(nonce, bytes(tampered), header)
except Exception as e:
    print("Tampering detected:", type(e).__name__)
```

Output

```
Ciphertext+tag length: 40 bytes
Decrypted: b'GPA=3.85; ID=2024-CS-017'
Tampering detected: InvalidTag
```

Output interpretation	Ciphertext is plaintext length plus 16-byte tag (24+16=40). A modified ciphertext raises InvalidTag.
Verification procedure	Decrypt and compare; then tamper and confirm rejection.
Common errors	ModuleNotFoundError.
Troubleshooting	Install the library in the same Python.
Security precautions	Do not store the key beside the data; generate a new nonce for every message.
Practical conclusion	Authenticated encryption is the default choice.
Student questions	1) Why is the nonce 12 bytes? 2) What happens if the nonce repeats? 3) Why include associated data?

Module 9 Revision

1. Module summary

AES is a 128-bit block cipher with 10, 12 or 14 rounds (SubBytes, ShiftRows, MixColumns, AddRoundKey). Safe use requires AEAD modes, unique nonces and proper key management.

2. Key definitions

Term	Definition
AES	NIST standard block cipher.
AEAD	Encryption with integrity.
Nonce	Value used once.
ECB	Insecure mode.

3. Important terminology

State, S-box, Rcon, GF(2^8), tag.

4. Comparison table

Table 33. Comparison for Module 9

	DES	AES
Block	64	128
Key	56	128/192/256
Structure	Feistel	SPN

5. Quick revision notes

- AES-GCM = encryption plus authentication.
- No nonce reuse.

6. Memory techniques

- AES round: "Sub, Shift, Mix, Add".

7. Important formulas

- Rounds: 10/12/14 for 128/192/256-bit keys

8. Common examination mistakes

- Using ECB.
- Reusing nonces.
- Believing larger key fixes bad mode.

Module 9 Question Bank

9. Short questions (15)

S1. What is the AES block size?
S2. Name the AES key sizes.
S3. How many rounds for AES-128?
S4. What does SubBytes do?
S5. What does ShiftRows do?
S6. What does MixColumns do?
S7. What does AddRoundKey do?
S8. Which round omits MixColumns?
S9. What is key expansion?
S10. What is an IV?
S11. What is a nonce?
S12. Why is ECB insecure?
S13. What is AEAD?
S14. What is GCM?
S15. Name one safe key-handling rule.

10. Long questions (8)

L1. Explain AES structure with diagram.
L2. Explain each AES round step.
L3. Explain key expansion.
L4. Explain AES decryption.
L5. Compare ECB, CBC, CTR and GCM.
L6. Explain nonce reuse risk.
L7. Explain safe key management.

L8. Compare AES with DES.

11. Multiple-choice questions (10)

Q1. AES block size:

A) 192
B) 256
C) 64
D) 128

Q2. AES-256 rounds:

A) 12
B) 16
C) 10
D) 14

Q3. Non-linear AES step:

A) MixColumns
B) ShiftRows
C) SubBytes
D) AddRoundKey

Q4. Never-use mode:

A) GCM
B) ECB
C) CTR
D) CBC with MAC

Q5. GCM provides:

A) Secrecy and integrity
B) Compression
C) Hashing only
D) Only secrecy

Q6. Nonce rule in GCM:

A) Never reuse with same key
B) Reuse often
C) Use zero
D) Publish key

Q7. AES is a:

A) Hash
B) Feistel cipher
C) Substitution-permutation network
D) Stream cipher

Q8. Key storage best practice:

A) Email
B) Key vault
C) Hard-code
D) Comment in code

Q9. MixColumns operates over:

A) Integers

B) GF(2^8)

C) Matrices of letters

D) Primes

Q10. AES standard:

A) RFC 7465

B) FIPS 46-3

C) SP 800-61

D) FIPS 197

12. Scenario-based questions (5)

C1. An app encrypts every block with ECB. What do you tell the developer?

C2. A developer reuses a GCM nonce. Consequence?

C3. Where should the key for a web app database encryption be stored?

C4. AES-128 or AES-256 for student records?

C5. You receive ciphertext with tag failing. What do you do?

13. Practical and problem-solving exercises (3)

X1. Run the lab and measure ciphertext length for 100-byte plaintext.

X2. Draw ShiftRows on a 4x4 state.

X3. Write the AES-GCM function that returns nonce+ciphertext.

Model answers (selected short questions)

S12. ECB encrypts equal blocks identically, leaking patterns.

S13. Authenticated Encryption with Associated Data: provides confidentiality and integrity.

Detailed model answers (selected long questions)

L5. ECB: independent blocks, patterns leak, no integrity. CBC: chaining with random IV, no integrity. CTR: stream mode with unique nonce, no integrity. GCM: CTR plus authentication tag, preferred.

14. References

National Institute of Standards and Technology. (2001). *Advanced Encryption Standard (AES)* (FIPS PUB 197). https://doi.org/10.6028/NIST.FIPS.197

Dworkin, M. (2007). *Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC* (NIST SP 800-38D). https://doi.org/10.6028/NIST.SP.800-38D

Python Cryptographic Authority. (n.d.). *cryptography documentation*. https://cryptography.io/

Paar, C., & Pelzl, J. (2010). *Understanding cryptography: A textbook for students and practitioners*. Springer.

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

Module 10 Public-Key Cryptography and RSA

Introduction. Symmetric ciphers need a shared secret. Public-key cryptography removes that need: anyone can encrypt with a public key, but only the private-key holder can decrypt. RSA is the best-known public-key system and the foundation for many digital signatures and secure key exchanges.

Why this topic matters. Without public-key cryptography, strangers could not establish secure connections on the Internet.

Real-world relevance. TLS handshakes, software signing, secure email, digital certificates.

Learning objectives	Expected learning outcomes
• Explain public and private keys. • Describe the mathematics behind RSA (primes, modulus, totient). • Generate an RSA key pair and encrypt and decrypt small numbers. • Explain padding and why textbook RSA is insecure. • State modern key-size guidance.	• Perform an RSA calculation with full working. • Use a recognised library with OAEP and PSS.

Prerequisites	Estimated learning time
Modules 8 and 9.	About 8 hours.

10.1 Public-Key Cryptography

In simple words. Each person has two linked keys: a public one to share and a private one to guard.

Formal definition. Public-key (asymmetric) cryptography uses a key pair (PK, SK) such that data encrypted with PK can be decrypted only with SK, and signatures made with SK can be verified with PK. Security rests on mathematical problems believed hard, such as integer factorisation.

Everyday analogy. A public mailbox: anybody posts a letter through the slot, only the key holder opens the box.

How it works. The sender encrypts with the recipient's public key; the recipient decrypts with the private key. For signatures the roles reverse: sign with private, verify with public.

Term	Meaning
Public key	Shared openly.
Private key	Kept secret.
One-way trapdoor function	Easy to compute, hard to reverse without secret information.
Key exchange	Agreeing on a shared secret over an open channel.

Practical applications. Establishing TLS sessions, software updates, e-mail encryption.

Common misconception

The public key can decrypt what the private key encrypted for secrecy.

Security implications. Encryption uses the public key; signing uses the private key.

10.2 RSA Algorithm

In simple words. RSA multiplies two big primes to make a public number; breaking it means factoring that number.

Formal definition. RSA (Rivest, Shamir, Adleman, 1977) uses modulus n = pq (p, q prime), public exponent e coprime to φ(n) = (p-1)(q-1), and private exponent d = e^{-1} mod φ(n). Encryption: C = M^e mod n. Decryption: M = C^d mod n.

Everyday analogy. Multiplying two large primes is like mixing two paint colours; recovering the original colours is extremely hard.

How it works. Key generation: choose primes p, q; compute n and φ(n); choose e (commonly 65537); compute d with the extended Euclidean algorithm. Public key (e, n); private key (d, n).

Figure 20 | RSA key generation and use

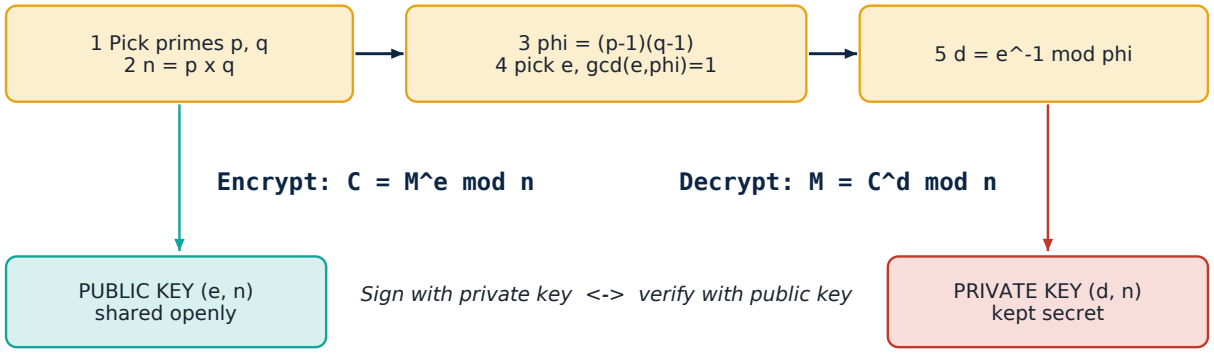


Figure 20. *RSA key generation and use*

Term	Meaning
n (modulus)	Product p x q.
φ(n)	Euler totient: count of integers coprime to n; (p-1)(q-1).
e	Public exponent.
d	Private exponent, inverse of e modulo φ(n).

Practical applications. Digital certificates, secure shell, code signing.

Common misconception
RSA with small numbers like in textbooks is secure.

Security implications. Classroom primes are tiny; real RSA uses 2048-bit or larger moduli and padding.

Worked example 1: p = 61, q = 53

n = 3233; φ = 60 x 52 = 3120; e = 17; d = 17^{-1} mod 3120 = **2753** (verified: 17 x 2753 = 46801 = 15 x 3120 + 1). Encrypt M = 65: C = 65^{17} mod 3233 = **2790**. Decrypt: 2790^{2753} mod 3233 = **65**.

Worked example 2: p = 3, q = 11

n = 33; φ = 20; e = 7; d = 3 (since 7 x 3 = 21 = 1 mod 20). Encrypt M = 4: C = 4^7 mod 33 = **16**. Decrypt: 16^3 mod 33 = **4**.

Security of RSA

- Security relies on the difficulty of factoring n; therefore n must be large. NIST treats 2048-bit RSA as the minimum acceptable today (about 112-bit security) and 3072-bit for about 128-bit security.
- Textbook RSA is deterministic and malleable; always use padding: **OAEP** for encryption and **PSS** for signatures (RFC 8017).
- Do not encrypt bulk data with RSA; encrypt a symmetric key instead (hybrid encryption).
- Protect the private key; implementation flaws and side channels cause most real failures.
- Large quantum computers (Shor's algorithm) would break RSA; post-quantum algorithms are being standardised.

Case Study: Weak keys in an old intranet server

Background	An old intranet server uses a 512-bit RSA key.
Organisation and environment	University intranet with legacy hardware.
Information assets	Staff login data, server private key.
Security problem	Key length is too short.
Threats and vulnerabilities	Threat: attacker with factoring resources. Vulnerability: obsolete key size.
Potential impact	Possible impersonation of the server.
Detection	Security scan flags 512-bit key.
Investigation	Certificate review shows it expires in 2 years.
Response	Generate a 3072-bit key, install new certificate, revoke the old.
Prevention	Keep a certificate and key inventory; set renewal alerts.
Lessons learned	Key length must follow current guidance. Fictional scenario.

Practical 17: RSA Key Generation, Encryption and Decryption

Learning objective	Perform toy RSA and real RSA with OAEP and PSS.
Software and tools	Python 3, *cryptography* library.
Environment	Terminal, virtual environment.
Background	Toy RSA teaches mathematics; library RSA teaches safe practice.

Step-by-step procedure

1. Run rsa_toy.py and verify with your hand calculation.
2. Run rsa_lib.py.
3. Note ciphertext size equals key size in bytes.

Program: rsa_toy.py (educational only)

```
# Textbook RSA with tiny primes - for understanding ONLY, completely insecure
p, q = 61, 53
n = p * q; phi = (p - 1) * (q - 1); e = 17
d = pow(e, -1, phi)                 # modular inverse (Python 3.8+)
print("n =", n, " phi =", phi, " e =", e, " d =", d)
m = 65
c = pow(m, e, n); print("Encrypt:", m, "->", c)
print("Decrypt:", c, "->", pow(c, d, n))
```

Output

```
n = 3233  phi = 3120  e = 17  d = 2753
Encrypt: 65 -> 2790
Decrypt: 2790 -> 65
```

Program: rsa_lib.py (recognised library)

```
# Real RSA with a recognised library: OAEP encryption and PSS signature
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes
priv = rsa.generate_private_key(public_exponent=65537, key_size=3072)
pub = priv.public_key()
oaep = padding.OAEP(mgf=padding.MGF1(hashes.SHA256()), algorithm=hashes.SHA256(), label=None)
ct = pub.encrypt(b"session key material", oaep)
print("Ciphertext bytes:", len(ct), " Decrypted:", priv.decrypt(ct, oaep))
pss = padding.PSS(mgf=padding.MGF1(hashes.SHA256()), salt_length=padding.PSS.MAX_LENGTH)
sig = priv.sign(b"exam results v1", pss, hashes.SHA256())
pub.verify(sig, b"exam results v1", pss, hashes.SHA256()); print("Signature valid")
try: pub.verify(sig, b"exam results v2", pss, hashes.SHA256())
except Exception as e: print("Altered message rejected:", type(e).__name__)
```

Output

```
Ciphertext bytes: 384  Decrypted: b'session key material'
Signature valid
Altered message rejected: InvalidSignature
```

Output interpretation	Toy RSA reproduces M = 65 -> 2790 -> 65. Library RSA gives a 384-byte ciphertext for a 3072-bit key and rejects altered messages.
Verification procedure	Match toy results to hand calculations; confirm signature fails after message change.
Common errors	Slow key generation.
Troubleshooting	Wait; 3072-bit generation takes a moment.
Security precautions	Never use toy primes; never keep private keys in source code.
Practical conclusion	Real systems use RSA only with proper padding.
Student questions	1) Why is textbook RSA insecure? 2) What is OAEP for? 3) What size key do you choose?

Module 10 Revision

1. Module summary

RSA: n = pq, e coprime to (p-1)(q-1), d = e inverse mod phi. Encrypt M^e, decrypt C^d mod n. Use 2048+ bit keys with OAEP/PSS padding and hybrid encryption.

2. Key definitions

Term	Definition
Public key	Shareable key.
Private key	Secret key.
Totient	(p-1)(q-1).
OAEP	Encryption padding.

3. Important terminology

Modulus, exponent, factoring, trapdoor.

4. Comparison table

Table 34. Comparison for Module 10

	Symmetric	RSA
Keys	1	2
Speed	Fast	Slow
Use	Data	Keys, signatures

5. Quick revision notes

- d = e^-1 mod phi.
- Always pad.

6. Memory techniques

- RSA: "Public Encrypts, Private Decrypts; Private Signs, Public Verifies".

7. Important formulas

- n=pq
- phi=(p-1)(q-1)
- C=M^e mod n
- M=C^d mod n

8. Common examination mistakes

- Mixing up which key signs.
- Using tiny keys.

Module 10 Question Bank

9. Short questions (15)

S1. What is public-key cryptography?
S2. Which key is secret?
S3. Define the RSA modulus.
S4. Define totient.
S5. What is e commonly?
S6. How is d computed?
S7. Write RSA encryption.
S8. Write RSA decryption.
S9. Why is padding needed?
S10. What is OAEP?
S11. What is PSS?
S12. What is hybrid encryption?
S13. Which problem underlies RSA security?
S14. What key size do you recommend?
S15. What threatens RSA in the future?

10. Long questions (8)

L1. Explain public-key cryptography.
L2. Explain RSA key generation with example.
L3. Explain RSA encryption and decryption.
L4. Explain RSA security and padding.
L5. Compare symmetric and RSA.

L6. Explain how hybrid encryption combines RSA with a symmetric cipher.

L7. Solve an RSA problem with p=3, q=11.

L8. Discuss key-size guidance.

11. Multiple-choice questions (10)

Q1. phi for p=61, q=53:

A) 3233

B) 3000

C) 3120

D) 3333

Q2. Which key decrypts RSA messages?

A) Private

B) Hash

C) Either

D) Public

Q3. RSA security rests on:

A) Factoring

B) Short keys

C) Discrete logs only

D) Hashing

Q4. For n=33, e=7, d=

A) 2

B) 7

C) 5

D) 3

Q5. Recommended minimum RSA:

A) 512

B) 2048

C) 1024

D) 256

Q6. RSA encryption padding:

A) ECB

B) PSS

C) OAEP

D) CBC

Q7. Signatures use:

A) Same key

B) No keys

C) Public to sign

D) Private to sign

Q8. RSA for bulk data:

A) Required

B) Not recommended

C) Best

D) Illegal

Q9. Textbook RSA is:

A) Deterministic

B) Authenticated

C) Quantum safe

D) Randomised

Q10. (M^e)^d mod n gives:

A) d

B) e

C) n

D) M

12. Scenario-based questions (5)

C1. Two firms wish to exchange files using RSA only. Advise.

C2. A key of 1024 bits is found. Action?

C3. Explain what happens if p and q are found.

C4. Why must d remain secret?

C5. Is RSA safe against quantum computers?

13. Practical and problem-solving exercises (3)

X1. Compute RSA for p=5,q=11,e=7, M=9.

X2. Verify 17 x 2753 mod 3120 =1.

X3. Generate a key pair with openssl and inspect.

Model answers (selected short questions)

S9. Padding randomises encryption and prevents attacks on deterministic textbook RSA.

S13. The difficulty of factoring the modulus n.

Detailed model answers (selected long questions)

L7. p=3, q=11: n=33, phi=20, choose e=7, d=3 (7x3=21=1 mod 20). Encrypt M=4: 4^7 mod 33 = 16. Decrypt: 16^3 mod 33 = 4.

14. References

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

Moriarty, K., Kaliski, B., Jonsson, J., & Rusch, A. (2016). *PKCS #1: RSA cryptography specifications version 2.2* (RFC 8017). https://www.rfc-editor.org/info/rfc8017

Paar, C., & Pelzl, J. (2010). *Understanding cryptography: A textbook for students and practitioners*. Springer.

Python Cryptographic Authority. (n.d.). *cryptography documentation*. https://cryptography.io/

Joint Task Force. (2020). *Security and privacy controls for information systems and organizations* (NIST SP 800-53 Rev. 5).

Module 11
Hash Functions, Digital Signatures and PKI

Introduction. Encryption hides data; hashes and signatures prove that data has not changed and who produced it. Public Key Infrastructure (PKI) extends this trust to the whole Internet by linking public keys to identities through certificates.

Why this topic matters. Every software download, HTTPS padlock and signed document depends on hashes, signatures and certificates.

Real-world relevance. Software updates, e-government portals, online banking and secure e-mail.

Learning objectives	Expected learning outcomes
• Explain hash function properties. • Distinguish hashing from encryption. • Describe the SHA family. • Explain digital signatures (signing and verification). • Explain PKI, certificates, CAs, chains, trust models and revocation.	• Compute SHA-256 and verify file integrity. • Sign and verify a message. • Inspect a certificate.

Prerequisites	Estimated learning time
Modules 9 and 10.	About 8 hours.

11.1 Cryptographic Hash Functions

In simple words. A hash function turns any input into a short, fixed-length "fingerprint" that is practically impossible to reverse or fake.

Formal definition. A cryptographic hash function H maps arbitrary-length input to a fixed-length digest and satisfies preimage resistance, second-preimage resistance and collision resistance.

Everyday analogy. A fingerprint: easy to take from a person, impossible to rebuild the person from it, and different people have different fingerprints.

How it works. The input is processed block by block with a compression function. A one-bit change in the input alters about half the output bits (avalanche effect).

Figure 21 | Hashing: fixed-size, one-way digest

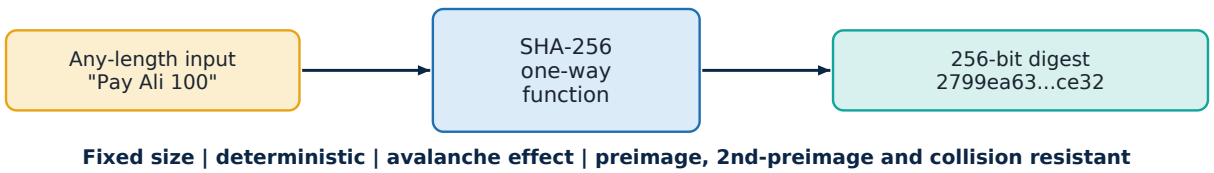


Figure 21. *Hashing: fixed-size, one-way digest*

Term	Meaning
Digest / hash	Fixed-size output.
Preimage resistance	Given h, cannot find m with H(m) = h.
Second-preimage resistance	Given m1, cannot find different m2 with the same hash.
Collision resistance	Cannot find any two inputs with the same hash.
Avalanche effect	Small input change causes large output change.

Practical applications. File integrity, password storage (with salt and slow hashing), digital signatures, blockchain.

Common misconception
Hashing is a kind of encryption.

Security implications. Hashing is one-way and keyless; encryption is reversible with a key.

Table 35. Hashing versus encryption

Feature	Hashing	Encryption
Reversible?	No	Yes, with key
Key?	No (HMAC adds one)	Yes
Output size	Fixed	Related to input
Purpose	Integrity, fingerprint	Confidentiality

SHA family

Table 36. Hash algorithms

Algorithm	Digest	Status
MD5	128 bits	Broken; do not use for security
SHA-1	160 bits	Collisions demonstrated (2017); deprecated
SHA-256 / SHA-512 (SHA-2)	256 / 512 bits	Secure; widely used (FIPS 180-4)
SHA-3	224-512 bits	Secure; different design (Keccak)

Verified examples. SHA-256("abc") = `ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad`. SHA-256 of the empty string = `e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855`. SHA-256("Pay Ali 100") = `2799ea63602ad4a828a9703dbd797fd8a9d977b5993ca4ffa92f7627b224ce32` but "Pay Ali 101" gives `9b7057acc0415bd51bc1176b5955a5e92f2cd70752247c16d5ea417a216b56be`: a single character changes the whole digest.

Passwords
Never store plain SHA-256 of passwords. Use a salted, deliberately slow function such as scrypt, bcrypt or Argon2 (Module 12). HMAC provides keyed integrity: HMAC(K, m).

11.2 Digital Signatures

In simple words. A digital signature proves who wrote a message and that nobody changed it.

Formal definition. A digital signature is a value computed from a message and the signer's private key that anyone can verify with the signer's public key, providing authentication, integrity and non-repudiation.

Everyday analogy. A handwritten signature plus a tamper-proof seal on every page.

How it works. Signing: hash the message, then transform the hash with the private key. Verification: recompute the hash, recover the signed hash using the public key, and compare. Common schemes: RSA-PSS, ECDSA, EdDSA (FIPS 186-5).

Figure 22 | Digital signature: sign and verify

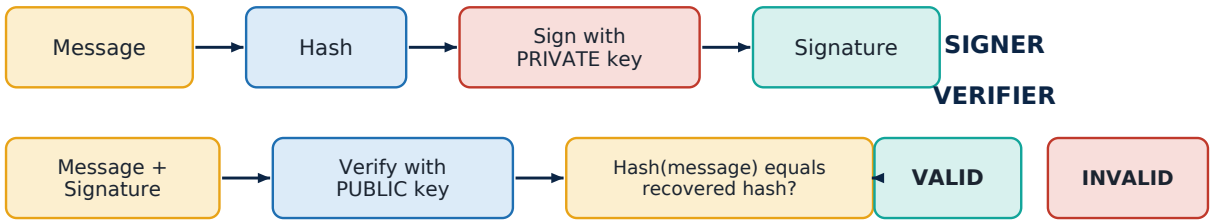


Figure 22. *Digital signature: sign and verify*

Term	Meaning
Sign	Create signature with private key.
Verify	Check signature with public key.
Non-repudiation	Signer cannot credibly deny signing.

Practical applications. Software updates, signed PDFs, e-tax filing, code signing.

> **Common misconception**
> A signature hides the message.

Security implications. A signature does not encrypt; the message stays readable unless separately encrypted.

11.3 Public Key Infrastructure (PKI)

In simple words. PKI is the system that tells you a public key really belongs to the named person or website.

Formal definition. PKI is the set of roles, policies, hardware, software and procedures needed to create, manage, distribute, use, store and revoke digital certificates (X.509, RFC 5280).

Everyday analogy. A passport system: governments (CAs) vouch that the photo and name belong together.

How it works. A website sends its certificate; the browser checks the chain of signatures up to a trusted root, verifies the hostname, validity dates and revocation status, then uses the public key to start a secure session (TLS 1.3, RFC 8446).

Figure 23 | PKI certificate chain

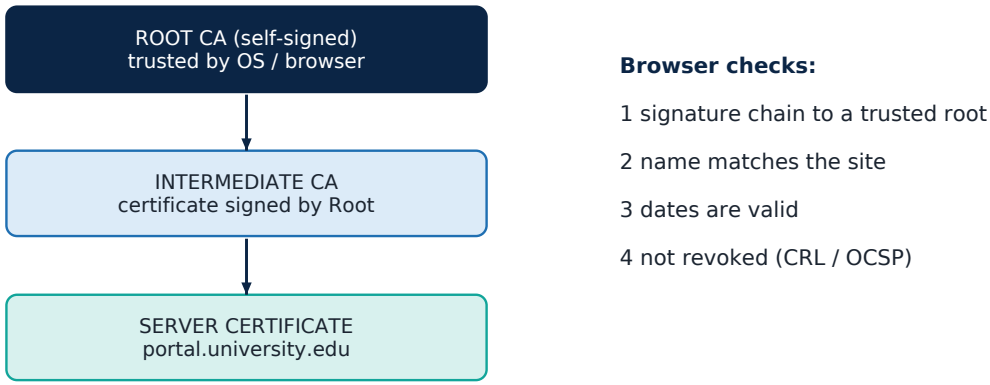


Figure 23. *PKI certificate chain*

Term	Meaning
Digital certificate	Signed statement binding a public key to an identity.
Certificate Authority (CA)	Trusted issuer of certificates.
Root / intermediate CA	Top and middle layers of the chain.
CRL	Certificate Revocation List.
OCSP	Online Certificate Status Protocol.
Trust model	How trust is established: hierarchical CA, web of trust, trust on first use.

Practical applications. HTTPS, VPNs, smart cards, signed e-mail.

Common misconception
The padlock means the website is safe.

Security implications. It only means the connection is encrypted to the named site; the site may still be malicious.

Certificate contents include version, serial number, issuer, validity period, subject, subject public key, extensions such as Subject Alternative Name (SAN) and Key Usage, and the CA signature. **Revocation** handles compromised certificates before expiry: clients consult CRLs or OCSP; OCSP stapling lets the server supply a recent signed response.

Case Study: Expired certificate on the student portal

Background	The portal certificate expires during the registration week.
Organisation and environment	University portal behind a reverse proxy.
Information assets	Portal availability and student trust.
Security problem	Browsers show a full-page warning.
Threats and vulnerabilities	Threat: user confusion, attackers exploiting warning fatigue. Vulnerability: manual renewal.
Potential impact	Registration disrupted; some students bypass warnings.
Detection	Helpdesk receives complaints; monitoring did not alert.
Investigation	Certificate details show notAfter in the past.
Response	Issue a new certificate, install it, restart the proxy.
Prevention	Automate renewal and add expiry monitoring 30 days ahead.
Lessons learned	Certificates need lifecycle management. Fictional scenario.

Practical 18: SHA-256 Hashing and File Integrity

Learning objective	Generate hashes, observe avalanche, detect file change.
Software and tools	Python 3.
Environment	Terminal.
Background	Integrity checking starts with hashing.

Step-by-step procedure

1. Run sha256_lab.py.
2. Compare digests for 100 and 101.
3. Check file integrity before and after change.
4. On Linux: sha256sum sample.txt

Program: sha256_lab.py

```
# SHA-256 hashing, avalanche effect and file integrity check
import hashlib
print(hashlib.sha256(b"abc").hexdigest())
a = hashlib.sha256(b"Pay Ali 100").hexdigest(); b = hashlib.sha256(b"Pay Ali 101").hexdigest()
print(a); print(b)
diff = bin(int(a, 16) ^ int(b, 16)).count("1"); print("Bits different:", diff, "of 256")
open("sample.txt", "w").write("Official notice\n")
def file_hash(path):
    h = hashlib.sha256()
    with open(path, "rb") as f:
        for block in iter(lambda: f.read(4096), b""): h.update(block)
    return h.hexdigest()
original = file_hash("sample.txt")
open("sample.txt", "a").write("!")
print("Integrity intact:", file_hash("sample.txt") == original)
```

Output

```
ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
2799ea63602ad4a828a9703dbd797fd8a9d977b5993ca4ffa92f7627b224ce32
9b7057acc0415bd51bc1176b5955a5e92f2cd70752247c16d5ea417a216b56be
Bits different: 136 of 256
Integrity intact: False
```

Output interpretation	About half the bits differ (136 of 256 here); modifying the file changes the hash.
Verification procedure	Run sha256sum and compare.
Common errors	Printing text with newline differences.
Troubleshooting	Hash bytes exactly.
Security precautions	Download hashes only from trusted channels.
Practical conclusion	Hashes detect change, not authorship.
Student questions	1) Why is the file hash different after appending "!"? 2) Why is MD5 unsuitable? 3) What is the avalanche effect?

Practical 19: Digital Signature Creation and Verification

Learning objective	Sign a message and verify it; detect tampering.
Software and tools	Python 3, cryptography.
Environment	Terminal.
Background	Signing proves origin and integrity.

Step-by-step procedure

1. Run signature_lab.py.
2. Verify the true message and the altered message.

Program: signature_lab.py (Ed25519)

```
# Ed25519 digital signature: sign, verify, detect tampering
from cryptography.hazmat.primitives.asymmetric.ed25519 import Ed25519PrivateKey
from cryptography.exceptions import InvalidSignature
priv = Ed25519PrivateKey.generate(); pub = priv.public_key()
msg = b"Transcript issued to student 2024-CS-017"
sig = priv.sign(msg)
print("Signature length:", len(sig), "bytes")
for m in (msg, b"Transcript issued to student 2024-CS-018"):
    try: pub.verify(sig, m); print("VALID  :", m.decode())
    except InvalidSignature: print("INVALID:", m.decode())
```

Output

```
Signature length: 64 bytes
VALID  : Transcript issued to student 2024-CS-017
INVALID: Transcript issued to student 2024-CS-018
```

Output interpretation	Valid for the original message; invalid for the altered one. Signature length is 64 bytes.
Verification procedure	Change one character and re-verify.
Common errors	Public/private key confusion.
Troubleshooting	Sign with private, verify with public.
Security precautions	Protect the private key.
Practical conclusion	Signatures provide integrity and non-repudiation.
Student questions	1) Why does a changed message fail? 2) Why is the private key never shared? 3) How is this different from encryption?

Practical 20: TLS Certificate Inspection

Learning objective	Inspect a certificate and read its chain.
Software and tools	OpenSSL, browser.
Environment	Local lab; for a real website use a browser or openssl s_client where network access is permitted.
Background	Reading certificates is a daily security skill.

Step-by-step procedure

1. Create a local test certificate and inspect it with openssl.
2. Open any HTTPS site in a browser, click the padlock, and view the certificate chain.
3. Optional (networked): openssl s_client -connect example.org:443 -showcerts

Script: cert_lab.sh

```
# Create and inspect a local self-signed certificate (offline TLS-certificate lab)
openssl req -x509 -newkey rsa:2048 -nodes -keyout lab.key -out lab.crt -days 30 \
  -subj "/CN=portal.lab.local/O=Lab University" -addext "subjectAltName=DNS:portal.lab.local"
    ↪ 2>/dev/null
openssl x509 -in lab.crt -noout -subject -issuer -dates -ext subjectAltName
openssl x509 -in lab.crt -noout -fingerprint -sha256 | cut -c1-60
```

Output

```
subject=CN = portal.lab.local, O = Lab University
issuer=CN = portal.lab.local, O = Lab University
notBefore=Nov 22 04:09:55 2026 GMT
notAfter=Dec 22 04:09:55 2026 GMT
X509v3 Subject Alternative Name:
    DNS:portal.lab.local
sha256 Fingerprint=77:77:57:AD:1B:09:22:42:32:60:0F:E4:3B:B2
```

Output interpretation	Subject and issuer are identical (self-signed); dates and fingerprint will differ on your machine.
Verification procedure	Compare with a CA-issued certificate: issuer differs from subject and the chain has several levels.
Common errors	openssl missing.
Troubleshooting	Install OpenSSL.
Security precautions	Inspect only certificates of sites you are allowed to connect to.
Practical conclusion	Chains of trust give assurance.
Student questions	1) Why is a self-signed cert not trusted by browsers? 2) What is SAN? 3) What does revocation do?

Module 11 Revision

1. Module summary

Hashes provide fixed-size one-way fingerprints; signatures combine hashing and private-key operations to give integrity, authentication and non-repudiation; PKI binds keys to identities using certificates and CAs.

2. Key definitions

Term	Definition
Hash	One-way digest.
Digital signature	Private-key-signed digest.
Certificate	Signed key-identity binding.
CA	Trusted issuer.

3. Important terminology

Collision, preimage, chain of trust, OCSP.

4. Comparison table

Table 37. Comparison for Module 11

	Hash	MAC	Signature
Key	None	Shared	Key pair
Gives	Integrity	Integrity+authentication	Integrity+authentication+non-repudiation

5. Quick revision notes

- Sign with private; verify with public.
- Hash is not encryption.

6. Memory techniques

- Hash: "One-way, fixed-size, avalanche".

7. Important formulas

- Signature = Sign(SK, H(m))

8. Common examination mistakes

- Calling hashing encryption.
- Using MD5/SHA-1.

Module 11 Question Bank

9. Short questions (15)

S1. Define hash function.
S2. Define preimage resistance.
S3. Define collision resistance.
S4. What is avalanche effect?
S5. Hashing vs encryption.
S6. Name two secure SHA algorithms.
S7. Why is MD5 broken?
S8. Define digital signature.
S9. Which key creates a digital signature?
S10. Which key verifies?
S11. Define certificate.
S12. Define CA.
S13. What is a chain of trust?
S14. What is CRL?
S15. What is OCSP?

10. Long questions (8)

L1. Explain hash properties.
L2. Compare hashing, MAC and signatures.
L3. Explain digital signature process.
L4. Explain PKI components.
L5. Explain certificate chains.
L6. Discuss trust models.
L7. Explain certificate revocation.
L8. Discuss file integrity verification.

11. Multiple-choice questions (10)

Q1. SHA-256 digest size:

A) 160
B) 256
C) 512
D) 128

Q2. Digital signatures give:

A) Confidentiality only
B) Integrity and non-repudiation
C) Speed
D) Compression

Q3. Which key signs?

A) Public

B) Both

C) None

D) Private

Q4. Broken hash:

A) SHA-3

B) SHA-256

C) MD5

D) SHA-512

Q5. CA is:

A) Certificate Authority

B) Cache

C) Attacker

D) Cipher Algorithm

Q6. Avalanche effect means:

A) Encryption

B) Compression

C) No change

D) Small change, big output change

Q7. HTTPS padlock means:

A) Site is safe

B) Virus free

C) Official

D) Connection encrypted to named site

Q8. Revocation lists are:

A) CRC

B) CPU

C) CRL

D) CSS

Q9. Hash is:

A) Encryption

B) Reversible

C) One-way

D) Key-based

Q10. Chain ends at:

A) Root CA

B) User

C) Server

D) Router

12. Scenario-based questions (5)

C1. A download page shows a SHA-256 hash. How do you use it?

C2. A site certificate expired. Risks?

C3. Why can two files not share the same SHA-256 in practice?

C4. A signature fails. List causes.

C5. Can a hash alone authenticate a sender?

13. Practical and problem-solving exercises (3)

X1. Hash two files and compare.

X2. Inspect your university portal certificate.

X3. Draw the signing and verifying steps.

Model answers (selected short questions)

S5. Hashing is one-way and keyless; encryption is reversible with a key.

S9. The private key signs.

Detailed model answers (selected long questions)

L3. The signer hashes the message and signs the digest with the private key. The verifier recomputes the hash and uses the public key to verify the signature; match means integrity and origin are confirmed.

14. References

National Institute of Standards and Technology. (2015). *Secure Hash Standard (SHS)* (FIPS PUB 180-4). https://doi.org/10.6028/NIST.FIPS.180-4

National Institute of Standards and Technology. (2023). *Digital Signature Standard (DSS)* (FIPS PUB 186-5). https://doi.org/10.6028/NIST.FIPS.186-5

Cooper, D., et al. (2008). *Internet X.509 public key infrastructure certificate and CRL profile* (RFC 5280). https://www.rfc-editor.org/info/rfc5280

Rescorla, E. (2018). *The Transport Layer Security (TLS) protocol version 1.3* (RFC 8446). https://www.rfc-editor.org/info/rfc8446

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

Python Cryptographic Authority. (n.d.). *cryptography documentation*. https://cryptography.io/

Module 12 Authentication and Access Management

Introduction. Before a system decides what you may do, it must decide who you are. This module covers identification, authentication and authorisation; passwords and how they must be stored; tokens, factors and multi-factor authentication; protocols and sessions.

Why this topic matters. Weak authentication is the most common cause of account takeover.

Real-world relevance. Online banking login, university portal, mobile app sign-in.

Learning objectives	Expected learning outcomes
• Differentiate identification, authentication and authorisation. • Explain password storage with hashing and salting. • Explain authentication factors and MFA. • Describe tokens, TOTP and session management. • Design a secure password policy.	• Implement salted password hashing. • Verify a TOTP implementation. • Analyse authentication logs.

Prerequisites	Estimated learning time
Modules 9 and 11.	About 7 hours.

12.1 Identification, Authentication and Authorisation

In simple words. Identification says who you claim to be, authentication proves it, authorisation says what you may do.

Formal definition. Identification is claiming an identity; authentication is verifying the claim; authorisation is granting permitted actions to an authenticated identity; accounting records what was done.

Everyday analogy. Airport: stating your name (identification), showing your passport (authentication), boarding pass for a specific flight (authorisation).

How it works. Systems check credentials, create a session, and consult access rules on every request.

Figure 24 | Authentication, authorisation and factors

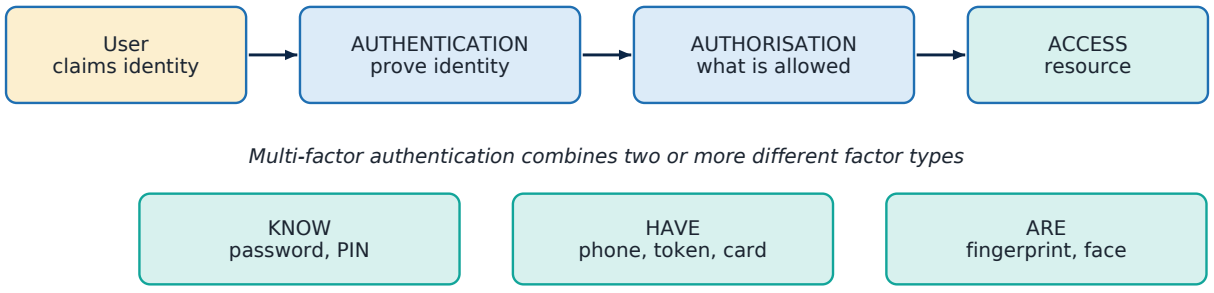


Figure 24. *Authentication, authorisation and factors*

Term	Meaning
Credential	Evidence of identity such as a password.
Factor	Category of evidence: know, have, are.
AAA	Authentication, Authorisation, Accounting.

Practical applications. Every login system.

Security implications. A user can be authenticated yet not authorised for a resource.

12.2 Password-Based Authentication and Storage

In simple words. Passwords are the most common login method. Systems must never store them in readable form.

Formal definition. Password-based authentication verifies a secret known to the user. Secure systems store only a salted, slow, one-way hash of the password.

Everyday analogy. The bank keeps a scrambled fingerprint of your PIN, never the PIN itself.

How it works. On sign-up: generate a random salt, compute hash = KDF(password, salt), store salt and hash. On login: repeat with the supplied password and compare in constant time.

Term	Meaning
Salt	Random per-user value defeating precomputed tables.
KDF	Key derivation function (scrypt, bcrypt, Argon2, PBKDF2).
Work factor	Deliberate computational cost.
Pepper	Secret value stored separately from the database.

Practical applications. Every web application.

> **Common misconception**
> Hashing passwords with plain SHA-256 is enough.

Security implications. Fast hashes allow billions of guesses per second; slow KDFs are required.

Table 38. Password protections

Technique	Defends against	Notes
Salting	Rainbow tables, identical-hash detection	Unique random value per user
Slow hashing	Brute-force guessing	scrypt, bcrypt, Argon2
Rate limiting and lockout	Online guessing	Avoid easy denial of service
MFA	Stolen passwords	Strongest single improvement
Breached-password screening	Reused passwords	NIST SP 800-63B

Secure password policy (NIST-aligned)

- Allow long passphrases (at least 8 characters, recommend 12 or more; permit 64).
- Check new passwords against lists of breached and common passwords.
- Do not force periodic changes without evidence of compromise; do not require arbitrary composition rules.
- Allow paste and password managers.
- Never store hints or enable knowledge-based recovery questions.
- Throttle failed attempts and monitor logins.

12.3 Tokens, Factors and Multi-Factor Authentication

In simple words. Something you know, something you have, something you are. Using two different kinds greatly reduces risk.

Formal definition. Multi-factor authentication requires two or more independent factors from different categories: knowledge (password), possession (phone, hardware key) and inherence (biometric).

Everyday analogy. A safe needing both a key and a code.

How it works. After the password, the system requests a second factor such as a time-based one-time password (TOTP, RFC 6238), a push approval or a FIDO2/WebAuthn security key. TOTP computes HMAC over the current 30-second time step and truncates it to 6 digits.

Term	Meaning
TOTP	Time-based one-time password.
FIDO2/WebAuthn	Phishing-resistant public-key login.
SMS code	Weaker; vulnerable to SIM swapping.

Practical applications. Banking apps, email, university systems.

> **Common misconception**
> MFA makes accounts unhackable.

Security implications. Attackers use real-time phishing and fatigue attacks; phishing-resistant factors are best.

Authentication protocols and session management

Protocols include Kerberos (ticket-based, used in enterprise networks), OAuth 2.0/OpenID Connect (delegated sign-in) and SAML (single sign-on). **Sessions** must use long random identifiers, be sent only over HTTPS, set cookie flags (*Secure*, *HttpOnly*, *SameSite*), regenerate identifiers after login, expire after inactivity and be destroyed on logout. **Account protection** includes lockout or throttling, anomaly detection, notification of new-device logins and secure recovery.

Common authentication weaknesses

- Default or reused passwords.
- Credential stuffing using leaked databases.
- Unsalted or fast hashes.
- Predictable session identifiers or session fixation.
- Account recovery via weak security questions.
- Missing rate limits and monitoring.

Case Study: Credential stuffing on a university portal

Background	After a public breach of an unrelated website, attackers try leaked passwords on the portal.
Organisation and environment	University portal for 12,000 students.
Information assets	Student accounts and personal records.
Security problem	Many students reuse passwords.
Threats and vulnerabilities	Threat: automated credential stuffing. Vulnerability: no MFA, no rate limits.
Potential impact	About 450 accounts accessed.
Detection	Authentication log shows thousands of failures from a few addresses followed by successes.
Investigation	Log analysis separates automated attempts from real logins.
Response	Block offending ranges, force password reset, enable MFA.
Prevention	Rate limiting, breached-password screening, MFA and login alerts.
Lessons learned	Monitoring plus MFA limits damage. Fictional scenario.

Practical 21: Password Security: Salted Hashing

Learning objective	Store and verify passwords safely.
Software and tools	Python 3.
Environment	Terminal.
Background	Secure storage is mandatory.

Step-by-step procedure

1. Run password_lab.py.
2. Observe different hashes for the same password.
3. Try a wrong password.

Program: password_lab.py

```
# Salted, slow password hashing with scrypt (standard library) and constant-time check
import hashlib, os, hmac
def hash_pw(pw, salt=None):
    salt = salt or os.urandom(16)
    return salt, hashlib.scrypt(pw.encode(), salt=salt, n=2**14, r=8, p=1, dklen=32)
def verify(pw, salt, stored):
    return hmac.compare_digest(hash_pw(pw, salt)[1], stored)
s1, h1 = hash_pw("Correct-Horse-9"); s2, h2 = hash_pw("Correct-Horse-9")
print("Same password, different salts -> different hashes:", h1 != h2)
print("Right password accepted:", verify("Correct-Horse-9", s1, h1))
print("Wrong password rejected:", not verify("correct-horse-9", s1, h1))
```

Output

```
Same password, different salts -> different hashes: True
Right password accepted: True
Wrong password rejected: True
```

Output interpretation	Same password with different salts yields different hashes; right password accepted, wrong rejected.
Verification procedure	Print salt and hash lengths.
Common errors	Comparing with == instead of constant-time compare.
Troubleshooting	Use hmac.compare_digest.
Security precautions	Never log passwords.

Practical conclusion	Salting plus slow hashing defeats precomputed tables and makes guessing expensive.
Student questions	1) Why use random salt? 2) What does n=2**14 control? 3) Why constant-time comparison?

Practical 22: Authentication and MFA: TOTP

Learning objective	Implement and verify TOTP using RFC 6238 test vectors.
Software and tools	Python 3.
Environment	Terminal.
Background	TOTP is the code in authenticator apps.

Step-by-step procedure

1. Run totp_lab.py.
2. Compare against RFC test vectors.
3. Change time step and observe the code.

Program: totp_lab.py

```
# TOTP (RFC 6238) - the idea behind authenticator-app codes
import hmac, hashlib, struct, time
def hotp(key, counter, digits=6):
    mac = hmac.new(key, struct.pack(">Q", counter), hashlib.sha1).digest()
    o = mac[-1] & 0x0F
    code = (struct.unpack(">I", mac[o:o+4])[0] & 0x7FFFFFFF) % 10**digits
    return str(code).zfill(digits)
def totp(key, t=None, step=30, digits=6):
    return hotp(key, int((time.time() if t is None else t) // step), digits)
key = b"12345678901234567890"                 # RFC 6238 test key (SHA-1)
print("RFC test, t=59 ->", totp(key, 59, digits=8), "(expected 94287082)")
print("RFC test, t=1111111109 ->", totp(key, 1111111109, digits=8), "(expected 07081804)")
```

Output

```
RFC test, t=59 -> 94287082 (expected 94287082)
RFC test, t=1111111109 -> 07081804 (expected 07081804)
```

Output interpretation	Both RFC vectors match (94287082 and 07081804).
Verification procedure	Compare with an authenticator app using a test secret.
Common errors	Clock drift.
Troubleshooting	Synchronise time.
Security precautions	Treat shared secrets like passwords.
Practical conclusion	MFA adds an independent factor.
Student questions	1) Why do codes expire? 2) What secret is shared? 3) What attack remains?

Practical 36: Inspecting Authentication Logs

Learning objective	Identify suspicious sign-in activity.
Software and tools	Python 3.
Environment	Terminal.
Background	Logs reveal attacks.

Step-by-step procedure

1. Run log_lab.py.
2. Change the threshold.
3. List which IPs triggered an alert.

Program: log_lab.py

```
# Generate a sample auth log, then detect brute-force behaviour (simple IDS rule)
import re, collections
log = []
for i in range(7): log.append(f"Nov 20 10:0{i}:11 srv sshd[411]: Failed password for admin from
  ↪ 203.0.113.50 port 5{i}22 ssh2")
log += ["Nov 20 10:09:01 srv sshd[500]: Accepted password for ali from 198.51.100.7 port 40022
  ↪ ssh2",
        "Nov 20 10:10:02 srv sshd[501]: Failed password for sara from 198.51.100.9 port 40100 ssh2"]
fails = collections.Counter(m.group(1) for l in log if (m := re.search(r"Failed password .* from
  ↪ (\S+)", l)))
THRESHOLD = 5
for ip, n in fails.items():
    print(f"{ip}: {n} failures ->", "ALERT possible brute force" if n >= THRESHOLD else "normal")
```

Output

```
203.0.113.50: 7 failures -> ALERT possible brute force
198.51.100.9: 1 failures -> normal
```

Output interpretation	203.0.113.50 produces 7 failures and is flagged.
Verification procedure	Check if any success follows failures.
Common errors	Regex mismatch.
Troubleshooting	Test pattern on one line.
Security precautions	Use sample logs only.
Practical conclusion	Simple rules give useful alerts.
Student questions	1) What is a brute-force signature? 2) How would you avoid false positives? 3) What next step after an alert?

Module 12 Revision

1. Module summary

Authentication proves identity; authorisation grants rights. Passwords need salted, slow hashes; MFA adds independent factors; sessions need secure cookies and lifetimes.

2. Key definitions

Term	Definition
MFA	Two or more factor types.
Salt	Random value per password.
TOTP	Time-based code.
Session	Authenticated interaction state.

3. Important terminology

KDF, factor, credential stuffing, session fixation.

4. Comparison table

Table 39. Comparison for Module 12

Factor	Example	Weakness
Know	Password	Phishing, reuse
Have	Phone	Theft, SIM swap
Are	Fingerprint	Cannot change

5. Quick revision notes

- Salting + slow hash.
- MFA beats most password attacks.

6. Memory techniques

- Factors: KHA = Know, Have, Are.

7. Important formulas

- hash = KDF(password, salt)

8. Common examination mistakes

- Calling a hash encryption.
- Forcing frequent password changes.

Module 12 Question Bank

9. Short questions (15)

S1. Define authentication.
S2. Define authorisation.
S3. Define identification.
S4. What is a salt?
S5. What is a KDF?
S6. List three factor types.
S7. What is MFA?
S8. What is TOTP?
S9. Why is SMS weaker?
S10. What is credential stuffing?
S11. What is session fixation?
S12. Name two cookie flags.
S13. What is Kerberos?
S14. Why rate-limit logins?
S15. What is FIDO2?

10. Long questions (8)

L1. Differentiate identification, authentication, authorisation.
L2. Explain secure password storage.
L3. Explain MFA and factor types.
L4. Explain TOTP.
L5. Discuss password policy.
L6. Explain session management.
L7. Discuss authentication weaknesses.
L8. Explain Kerberos/OAuth.

11. Multiple-choice questions (10)

Q1. Salt prevents:

A) Phishing
B) Brute force
C) XSS
D) Rainbow tables

Q2. Authentication verifies:

A) Backups
B) Identity
C) Rights
D) Speed

Q3. Fingerprint is:

A) Know
B) Do
C) Have
D) Are

Q4. Best password storage:

A) Salted slow KDF
B) Base64
C) MD5
D) Plaintext

Q5. TOTP is defined in:

A) SP 800-61
B) RFC 6238
C) RFC 7465
D) FIPS 197

Q6. HttpOnly cookie flag:

A) Stops JavaScript reading it
B) Allows HTTP
C) Encrypts cookie
D) Sets expiry

Q7. SMS codes are:

A) Weaker factor
B) Strongest
C) Hashes
D) Phishing-proof

Q8. Credential stuffing uses:

A) Zero-days
B) Hardware
C) Mail
D) Leaked credentials

Q9. Authorisation follows:

A) Hashing
B) Nothing
C) Authentication
D) Logging out

Q10. Constant-time compare avoids:

A) Phishing

B) DoS

C) Timing attacks

D) XSS

12. Scenario-based questions (5)

C1. A user is authenticated but cannot open a file. Explain.

C2. Passwords found in plain text in a database. Actions?

C3. Design MFA rollout for staff.

C4. Login attempts from 20 countries. Response?

C5. A developer proposes MD5 for passwords. Reply.

13. Practical and problem-solving exercises (3)

X1. Write a password policy.

X2. Run the lab and add a pepper.

X3. Analyse a log of 20 lines.

Model answers (selected short questions)

S4. A random per-user value added to the password before hashing.

S7. Authentication using two or more different factor types.

Detailed model answers (selected long questions)

L2. Generate random salt, derive hash with scrypt/bcrypt/Argon2, store salt and hash; compare in constant time; rate limit; MFA; breach screening.

14. References

Grassi, P. A., et al. (2017). *Digital identity guidelines: Authentication and lifecycle management* (NIST SP 800-63B).

M'Raihi, D., Machani, S., Pei, M., & Rydell, J. (2011). *TOTP: Time-based one-time password algorithm* (RFC 6238). https://www.rfc-editor.org/info/rfc6238

OWASP Foundation. (2021). *OWASP Top 10:2021*. https://owasp.org/Top10/

OWASP Foundation. (n.d.). *OWASP Cheat Sheet Series*. https://cheatsheetseries.owasp.org/

Stallings, W., & Brown, L. (2018). *Computer security: Principles and practice* (4th ed.). Pearson.

Python Cryptographic Authority. (n.d.). *cryptography documentation*. https://cryptography.io/

Module 13
Software Security and Buffer Overflow

Introduction. Most successful attacks exploit mistakes in software. This module explains why, focusing on the buffer overflow: what it is, how memory is arranged, how modern systems defend against it and how programmers prevent it. Everything is demonstrated safely and defensively.

Why this topic matters. Memory-safety bugs have caused some of the most damaging real-world vulnerabilities, and the prevention techniques are cheap compared with the cost of a breach.

Real-world relevance. Operating systems, browsers, routers, embedded devices and network services written in C and C++.

Learning objectives	Expected learning outcomes
• Explain secure software development principles. • Describe common software vulnerabilities. • Explain stack memory layout and how a buffer overflow arises. • Describe ASLR, DEP/NX and stack canaries. • Write and review code with bounds checking and input validation.	• Identify the root cause of an overflow. • Compare vulnerable and corrected code.

Prerequisites	Estimated learning time
C basics and Module 1.	About 7 hours.

13.1 Software Security and Secure Development

In simple words. Software security means building programs that keep working correctly even when someone feeds them hostile input.

Formal definition. Software security is the practice of designing, implementing and testing software so that it resists attack, using principles such as least privilege, input validation, defence in depth and fail-safe defaults, throughout the development lifecycle.

Everyday analogy. Building a house with locks, strong walls and a fire exit from the start, instead of adding a lock after a burglary.

How it works. Secure development lifecycle: requirements (misuse cases), design (threat modelling), implementation (secure coding), testing (static analysis, fuzzing, penetration testing), release, patching.

Term	Meaning
Vulnerability	Flaw in code or design.
Input validation	Checking data is within expected type, length and range.
Static analysis	Tools that read code to find defects.
Fuzzing	Feeding random or malformed input to find crashes.

Practical applications. Operating system kernels, web servers, mobile apps.

Common misconception
Security can be added at the end.

Security implications. Fixing a flaw after release costs far more than preventing it in design.

Common software vulnerabilities

- Buffer overflow and other memory-safety errors.
- Injection (SQL, command), covered in Module 16.
- Insecure deserialisation and race conditions.
- Hard-coded secrets and weak error handling.
- Use of vulnerable third-party components.

13.2 Buffer Overflow

In simple words. A buffer overflow happens when a program writes more data into a fixed-size memory area than it can hold, spilling into neighbouring memory.

Formal definition. A buffer overflow is a condition in which a process attempts to store data beyond the boundaries of a fixed-length buffer, overwriting adjacent memory locations (Stallings & Brown, 2018).

Everyday analogy. Pouring a litre of water into a 250 ml glass: the excess runs onto the table and spoils everything nearby.

How it works. In C, functions such as strcpy, gets and sprintf do not check lengths. In a typical stack frame, local buffers sit below saved registers and the return address. Writing past the buffer corrupts neighbouring variables, and in the worst case the saved return address, changing the program's control flow. Other forms include heap overflows, off-by-one errors, integer overflows leading to small allocations, and format-string bugs.

Figure 25 | Stack layout and the direction of an overflow

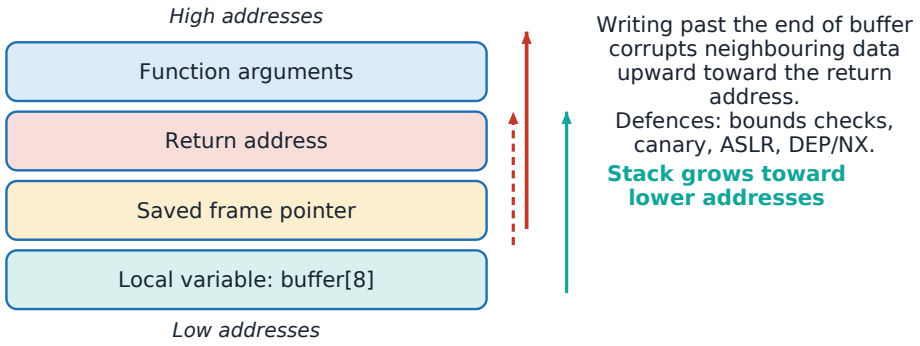


Figure 25. *Stack layout and the direction of an overflow*

Term	Meaning
Stack	Memory region holding function frames.
Return address	Where execution continues after a function returns.
Stack frame	Area for a function's arguments and locals.
Heap	Dynamically allocated memory.

Practical applications. Several historic worms and many remote-code-execution vulnerabilities came from overflows.

Common misconception
Overflows only crash programs.

Security implications. Crashes are the visible symptom; the same flaw can allow data corruption or attacker-chosen behaviour.

Defences

Table 40. Buffer overflow defences

Defence	What it does	Limitation
Bounds checking / safe functions	Prevents writing past buffer end (strncpy care, snprintf, memcpy with sizes)	Needs correct size use
Stack canary	Secret value before return address; checked on return	Can be leaked or bypassed in some cases
DEP / NX	Marks data memory non-executable	Does not stop data corruption or reuse of existing code
ASLR	Randomises memory addresses each run	Information leaks reduce its value
Memory-safe languages	Rust, Java, Python, Go prevent most overflows	Legacy code and unsafe blocks
Compiler hardening	-fstack-protector-strong, FORTIFY_SOURCE, PIE	Must be enabled in builds

Defence in depth matters: each protection can fail, so use them together with correct code and input validation.

Secure coding rules

- Always know and check the size of the destination buffer.
- Prefer snprintf, strlcpy-style functions or safe string classes over strcpy and gets.
- Validate length, type and range of every external input before use.
- Use memory-safe languages for new components where possible.
- Compile with warnings (-Wall -Wextra), sanitizers (AddressSanitizer) and run static analysis and fuzzers.
- Apply least privilege so that a compromised process can do little.

Vulnerable versus corrected code (for reading)

```
/* VULNERABLE */
void greet(const char *input) {
    char name[8];
    strcpy(name, input);          /* no length check: overflow if input >= 8 bytes */
    printf("Hello %s\n", name);
}

/* CORRECTED */
void greet(const char *input) {
    char name[8];
    snprintf(name, sizeof name, "%s", input);   /* truncates safely */
    printf("Hello %s\n", name);
}
```

Case Study: Overflow in a campus lab-booking service

Background	A small C service accepts booking requests over the lab network.
Organisation and environment	University computer lab; legacy service written in C.
Information assets	Booking data and the lab server.
Security problem	Name field copied with strcpy into 32-byte buffer.
Threats and vulnerabilities	Threat: malformed long input. Vulnerability: missing bounds check.
Potential impact	Service crashes; possible code execution risk on the server.
Detection	Crash logs repeat at the same function.
Investigation	Debugger and AddressSanitizer show the write past the buffer.
Response	Patch with length check, rebuild with stack protector and ASLR/DEP, restart service with a low-privilege account.
Prevention	Fuzz test inputs, code review, run services with least privilege.
Lessons learned	Crashes are warnings. Fictional scenario.

Practical 23: Safe Buffer Overflow Demonstration

Learning objective	Observe how an unchecked copy corrupts adjacent data and how bounds checking prevents it.
Software and tools	GCC (or any C compiler).
Environment	Your own computer or lab VM. The program simulates memory in an array so nothing real is corrupted.
Background	Seeing the corruption makes the defence obvious.

Step-by-step procedure

1. Save buffer_sim.c.
2. Compile: gcc -Wall -o buffer_sim buffer_sim.c
3. Run: ./buffer_sim
4. Change the input length and observe.
5. Fix and re-test.

Program: buffer_sim.c

```
/* Safe buffer-boundary simulation: memory is an ordinary 16-byte array, so nothing real is
  ↪ corrupted.
   Layout: bytes 0-7 = name buffer, byte 8 = is_admin flag.   Compile: gcc -Wall -o buffer_sim
     ↪ buffer_sim.c */
#include <stdio.h>
#include <string.h>
#define BUF_SIZE 8
static unsigned char mem[16];
static void show(const char *label) {
    printf("%-22s name=[", label);
    for (int i = 0; i < BUF_SIZE; i++) putchar(mem[i] ? mem[i] : '.');
    printf("]  is_admin=%d\n", mem[BUF_SIZE]);
}
static void copy_unchecked(const char *in) {          /* BUG: ignores BUF_SIZE */
    size_t i = 0;
    while (in[i] && i < sizeof mem - 1) { mem[i] = (unsigned char)in[i]; i++; }
}
static void copy_checked(const char *in) {            /* FIX: bounds check */
    size_t n = strlen(in);
    if (n >= BUF_SIZE) n = BUF_SIZE - 1;             /* leave room for terminator */
    memset(mem, 0, BUF_SIZE); memcpy(mem, in, n);
}
int main(void) {
    const char *input = "AAAAAAAAAAB";               /* 11 characters, buffer holds 7 + terminator
      ↪ */
    memset(mem, 0, sizeof mem); show("initial state");
    copy_unchecked(input); show("after unchecked copy");
    printf("  -> byte 8 (is_admin) was overwritten by input: value %d\n", mem[BUF_SIZE]);
    memset(mem, 0, sizeof mem); copy_checked(input); show("after checked copy");
    return 0;
}
```

Output

```
initial state          name=[........]  is_admin=0
after unchecked copy   name=[AAAAAAAA]  is_admin=65
  -> byte 8 (is_admin) was overwritten by input: value 65
after checked copy     name=[AAAAAAA.]  is_admin=0
```

Linux ASLR setting (2 = full randomisation)

```
2
```

Output interpretation	After the unchecked copy, byte 8 (is_admin) changes from 0 to 65 (letter A). The checked copy leaves it 0.
Verification procedure	Re-run with a 5-character input: both versions behave the same.
Common errors	Compiler warnings on unused functions.
Troubleshooting	Read warnings, they are teaching signals.
Security precautions	Do not attempt exploits on real systems; use only this simulation and lab VMs.
Practical conclusion	Bounds checks prevent corruption.
Student questions	1) Which line is the root cause? 2) What does the canary protect? 3) Why does ASLR not remove the bug?

Module 13 Revision

1. Module summary

Software security needs secure design, coding and testing. Buffer overflows arise from unchecked writes and are mitigated by bounds checking, canaries, DEP/NX, ASLR and memory-safe languages.

2. Key definitions

Term	Definition
Buffer overflow	Writing past buffer bounds.
ASLR	Address randomisation.
DEP/NX	Non-executable data.
Canary	Guard value on stack.

3. Important terminology

Stack frame, return address, heap, fuzzing.

4. Comparison table

Table 41. Comparison for Module 13

Defence	Layer
Bounds check	Code
Canary	Compiler
DEP/ASLR	OS

5. Quick revision notes

- Fix the code first.
- Use defences in depth.

6. Memory techniques

- Overflow cure: "Check, Guard, Randomise, Forbid execution".

7. Important formulas

- None; focus on size checks: copy at most sizeof(dest)-1.

8. Common examination mistakes

- Calling DEP a fix for the bug.
- Using gets().

Module 13 Question Bank

9. Short questions (15)

S1. Define software security.
S2. Define buffer overflow.
S3. What is a stack frame?
S4. What is the return address?
S5. What is a stack canary?
S6. What is DEP/NX?
S7. What is ASLR?
S8. Name two unsafe C functions.
S9. Name two safe alternatives.
S10. What is input validation?
S11. What is fuzzing?
S12. What is least privilege?
S13. What is a heap overflow?
S14. What is an off-by-one error?
S15. Why use memory-safe languages?

10. Long questions (8)

L1. Explain buffer overflow with diagram.
L2. Explain stack layout.
L3. Explain ASLR, DEP and canaries.
L4. Explain secure coding rules.
L5. Compare vulnerable and fixed code.
L6. Discuss secure development lifecycle.
L7. Explain other overflow types.
L8. Explain defence in depth.

11. Multiple-choice questions (10)

Q1. Which function is unsafe?
A) snprintf
B) strcpy
C) memcpy with size
D) fgets

Q2. Canary protects:
A) Disk
B) Return address
C) Registers
D) Heap

Q3. DEP makes data:
A) Writable
B) Faster
C) Non-executable
D) Encrypted

Q4. ASLR changes:
A) Passwords
B) Memory addresses
C) Keys
D) Code logic

Q5. Overflow root cause:
A) Fast CPU
B) Firewall
C) Long variable name
D) Missing bounds check

Q6. Best long-term fix:
A) Fix the code
B) Add banner
C) Hide the bug
D) Ignore

Q7. Fuzzing finds:
A) Hash collisions
B) Certificates
C) Passwords
D) Crashes from odd input

Q8. Memory-safe language:

A) Machine code

B) C

C) Rust

D) Assembly

Q9. On most common architectures the stack grows toward:

A) Lower addresses

B) The heap

C) The disk

D) Higher addresses

Q10. Least privilege helps by:

A) Removing logs

B) Eliminating bugs

C) Limiting damage

D) Speeding code

12. Scenario-based questions (5)

C1. A server crashes on long usernames. What next?

C2. Why can ASLR be defeated by information leaks?

C3. A team wants to disable warnings to compile. Respond.

C4. Which language for a new embedded parser?

C5. How would you test a service for overflows safely?

13. Practical and problem-solving exercises (3)

X1. Modify the simulation to show a 2-byte overflow.

X2. Write a safe string copy function.

X3. List compiler flags for hardening.

Model answers (selected short questions)

S6. DEP/NX marks data regions as non-executable to stop injected data running as code.

S7. ASLR randomises memory locations each run so attackers cannot rely on fixed addresses.

Detailed model answers (selected long questions)

L3. Canary: random value before return address, checked on return; ASLR: randomised addresses; DEP/NX: data pages non-executable. Together they make exploitation harder; none replaces fixing the bug.

14. References

Stallings, W., & Brown, L. (2018). *Computer security: Principles and practice* (4th ed.). Pearson.

Anderson, R. (2020). *Security engineering: A guide to building dependable distributed systems* (3rd ed.). Wiley.

Pfleeger, C. P., Pfleeger, S. L., & Margulies, J. (2015). *Security in computing* (5th ed.). Prentice Hall.

OWASP Foundation. (2021). *OWASP Top 10:2021*. https://owasp.org/Top10/

Module 14
Database Security and Operating System Security

Introduction. Databases hold the most valuable information; operating systems control every resource on a computer. This module covers how both enforce who may do what, using privileges, permissions and access-control models.

Why this topic matters. A strong database is useless on a weak operating system, and the reverse. Least privilege, logging and patching protect both.

Real-world relevance. Student information systems, hospital records, banking databases, servers and workstations.

Learning objectives	Expected learning outcomes
• Explain database assets, threats and access control. • Use GRANT and REVOKE to manage SQL privileges. • Explain least privilege, backup and recovery. • Explain file permissions, MAC, DAC, ACL and RBAC. • Describe system hardening, patching and auditing.	• Create a database user and assign and remove privileges. • Demonstrate file permissions. • Complete an operating-system hardening checklist.

Prerequisites	Estimated learning time
Modules 1, 2 and 12.	About 9 hours.

Part A: Database Security

14.1 Database Security Fundamentals

In simple words. Database security protects stored data from misuse, loss and unauthorised change, while keeping it available to the right people.

Formal definition. Database security is the collection of controls, policies and procedures that protect the confidentiality, integrity and availability of a database and the data it contains.

Everyday analogy. A bank vault: only certain staff may open certain boxes, every access is logged, and copies of the contents are kept elsewhere.

How it works. Layers: network and host security, database authentication, authorisation (privileges and roles), encryption at rest and in transit, views to restrict columns and rows, auditing, and backup and recovery.

Figure 26 | Database access control flow

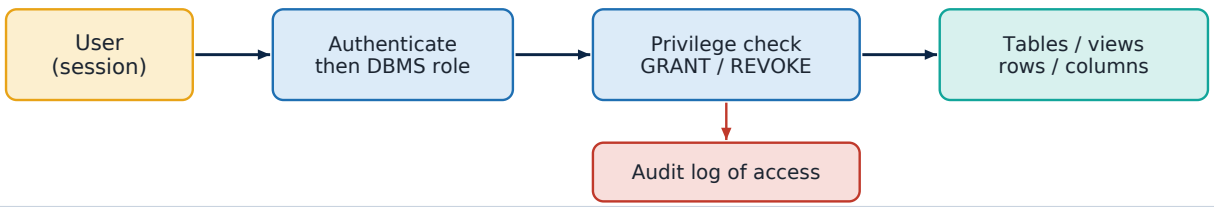


Figure 26. *Database access control flow*

Every request is authenticated, checked against privileges, and recorded in an audit log.

Term	Meaning
Database asset	Data, schema, DBMS, backups, credentials.
Privilege	Right to perform an action on an object.
Role	Named set of privileges assigned to users.
View	Virtual table exposing only chosen columns/rows.

Practical applications. Student records, patient data, payment systems.

Common misconception
Firewalls alone protect databases.

Security implications. Most database attacks use legitimate credentials, injection or excess privilege.

Threats

- SQL injection (Module 16), excessive privileges, privilege abuse by insiders.
- Weak or default credentials, unpatched DBMS.
- Unprotected backups, exposed database ports.
- Data leakage through logs and test copies.
- Loss or corruption without tested recovery.

Access control with GRANT and REVOKE

SQL controls privileges with **GRANT** (give) and **REVOKE** (take away). The principle of **least privilege** says each account receives only the rights needed for its task. Syntax is similar across systems; the examples use PostgreSQL syntax (MySQL and others differ in details such as user creation).

SQL listing (PostgreSQL)

```
-- run as an administrator in a TEST database
CREATE DATABASE uni_demo;
\c uni_demo
CREATE TABLE students(id SERIAL PRIMARY KEY, name TEXT, cgpa NUMERIC(3,2), phone TEXT);
INSERT INTO students(name,cgpa,phone) VALUES ('Ali',3.40,'0300-0000001'),('Sara',3.75,'0300-0000002'
  ↪ );

-- 1. create a limited user
CREATE USER report_user WITH PASSWORD 'Use-a-long-random-passphrase-here';

-- 2. grant only what is needed
GRANT CONNECT ON DATABASE uni_demo TO report_user;
GRANT USAGE ON SCHEMA public TO report_user;
GRANT SELECT (id, name, cgpa) ON students TO report_user;   -- no phone column

-- 3. verify
SET ROLE report_user;
SELECT id, name, cgpa FROM students;      -- allowed
SELECT phone FROM students;               -- ERROR: permission denied for table students
RESET ROLE;

-- 4. remove the privilege
REVOKE SELECT (id, name, cgpa) ON students FROM report_user;
SET ROLE report_user; SELECT name FROM students;  -- ERROR: permission denied
RESET ROLE;
```

Verification note
These SQL commands could not be executed in the authoring environment because no database server was installed there. Run them in your own test database; the comments show the expected results. Never run them on a production database.

Policies, backup and recovery, sensitive data

- Policy: documented roles, approval for new privileges, regular privilege reviews, separation of duties.
- Backup: follow 3-2-1, encrypt backups, protect backup access, and **test restores** regularly.
- Recovery objectives: RPO (how much data loss is acceptable) and RTO (how long recovery may take).
- Sensitive information: classify, encrypt, mask or tokenise personal data; limit copies; audit access.

Part B: Operating System Security

14.2 Operating System Security and Access Control Models

In simple words. The operating system decides which user and program may read, write or run each file and resource.

Formal definition. Operating system security comprises the mechanisms that enforce isolation, authentication, access control, auditing and protection of processes, memory and files.

Everyday analogy. A building manager who issues key cards, decides which doors each card opens and keeps a log.

How it works. Every request for a resource passes through a reference monitor that applies the access-control model and the system policy, and generates audit events.

Figure 27 | MAC, DAC and RBAC compared

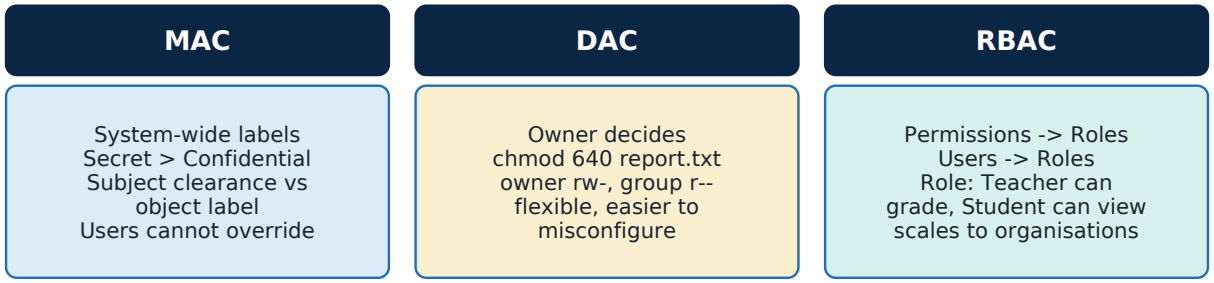


Figure 27. *MAC, DAC and RBAC compared*

Term	Meaning
DAC	Discretionary Access Control: the owner decides.
MAC	Mandatory Access Control: system-wide policy using labels; users cannot override.
ACL	Access Control List: list of (subject, permission) entries attached to an object.
RBAC	Role-Based Access Control: permissions attach to roles; users get roles.

Practical applications. Linux permissions and ACLs, Windows NTFS, SELinux (MAC), enterprise RBAC.

Security implications. The ordinary owner can break DAC by mistake; MAC and RBAC reduce that risk.

Table 42. Comparison of MAC, DAC, ACL and RBAC

Model	Who decides	Example	Strength	Weakness
DAC	Owner	chmod, file sharing	Flexible	Misconfiguration, Trojan risk
MAC	System policy	SELinux, labelled military systems	Strong, centrally enforced	Rigid, complex
ACL	Per-object list	setfacl, NTFS ACL	Fine-grained	Hard to audit at scale
RBAC	Roles defined by admin	Teacher / Student / Registrar	Scales, auditable	Role explosion if poorly designed

Unix file permissions

Each file has permissions for the **owner**, **group** and **others**: read (r = 4), write (w = 2), execute (x = 1). Mode 640 means owner rw-, group r--, others no access. Directories need execute to be entered. Example: `chmod 640 report.txt`; `chmod 750 shared`.

Hardening, patching, logging and auditing

- **Hardening**: remove unneeded software and services, disable default accounts, enforce strong authentication, enable the host firewall, apply secure baselines (for example CIS benchmarks).
- **Patch management**: inventory, prioritise by severity and exposure, test, deploy, verify, and keep a rollback plan.
- **Logging and auditing**: record logins, privilege changes and access to sensitive files; forward logs to a central, protected store; review and alert (NIST SP 800-92).
- **Least privilege**: daily work with standard accounts, sudo for specific tasks.

Case Study: Over-privileged report account at a university

Background	A reporting tool connects to the student database.
Organisation and environment	University IT; one shared database account for all tools.
Information assets	Student grades, phone numbers, fee data.
Security problem	The tool account has full privileges.
Threats and vulnerabilities	Threat: compromised tool or insider. Vulnerability: excess privilege and shared credentials.
Potential impact	Changes to grades possible with no trace.
Detection	Audit finds grade updates not made through the portal.
Investigation	Log review traces updates to the tool account.
Response	Create separate read-only accounts, revoke write rights, rotate credentials, restore from backup.
Prevention	Least privilege, role design, quarterly privilege review, database auditing.
Lessons learned	Privilege creep is silent. Fictional scenario.

Practical 24: Database GRANT and REVOKE

Learning objective	Create a limited database user, grant column-level access and revoke it.
Software and tools	PostgreSQL (or MySQL) in a test environment.
Environment	Local VM with a test database.
Background	Privileges control exposure.

Step-by-step procedure

1. Install a database server in a VM.
2. Run the SQL listing above as administrator.
3. Check each command result against the comments.
4. Test denied access after REVOKE.

Table 43. Expected results (not executed in the authoring environment)

Command	Expected result
SELECT id,name,cgpa ...	Rows returned
SELECT phone ...	permission denied
After REVOKE: SELECT name ...	permission denied

Output interpretation	Column-level privileges prevent exposure of phone numbers; REVOKE removes access.
Verification procedure	Query as report_user before and after each change.
Common errors	Role does not exist or permission denied for admin tasks.
Troubleshooting	Connect as an admin account and re-run.
Security precautions	Use only a test database with fake data.
Practical conclusion	Least privilege is configured, not assumed.
Student questions	1) Why is GRANT ALL a bad habit? 2) How does a view help? 3) What is a role?

Practical 25: Operating System Permissions

Learning objective	Display and modify Unix permissions and umask.
Software and tools	Linux or macOS terminal.
Environment	Scratch directory.
Background	Permissions are the basic OS access control.

Step-by-step procedure

1. Save os_perm_lab.sh.
2. Run: bash os_perm_lab.sh
3. Interpret each line.
4. Try chmod 700 and chmod 755.

Script: os_perm_lab.sh

```
# File-permission demonstration in a scratch directory
cd "$(mktemp -d)"
echo "marks" > report.txt
chmod 640 report.txt; stat -c "%A  octal=%a  %n" report.txt      # owner rw-, group r--, others ---
chmod 600 report.txt; stat -c "%A  octal=%a  %n" report.txt      # owner only
chmod g+r,o+r report.txt; stat -c "%A  octal=%a  %n" report.txt  # add read for group and others
mkdir shared; chmod 750 shared; stat -c "%A  octal=%a  %n" shared
umask 027; touch newfile; stat -c "umask 027 -> %A  %n" newfile
```

Output

```
-rw-r-----  octal=640  report.txt
-rw-------  octal=600  report.txt
-rw-r--r--  octal=644  report.txt
drwxr-x---  octal=750  shared
umask 027 -> -rw-r-----  newfile
```

Optional ACL practice

```
# ACL commands (Linux, requires the acl package; not executed in the authoring environment)
setfacl -m u:ali:r report.txt    # give user ali read access
getfacl report.txt               # show ACL
setfacl -x u:ali report.txt      # remove the entry
```

Output interpretation	640 = rw-r----- ; 600 = rw------- ; umask 027 gives new files mode 640.
Verification procedure	Use ls -l to confirm.
Common errors	Operation not permitted.
Troubleshooting	Check file owner.
Security precautions	Do not chmod -R 777 anything.
Practical conclusion	Permissions are least privilege at file level.
Student questions	1) What does 750 mean? 2) Why does a directory need x? 3) How does umask work?

Operating system security checklist (practical exercise)

Table 44. OS security checklist

#	Control	Check	Done
1	Updates	Automatic security updates enabled	[ ]
2	Accounts	Daily user is not an administrator	[ ]
3	Authentication	Strong passphrase and MFA where available	[ ]
4	Firewall	Host firewall on, default deny inbound	[ ]
5	Services	Unused services and software removed	[ ]
6	Encryption	Disk encryption enabled	[ ]
7	Backup	3-2-1 backup, restore tested	[ ]
8	Logging	Security logs enabled and retained	[ ]
9	Malware	Endpoint protection active	[ ]
10	Physical	Screen lock and secure storage	[ ]

Module 14 Revision

1. Module summary

Databases protect data using authentication, least-privilege GRANT/REVOKE, views, encryption, auditing and backup. Operating systems enforce access control (DAC, MAC, ACL, RBAC), hardening, patching and logging.

2. Key definitions

Term	Definition
GRANT	Gives a privilege.
REVOKE	Removes a privilege.
DAC	Owner-controlled access.
MAC	Policy-controlled access.
RBAC	Role-based access.

3. Important terminology

Privilege, role, view, reference monitor, RPO, RTO.

4. Comparison table

Table 45. Comparison for Module 14

Model	Controlled by
DAC	Owner
MAC	System policy
RBAC	Roles
ACL	Object list

5. Quick revision notes

- Least privilege.
- Test restores.

6. Memory techniques

- Permissions: r4 w2 x1.

7. Important formulas

- chmod 640 = rw-r-----
- RPO and RTO define recovery goals

8. Common examination mistakes

- GRANT ALL to everybody.
- Untested backups.

Module 14 Question Bank

9. Short questions (15)

S1. Define database security.
S2. What is GRANT?
S3. What is REVOKE?
S4. Define least privilege.
S5. What is a role?
S6. What is a view for security?
S7. What is RPO?
S8. What is RTO?
S9. Define DAC.
S10. Define MAC.
S11. Define ACL.
S12. Define RBAC.
S13. What does chmod 640 mean?
S14. What is patch management?
S15. Why audit logs?

10. Long questions (8)

L1. Explain database threats and controls.
L2. Explain GRANT/REVOKE with examples.
L3. Explain backup and recovery.
L4. Compare DAC, MAC, ACL and RBAC.
L5. Explain Unix permissions.
L6. Explain OS hardening.

L7. Explain logging and auditing.

L8. Design RBAC for a university.

11. Multiple-choice questions (10)

Q1. REVOKE:

A) Gives rights

B) Creates users

C) Deletes tables

D) Removes rights

Q2. Mode 750 gives group:

A) rwx

B) ---

C) r-x

D) rw-

Q3. MAC policy set by:

A) User

B) System

C) Guest

D) Owner

Q4. Least privilege means:

A) No rights

B) Minimum necessary rights

C) Admin rights

D) All rights

Q5. RBAC attaches permissions to:

A) Packets

B) Keys

C) Files

D) Roles

Q6. A view can:

A) Encrypt disks

B) Create keys

C) Hide columns

D) Patch OS

Q7. Backups must be:

A) Public

B) Tested and protected

C) Untested

D) Deleted

Q8. ACL is:

A) A cipher

B) A hash

C) List of permissions on an object

D) A firewall

Q9. r w x values:

A) 4,2,1

B) 3,2,1

C) 2,4,1

D) 1,2,4

Q10. RTO is:

A) Role time order

B) Risk threshold

C) Read-only

D) Recovery time objective

12. Scenario-based questions (5)

C1. A reporting tool has full DBA rights. Fix?

C2. Teacher should enter grades but not change fees. Design roles.

C3. A backup copy sits on a public share. Actions?

C4. A user leaves the university. What to do with accounts?

C5. Choose DAC or MAC for a hospital record system and justify.

13. Practical and problem-solving exercises (3)

X1. Write GRANT/REVOKE for a Teacher role.

X2. Interpret mode 754.

X3. Complete the OS checklist for your laptop.

Model answers (selected short questions)

S1. Database security protects the confidentiality, integrity and availability of stored data.

S4. Each account gets only the privileges needed for its task.

Detailed model answers (selected long questions)

L4. DAC: owner decides; MAC: system labels and policy decide; ACL: per-object list; RBAC: permissions attached to roles. A university can use RBAC for Teacher, Student and Registrar roles.

14. References

Stallings, W., & Brown, L. (2018). *Computer security: Principles and practice* (4th ed.). Pearson.

Pfleeger, C. P., Pfleeger, S. L., & Margulies, J. (2015). *Security in computing* (5th ed.). Prentice Hall.

Joint Task Force. (2020). *Security and privacy controls for information systems and organizations* (NIST SP 800-53 Rev. 5).

Kent, K., & Souppaya, M. (2006). *Guide to computer security log management* (NIST SP 800-92).

OWASP Foundation. (n.d.). *OWASP Cheat Sheet Series*. https://cheatsheetseries.owasp.org/

Module 15
Network Security, Firewalls and Intrusion Detection

Introduction. Networks carry data between systems and are the first line of attack and defence. We study network threats, how segmentation, firewalls and honeypots reduce exposure, and how intrusion detection and prevention systems watch for attacks.

Why this topic matters. Most breaches involve a network step: scanning, interception, lateral movement or command-and-control. Network defence reduces attacker freedom and speeds detection.

Real-world relevance. Campus networks, corporate offices, data centres, home Wi-Fi.

Learning objectives	Expected learning outcomes
• State network security objectives. • Describe packet sniffing, spoofing, MITM and DoS. • Explain segmentation, DMZ and honeypots. • Explain firewall types, basing, location and configuration. • Explain host-based IDS/IPS, IDS versus IPS and monitoring.	• Write safe firewall rules. • Inspect traffic in a controlled lab. • Analyse logs and recognise alerts.

Prerequisites	Estimated learning time
Modules 1, 2, 11, 12.	About 10 hours.

15.1 Network Security Fundamentals and Common Attacks

In simple words. Network security keeps data safe while it moves and keeps network services working.

Formal definition. Network security is the set of policies, devices and practices that protect the usability, confidentiality, integrity and availability of networks and the data travelling on them.

Everyday analogy. A road system with checkpoints, fences between districts, traffic cameras and emergency lanes.

How it works. Objectives: confidentiality (encryption), integrity (authentication codes), availability (redundancy, DoS defence), access control, accountability. Defences are layered: segmentation, firewalls, IDS/IPS, secure protocols, monitoring.

Term	Meaning
Packet sniffing	Capturing network traffic.
Spoofing	Forging addresses or identities (IP, ARP, DNS).
Man-in-the-middle (MITM)	Attacker relays and possibly alters communication.
Denial of Service (DoS/DDoS)	Overwhelming a service.
Segmentation	Dividing a network into zones.
Honeypot	Decoy system to attract and study attackers.

Practical applications. Defence of Wi-Fi, servers, campus LANs.

> **Common misconception**
> Switches stop sniffing completely.

Security implications. Switched networks reduce casual sniffing but ARP spoofing, port mirroring and compromised devices still expose traffic.

Table 46. Common network attacks and defences

Attack	How it works	Main defence
Packet sniffing	Capture of unencrypted traffic	TLS, VPN, switch port security
IP/ARP/DNS spoofing	False source identity	Ingress filtering, DAI, DNSSEC, authentication
MITM	Interception and relay	Certificate validation, HSTS, mutual authentication
DoS/DDoS	Flood of requests	Rate limiting, CDN, redundancy, upstream filtering
Port scan / reconnaissance	Probing open services	Firewall, IDS, minimal services

15.2 Firewalls

In simple words. A firewall allows or blocks traffic according to rules, like a security gate at the edge of a network.

Formal definition. A firewall is a network security device or software that monitors and controls incoming and outgoing network traffic based on predetermined security rules (NIST SP 800-41).

Everyday analogy. A gatehouse checking each vehicle against a list of permitted visitors and destinations.

How it works. Rules are checked in order; the first match decides. A good policy ends with default deny. Packet filters examine headers; stateful inspection tracks connections; application (proxy) firewalls inspect content; next-generation firewalls add application awareness and intrusion prevention.

Figure 28 | Firewall architecture with DMZ

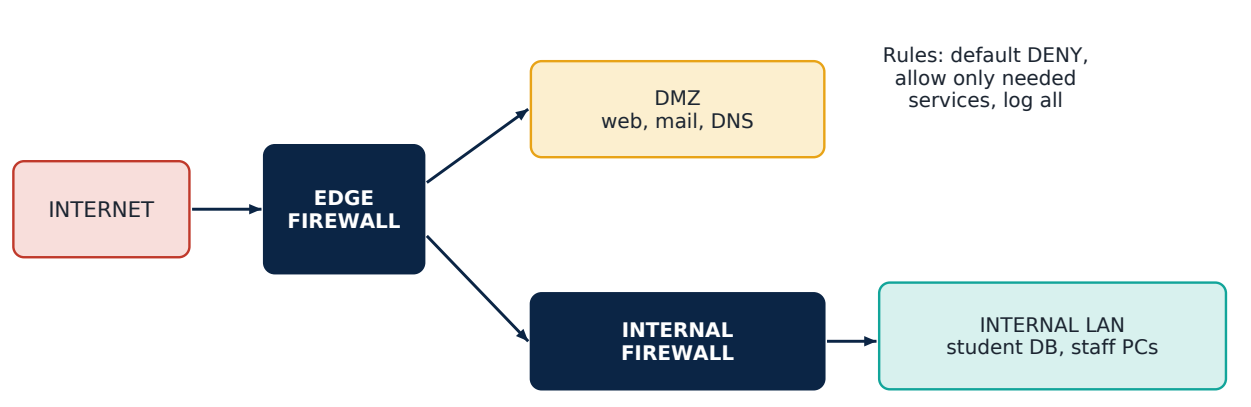


Figure 28. *Firewall architecture with DMZ*

Term	Meaning
Packet filtering	Decisions from IP, port and protocol.
Stateful inspection	Tracks connection state; allows replies to permitted requests.

Term	Meaning
Application / proxy firewall	Acts as intermediary and understands the protocol.
DMZ	Zone for public-facing servers.
Default deny	Block everything not explicitly allowed.

Practical applications. Edge firewalls, host firewalls, cloud security groups.

Common misconception
A firewall stops all attacks.

Security implications. A firewall cannot stop attacks that use allowed traffic, insiders, or malware already inside.

Table 47. Firewall types, basing and location

Aspect	Options
Types	Packet filter; stateful; application proxy; next-generation
Basing	Network firewall (hardware/virtual); host-based (OS or agent); cloud/service-based
Location	Network edge; between internal zones; in front of a DMZ; on each host

Safe firewall configuration

- Start from default deny for inbound traffic; allow only required services from required sources.
- Filter outbound traffic too (egress filtering) to limit data theft and malware callbacks.
- Allow established and related return traffic via stateful rules.
- Place public servers in a DMZ separate from the internal LAN.
- Document each rule with owner and reason; review regularly; log drops.
- Always keep console access before testing remote rules, so you do not lock yourself out.

Basic inbound and outbound filtering

```
# Reference rules in nftables syntax (run only in your own VM; not executed in the authoring
  ↪ environment)
nft add table inet filter
nft add chain inet filter input   '{ type filter hook input priority 0; policy drop; }'
nft add chain inet filter output  '{ type filter hook output priority 0; policy accept; }'
nft add rule  inet filter input ct state established,related accept   # replies to our requests
nft add rule  inet filter input iif lo accept                         # loopback
nft add rule  inet filter input tcp dport 22 ip saddr 192.168.56.0/24 accept   # SSH from lab
  ↪ network only
nft add rule  inet filter input tcp dport { 80, 443 } accept          # web server
nft add rule  inet filter input log prefix "DROP-IN " counter

# Equivalent idea on Ubuntu with ufw
ufw default deny incoming && ufw default allow outgoing
ufw allow from 192.168.56.0/24 to any port 22 proto tcp
ufw allow 80,443/tcp && ufw enable
```

Verification note
The firewall commands above could not be run in the authoring environment (no firewall tools installed). Test them in a lab VM with console access. Verify with: nft list ruleset, and test allowed and blocked ports from a second VM.

15.3 Intrusion Detection and Prevention

In simple words. An IDS watches for attacks and raises alerts; an IPS can also block them.

Formal definition. An intrusion detection system monitors network or host activity for malicious behaviour or policy violations and reports it. An intrusion prevention system additionally takes automatic action, such as blocking traffic, because it sits in-line.

Everyday analogy. A burglar alarm (IDS) versus an alarm connected to automatic steel shutters (IPS).

How it works. Detection methods: **signature-based** (known patterns), **anomaly-based** (deviation from a baseline) and **stateful protocol analysis**. Network IDS (NIDS) inspects traffic via a tap or mirror port. **Host-based IDS (HIDS)** runs on a computer and examines logs, processes, system calls and file integrity. **Distributed HIDS** sends host events to a central manager for correlation. **Host-based IPS (HIPS)** blocks malicious actions on the host.

Figure 29 | Network intrusion detection pipeline

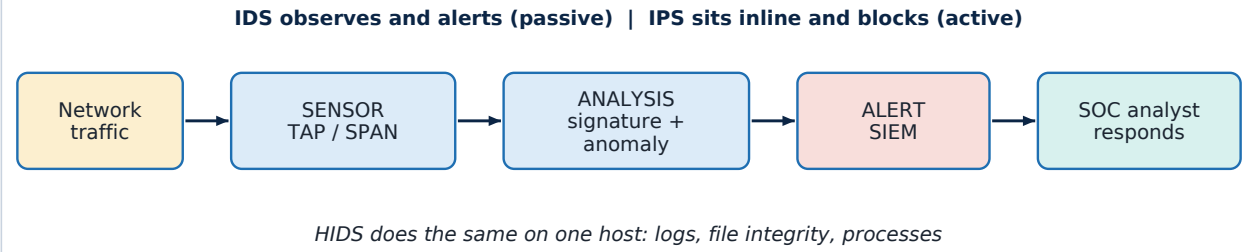


Figure 29. *Network intrusion detection pipeline*

Term	Meaning
NIDS	Monitors network traffic.
HIDS	Monitors a single host.
Distributed HIDS	Multiple agents with central correlation.
HIPS	Host-level prevention.
False positive	Alert for harmless activity.
False negative	Missed real attack.
SIEM	Central log collection and correlation.

Practical applications. University SOC, hospital networks, cloud workloads.

Common misconception
More alerts mean better security.

Security implications. Alert quality matters; tune rules to reduce false positives.

Table 48. IDS versus IPS

	IDS	IPS
Placement	Passive (tap/mirror)	In-line
Action	Alert	Alert and block
Risk	May miss response time	False positive can block legitimate traffic

Security monitoring and network defence strategy

- Defence in depth: segmentation, firewalls, IDS/IPS, endpoint protection, encryption.
- Centralised logging with time synchronisation; baseline normal behaviour.
- Zero-trust thinking: authenticate and authorise every connection.
- Patch exposed systems quickly; minimise attack surface.
- Practice incident response (Module 2 and Section 15.5).

Honeypots

A honeypot is a decoy system with no legitimate users; every interaction is suspicious. **Low-interaction** honeypots emulate a few services; **high-interaction** honeypots are real systems that must be strictly isolated. Honeypots detect scanning, collect attacker techniques and waste attacker time. They must be isolated, monitored and never placed where compromise could spread.

Incident response overview

Figure 30 | Incident response lifecycle

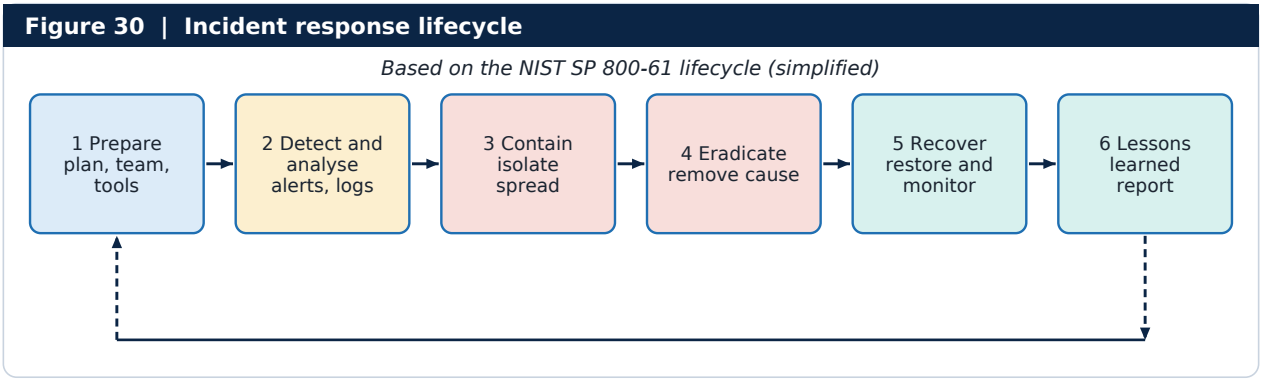


Figure 30. *Incident response lifecycle*

Case Study: Port scan and brute force against a campus server

Background	Campus servers face the Internet for remote access.
Organisation and environment	University network with edge firewall and one DMZ.
Information assets	Staff VPN and admin interface.
Security problem	Repeated connection attempts to many ports and to SSH.
Threats and vulnerabilities	Threat: automated scanning. Vulnerability: SSH exposed to the whole Internet.
Potential impact	Possible compromise of an admin account.
Detection	Firewall log shows one address touching many ports in seconds; auth log shows repeated failures.
Investigation	Log analysis confirms the pattern; no successful login.
Response	Block the source, restrict SSH to the VPN range, enable MFA, add alert rule.
Prevention	Minimal exposure, default deny, central log monitoring.
Lessons learned	Detection plus reduction of exposure stops most automated attacks. Fictional scenario.

Practical 26: Firewall Configuration

Learning objective	Write and test a default-deny inbound policy in a lab VM.
Software and tools	Linux VM (Ubuntu), nftables or ufw, a second VM or host to test from.
Environment	Isolated host-only network.
Background	Rules enforce least network exposure.

Step-by-step procedure

1. Take a VM snapshot.
2. Keep a console session open.
3. Apply the rules in the listing.
4. From the second machine test ports 22, 80 and a closed port.
5. Review the log and counters.

Table 49. Expected firewall behaviour (reference)

Test	Expected
SSH from lab subnet	Allowed
SSH from another subnet	Dropped
HTTP/HTTPS	Allowed
Unlisted port	Dropped and logged

Output interpretation	Only listed services answer; others time out.
Verification procedure	nft list ruleset; nmap against your own VM only.
Common errors	Locked out of SSH.
Troubleshooting	Use the console and flush ruleset.
Security precautions	Only test machines you own; never scan others.
Practical conclusion	Default deny is the safest posture.
Student questions	1) Why use default deny? 2) What does established,related mean? 3) Why log drops?

Practical 27: Wireshark Traffic Inspection

Learning objective	Capture and inspect traffic in a controlled lab.
Software and tools	Wireshark.
Environment	Your own computer, loopback or an isolated lab network.
Background	Seeing packets explains protocols and why encryption matters.

Step-by-step procedure

1. Start Wireshark and select the loopback or lab interface.
2. Browse to an HTTP page you host locally.
3. Apply the filters below.
4. Compare HTTP (readable) with HTTPS (encrypted payload).
5. Save the capture for the report.

Table 50. Common Wireshark display filters

Display filter	Purpose
ip.addr == 192.168.56.10	Traffic to/from a host
tcp.port == 80	HTTP traffic
http.request	HTTP requests
dns	DNS queries
tcp.flags.syn == 1 && tcp.flags.ack == 0	Connection attempts (SYN)
tls.handshake	TLS handshake

Output interpretation	Plain HTTP shows readable data; HTTPS shows only a handshake and encrypted application data.
Verification procedure	Follow TCP Stream on an HTTP capture.
Common errors	No interfaces listed.
Troubleshooting	Install capture driver and run with the required permission.
Security precautions	Capture only traffic you own or are authorised to inspect; handle captures as sensitive data.

Practical conclusion	Packet inspection shows the value of encryption.
Student questions	1) What is a SYN scan pattern? 2) Why is HTTP unsafe? 3) What can you still learn from HTTPS metadata?

Practical 28: Security Log Analysis

Learning objective	Analyse firewall and authentication logs.
Software and tools	Python 3.
Environment	Terminal.
Background	Logs reveal attacks and mistakes.

Step-by-step procedure

1. Run fwlog_lab.py.
2. Add more log lines and rerun.
3. Change the scan threshold.

Program: fwlog_lab.py

```
# Summarise a sample firewall log: blocked ports, noisy sources, and likely port scan
import re, collections
log = """\
Nov 20 11:00:01 fw DROP IN=eth0 SRC=203.0.113.9 DST=10.0.0.5 PROTO=TCP DPT=22
Nov 20 11:00:02 fw DROP IN=eth0 SRC=203.0.113.9 DST=10.0.0.5 PROTO=TCP DPT=23
Nov 20 11:00:02 fw DROP IN=eth0 SRC=203.0.113.9 DST=10.0.0.5 PROTO=TCP DPT=80
Nov 20 11:00:03 fw DROP IN=eth0 SRC=203.0.113.9 DST=10.0.0.5 PROTO=TCP DPT=443
Nov 20 11:00:03 fw DROP IN=eth0 SRC=203.0.113.9 DST=10.0.0.5 PROTO=TCP DPT=3389
Nov 20 11:01:10 fw DROP IN=eth0 SRC=198.51.100.4 DST=10.0.0.5 PROTO=UDP DPT=53
Nov 20 11:02:00 fw ACCEPT IN=eth0 SRC=198.51.100.77 DST=10.0.0.8 PROTO=TCP DPT=443""".splitlines()
drops = [dict(re.findall(r"(\w+)=(\S+)", l)) for l in log if " DROP " in l]
by_src = collections.Counter(d["SRC"] for d in drops)
ports = collections.defaultdict(set)
for d in drops: ports[d["SRC"]].add(d["DPT"])
for src, n in by_src.most_common():
    flag = "POSSIBLE PORT SCAN" if len(ports[src]) >= 4 else "ok"
    print(f"{src:15s} drops={n} distinct_ports={len(ports[src])} -> {flag}")
```

Output

```
203.0.113.9     drops=5 distinct_ports=5 -> POSSIBLE PORT SCAN
198.51.100.4    drops=1 distinct_ports=1 -> ok
```

Output interpretation	203.0.113.9 touched 5 different ports and is flagged as a possible scan.
Verification procedure	Cross-check with authentication logs.
Common errors	Regular expression does not match.
Troubleshooting	Print parsed fields.
Security precautions	Use sample data only.
Practical conclusion	Simple analytics create actionable alerts.
Student questions	1) What distinguishes a scan from normal traffic? 2) How can false positives be reduced? 3) What next after the alert?

Practical 29: Intrusion Detection Concepts and Honeypot Demonstration

Learning objective	Demonstrate signature detection, host integrity monitoring and a controlled honeypot.
Software and tools	Python 3.
Environment	Terminal; honeypot binds only to 127.0.0.1.
Background	Hands-on detection concepts.

Step-by-step procedure

1. Run ids_lab.py.
2. Run honeypot_lab.py.
3. Add a new signature and test.

Program: ids_lab.py

```
# Intrusion-detection concepts: signature rule, threshold rule and host file-integrity monitoring
  ↪ (HIDS)
import hashlib, os, tempfile
events = ["GET /index.html", "GET /search?q=students", "GET /search?q=' OR '1'='1", "GET
  ↪ /../../etc/passwd", "GET /about"]
signatures = {"SQL injection pattern": "' OR '", "Path traversal pattern": "../"}
print("== Signature-based detection ==")
for e in events:
    hits = [name for name, sig in signatures.items() if sig in e]
    print(("ALERT " + ", ".join(hits) if hits else "ok   ").ljust(40), e)
print("== Host-based integrity check ==")
d = tempfile.mkdtemp()
files = {"config.ini": "mode=secure\n", "app.py": "print('hello')\n"}
for n, c in files.items(): open(os.path.join(d, n), "w").write(c)
digest = lambda p: hashlib.sha256(open(p, "rb").read()).hexdigest()
baseline = {n: digest(os.path.join(d, n)) for n in files}
open(os.path.join(d, "config.ini"), "a").write("debug=true\n")       # simulated unauthorised change
for n in files:
    state = "UNCHANGED" if digest(os.path.join(d, n)) == baseline[n] else "MODIFIED -> ALERT"
    print(f"{n:12s} {state}")
```

Output

```
== Signature-based detection ==
ok                                       GET /index.html
ok                                       GET /search?q=students
ALERT SQL injection pattern              GET /search?q=' OR '1'='1
ALERT Path traversal pattern             GET /../../etc/passwd
ok                                       GET /about
== Host-based integrity check ==
config.ini   MODIFIED -> ALERT
app.py       UNCHANGED
```

Program: honeypot_lab.py

```
# Controlled low-interaction honeypot: listens on localhost only and logs connection attempts
import socket, threading, time
srv = socket.socket(); srv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
srv.bind(("127.0.0.1", 2222)); srv.listen(1)
def serve():
    conn, addr = srv.accept()
    conn.sendall(b"SSH-2.0-OpenSSH_7.4\r\n")                # fake banner, no real service
    data = conn.recv(64)
    print(f"[honeypot] connection from {addr[0]}  sent={data!r}")
    conn.close()
t = threading.Thread(target=serve); t.start(); time.sleep(0.2)
c = socket.create_connection(("127.0.0.1", 2222)); print("[client] got banner:", c.recv(64));
  ↪ c.sendall(b"root\r\n"); c.close()
t.join(); srv.close()
```

Output

```
[client] got banner: b'SSH-2.0-OpenSSH_7.4\r\n'
[honeypot] connection from 127.0.0.1  sent=b'root\r\n'
```

Output interpretation	Signatures flag the SQL-injection-like and path-traversal requests; the modified config file is flagged; the honeypot logs the connection.
Verification procedure	Confirm only malicious lines alert.
Common errors	Port already in use.
Troubleshooting	Change the port.
Security precautions	The honeypot listens only on localhost; never expose honeypots to a network without isolation and approval.
Practical conclusion	Detection needs rules, baselines and monitoring.
Student questions	1) What is a false positive? 2) Why hash files in HIDS? 3) Why isolate honeypots?

Module 15 Revision

1. Module summary

Network defence is layered: segmentation, firewalls (default deny), IDS/IPS (network and host) and monitoring. Honeypots provide early warning. Always combine prevention, detection and response.

2. Key definitions

Term	Definition
Firewall	Traffic filter.
IDS	Detects intrusion.
IPS	Prevents intrusion.
DMZ	Public server zone.
Honeypot	Decoy system.

3. Important terminology

Stateful inspection, signature, anomaly, SIEM, egress filtering.

4. Comparison table

Table 51. Comparison for Module 15

Type	Level	Strength
Packet filter	Network/transport	Fast
Stateful	Transport	Connection aware
Proxy	Application	Deep inspection

5. Quick revision notes

- Default deny.
- IDS alerts, IPS blocks.

6. Memory techniques

- IDS = Inspect and Detect; IPS = Inspect and Prevent.

7. Important formulas

- None; rule order and default deny are key.

8. Common examination mistakes

- Thinking firewalls stop all attacks.
- Ignoring outbound traffic.

Module 15 Question Bank

9. Short questions (15)

S1. List network security objectives.
S2. Define packet sniffing.
S3. Define spoofing.
S4. Define MITM.
S5. Define DoS.
S6. What is segmentation?
S7. What is a DMZ?
S8. What is a honeypot?
S9. What is a packet-filter firewall?
S10. What is stateful inspection?
S11. What is default deny?
S12. Define HIDS.
S13. Differentiate IDS and IPS.
S14. What is a false positive?
S15. What is a SIEM?

10. Long questions (8)

L1. Explain network attacks and defences.
L2. Explain firewall types, basing and location.
L3. Explain firewall configuration with rules.
L4. Explain stateful inspection.
L5. Explain HIDS and distributed HIDS.
L6. Compare IDS and IPS.
L7. Explain honeypots.
L8. Explain a layered network defence strategy.

11. Multiple-choice questions (10)

Q1. Default policy of a secure firewall:

A) Random
B) Deny
C) Allow
D) Log only

Q2. IPS differs from IDS by:

A) Blocking in-line
B) Detecting only
C) Using hashes
D) Being slower

Q3. A DMZ hosts:
A) Internal databases
B) Printers
C) Backups only
D) Public-facing servers

Q4. Stateful firewall tracks:
A) Colours
B) Connections
C) Passwords
D) Keys

Q5. HIDS monitors:
A) Cables
B) Whole Internet
C) One host
D) Printers

Q6. Honeypot purpose:
A) Backup
B) Speed
C) Storage
D) Decoy detection

Q7. Spoofing is:
A) Rebooting
B) Hashing
C) Forging identity
D) Compressing data

Q8. Wireshark is a:
A) Cipher
B) Firewall
C) Packet analyser
D) Database

Q9. False positive:
A) Harmless activity flagged
B) Encrypted data
C) Real attack flagged
D) Missed attack

Q10. Egress filtering controls:
A) Inbound only
B) Disk
C) Passwords
D) Outbound traffic

12. Scenario-based questions (5)

C1. A server shows 5000 SYN packets from one address. Action?
C2. Why would an IPS cause an outage?
C3. Design firewall rules for a web server.
C4. Where do you place a public web server?
C5. Why test restores of firewall configs?

13. Practical and problem-solving exercises (3)

X1. Write rules for a lab network.

X2. Capture HTTP and TLS and compare.

X3. Add three signatures to ids_lab.py.

Model answers (selected short questions)

S13. IDS detects and alerts passively; IPS sits in-line and can block traffic.

S15. A SIEM collects and correlates logs from many sources.

Detailed model answers (selected long questions)

L3. Default deny inbound; allow established/related; allow required services from specific sources; filter outbound; log drops; document and review rules; test from a second machine.

14. References

Scarfone, K., & Hoffman, P. (2009). *Guidelines on firewalls and firewall policy* (NIST SP 800-41 Rev. 1).

Scarfone, K., & Mell, P. (2007). *Guide to intrusion detection and prevention systems (IDPS)* (NIST SP 800-94).

Stallings, W., & Brown, L. (2018). *Computer security: Principles and practice* (4th ed.). Pearson.

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

Rescorla, E. (2018). *The Transport Layer Security (TLS) protocol version 1.3* (RFC 8446). https://www.rfc-editor.org/info/rfc8446

Module 16
Web Security, Application Attacks, Ethics and Projects

Introduction. Web applications are the most exposed software most organisations run. This final module explains how web applications work, why injection and cross-site scripting happen, how to prevent them, and the legal and ethical framework within which every security professional must act.

Why this topic matters. The same skills used to protect a system can be misused to attack one. Knowing the law and professional ethics is as important as technical skill.

Real-world relevance. University portals, online shops, banking, social media.

Learning objectives	Expected learning outcomes
• Describe web application architecture and common vulnerabilities. • Explain SQL injection, its types and prevention. • Explain XSS types and prevention including output encoding and CSP. • Explain legal issues, authorisation, responsible testing and research ethics. • Plan and present a security project.	• Demonstrate SQL injection and its fix in a local sandbox. • Demonstrate output encoding for XSS. • Assess a fictional university system.

Prerequisites	Estimated learning time
Modules 12 to 15 and basic SQL and HTML.	About 10 hours.

16.1 Web Security Fundamentals and Architecture

In simple words. A web application is a program that runs partly in your browser and partly on a server, talking through HTTP.

Formal definition. Web security is the protection of web applications, their users and their data from attacks that exploit flaws in code, configuration or design. Typical architecture: browser (client), web server, application logic, database, and supporting services.

Everyday analogy. A restaurant: the customer (browser) orders from the waiter (web server), who passes it to the kitchen (application) which uses the storeroom (database). Hostile orders can poison any step.

How it works. Every piece of data from the browser (forms, URLs, headers, cookies) is untrusted input. Security depends on validating input, encoding output, authenticating sessions, authorising each request and keeping components patched. The OWASP Top 10 lists the most critical risk categories (broken access control, cryptographic failures, injection, insecure design, security misconfiguration and others).

Term	Meaning
HTTP/HTTPS	Web protocol and its encrypted form.
Cookie	Small data stored by the browser, often holding a session ID.
Same-origin policy	Browser rule isolating content of different sites.
OWASP Top 10	Awareness list of critical web risks.

Practical applications. Any website or API.

> **Common misconception**
> Client-side checks are enough.

Security implications. Attackers can bypass the browser; the server must validate everything.

16.2 SQL Injection

In simple words. SQL injection tricks a website into running attacker-supplied SQL by mixing input into a query as text.

Formal definition. SQL injection is a vulnerability in which untrusted input is incorporated into a SQL statement without separation of code and data, allowing the attacker to alter the query.

Everyday analogy. A form letter says "Pay [NAME] 100 rupees". Someone writes their name as "Ali 100 rupees. Also pay Bob 1,000,000": the instruction is now different.

How it works. If the server builds `SELECT ... WHERE user='" + name + "' AND pw='" + pw + "'`, input such as `' OR '1'='1` makes the condition always true. Variants: in-band (error-based, union-based), blind (boolean or time-based) and out-of-band.

Figure 31 | SQL injection and its prevention

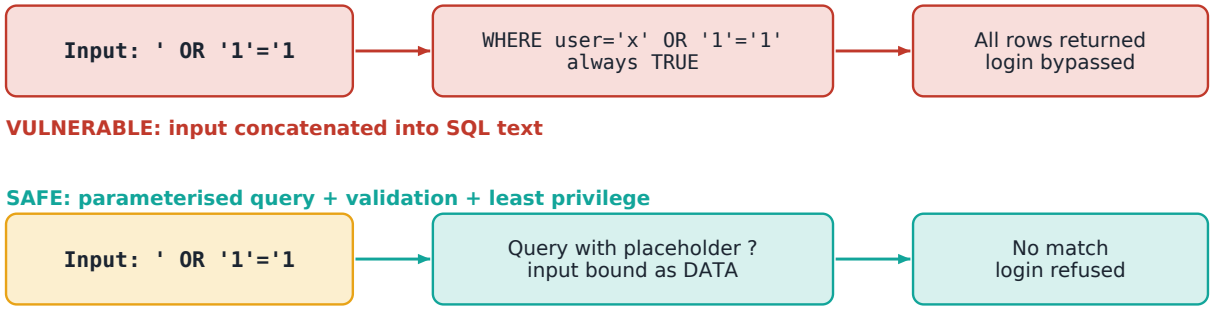


Figure 31. *SQL injection and its prevention*

Term	Meaning
Parameterised query	Query with placeholders; values sent separately as data.
Prepared statement	Pre-compiled query using parameters.
ORM	Object-relational mapper, normally parameterised.
Allow-list validation	Accept only values matching expected patterns.

Practical applications. Historic breaches of retail, telecom and government databases.

> **Common misconception**
> Escaping quotes manually is enough.

Security implications. Parameterised queries are the primary defence; manual escaping is error-prone.

Prevention checklist

- Use parameterised queries or prepared statements everywhere, including stored procedures.
- Validate input by type, length, format and allowed values (allow-list).

- Run the application with a least-privilege database account (no DROP or admin rights).
- Hide detailed database errors from users; log them securely.
- Use a web application firewall as an additional layer, not the only layer.
- Test with automated and manual testing in authorised environments.

16.3 Cross-Site Scripting (XSS)

In simple words. XSS makes a victim's browser run attacker-controlled script because a website displayed untrusted input as code.

Formal definition. Cross-site scripting is a vulnerability in which an application includes untrusted data in a web page without proper validation or encoding, allowing script execution in other users' browsers.

Everyday analogy. A notice board where anyone can pin notes. A hostile note with instructions that the board reader obeys, because the board does not mark notes as "just text".

How it works. Reflected: input in the request is immediately echoed in the response (for example in a search results page). **Stored**: malicious input is saved (comment, profile) and served to every visitor. **DOM-based**: client-side script writes untrusted data into the page (innerHTML). Impact: session theft, page defacement, actions in the victim's name.

Figure 32 | XSS and its prevention

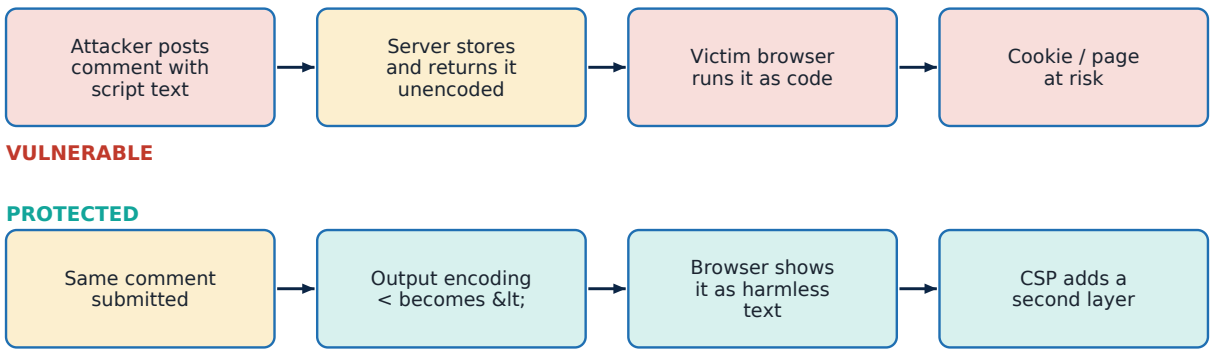


Figure 32. *XSS and its prevention*

Term	Meaning
Output encoding	Converting special characters (< > & " ') to harmless entities for the context.
Content Security Policy (CSP)	HTTP header restricting where scripts may load from.
HttpOnly cookie	Cookie unreadable by JavaScript.
Sanitisation	Removing dangerous markup using a trusted library.

Practical applications. Comment sections, search pages, profile fields.

Common misconception
Filtering the word "script" prevents XSS.

Security implications. Blacklists are easily bypassed; use context-aware output encoding.

XSS prevention

- Encode output for its context (HTML body, attribute, JavaScript, URL).
- Use safe APIs: textContent instead of innerHTML; template engines that auto-escape.

- Add a Content Security Policy such as `Content-Security-Policy: default-src 'self'; script-src 'self'; object-src 'none'`.
- Set HttpOnly, Secure and SameSite on cookies.
- Validate input and sanitise rich text with a maintained library.

16.4 Legal Issues, Ethics and Professional Conduct

Authorisation first. Testing or accessing a system without explicit written permission is illegal in most countries and can lead to prosecution, even when the intention is good. Laws vary; examples include the Prevention of Electronic Crimes Act (PECA) 2016 in Pakistan, the Computer Fraud and Abuse Act in the United States, the Computer Misuse Act 1990 in the United Kingdom and data-protection laws such as the GDPR in the European Union. This book is educational and not legal advice; consult current local law and your institution's policies.

Table 52. Principles of professional conduct

Principle	Meaning in practice
Authorisation	Written scope and permission before any testing; test only owned or authorised systems.
Minimise harm	Avoid disruption, do not access more data than necessary, protect any data seen.
Confidentiality	Handle findings and data as sensitive; share only with the owner.
Responsible disclosure	Report vulnerabilities privately to the owner or a coordinator, give time to fix, then publish if appropriate.
Honesty and integrity	Report accurately; do not exaggerate or hide errors.
Competence and learning	Stay current; recognise the limits of your skill.
Professional codes	Follow codes such as ACM and (ISC)2 codes of ethics.

Security research ethics. Use consent, minimal data, anonymisation, institutional review where human subjects or personal data are involved, and coordinated disclosure for vulnerabilities. Never publish working attack tools aimed at causing harm.

Case Study: Ethical dilemma: a student finds a flaw in the portal

Background	A student notices that changing an ID number in a URL displays another student's record.
Organisation and environment	University student portal.
Information assets	Student records.
Security problem	Broken access control (insecure direct object reference).
Threats and vulnerabilities	Threat: any student or outsider. Vulnerability: missing authorisation check.
Potential impact	Privacy breach for thousands of students.
Detection	The student realises the problem by accident.
Investigation	The student stops, takes no further data and does not share it.
Response	Reports privately to the IT help desk and security contact with minimal details.
Prevention	Add server-side authorisation checks and a published responsible-disclosure channel.
Lessons learned	Accidental discovery plus restraint and prompt reporting is the ethical path. Fictional scenario.

Practical 30: SQL Injection Demonstration and Prevention

Learning objective	Demonstrate injection and parameterised queries in a local sandbox.
Software and tools	Python 3 (sqlite3 is built in). Optional: OWASP Juice Shop or DVWA locally.
Environment	In-memory database on your own computer. For larger practice run a deliberately vulnerable app in a VM or container bound to localhost, for example: docker run --rm -p 127.0.0.1:3000:3000 bkimminich/juice-shop (not executed in the authoring environment).
Background	Seeing the failure makes the fix memorable.

Step-by-step procedure

1. Save sqli_lab.py.
2. Run it.
3. Observe the vulnerable and fixed result for the same input.
4. Add a validation rule for the username.

Program: sqli_lab.py

```
# Local SQL injection demo on an in-memory SQLite database (your own sandbox only)
import sqlite3
db = sqlite3.connect(":memory:")
db.execute("CREATE TABLE users(name TEXT, pw TEXT)")
db.execute("INSERT INTO users VALUES('ali','s3cret'),('sara','hunter2')")
def login_bad(name, pw):      # VULNERABLE: string concatenation
    q = f"SELECT name FROM users WHERE name='{name}' AND pw='{pw}'"
    return q, db.execute(q).fetchall()
def login_good(name, pw):     # FIXED: parameterised query
    return db.execute("SELECT name FROM users WHERE name=? AND pw=?", (name, pw)).fetchall()
attack = "' OR '1'='1"
print("Vulnerable query:", login_bad("ali", attack))
print("Parameterised   :", login_good("ali", attack))
print("Legitimate login:", login_good("ali", "s3cret"))
```

Output

```
Vulnerable query: ("SELECT name FROM users WHERE name='ali' AND pw='' OR '1'='1'", [('ali',),
  ↪ ('sara',)])
Parameterised   : []
Legitimate login: [('ali',)]
```

Output interpretation	The injected input returns a row from the vulnerable function and nothing from the parameterised function; legitimate login still works.
Verification procedure	Print the final SQL string in the vulnerable function.
Common errors	Syntax error from unbalanced quotes.
Troubleshooting	Examine the query string.
Security precautions	Practise only on your own sandbox or an authorised training application.
Practical conclusion	Separate code from data.
Student questions	1) Why does the vulnerable query succeed? 2) Why does the parameterised query fail the attack? 3) What else limits damage?

Practical 31: XSS Prevention with Output Encoding

Learning objective	Show how encoding turns hostile input into harmless text.
Software and tools	Python 3.
Environment	Terminal; nothing is executed in a browser.
Background	Encoding must match the output context.

Step-by-step procedure

1. Save xss_lab.py.
2. Run it.
3. Compare unsafe and safe fragments.
4. Optional: set a CSP header on a local test page.

Program: xss_lab.py

```
# Output encoding stops stored/reflected XSS (rendering shown as text, nothing is executed)
import html
comment = '<script>alert("XSS")</script>Nice post!'
print("UNSAFE page fragment:", f"<p>{comment}</p>")
print("SAFE   page fragment:", f"<p>{html.escape(comment)}</p>")
```

Output

```
UNSAFE page fragment: <p><script>alert("XSS")</script>Nice post!</p>
SAFE   page fragment: <p>&lt;script&gt;alert(&quot;XSS&quot;)&lt;/script&gt;Nice post!</p>
```

Output interpretation	In the safe version angle brackets become the entities < and >, so the browser displays text.
Verification procedure	View the output in a text file opened in a browser (the safe version shows literally).
Common errors	Encoding applied twice.
Troubleshooting	Encode once, at output.
Security precautions	Use harmless test strings on your own pages only.
Practical conclusion	Output encoding is the primary XSS defence.
Student questions	1) Why must encoding depend on context? 2) What does CSP add? 3) Why HttpOnly cookies?

Practical 32: Security Assessment of a Fictional University System

Learning objective	Perform a structured assessment and risk report for a fictional system.
Software and tools	Template below.
Environment	Paper exercise using the scenario.
Background	Combines the whole course.

Step-by-step procedure

1. Read the scenario: "Lahore Valley University" runs a web portal (login, fees, results), a student database server, Wi-Fi, a mail server and lab PCs. Passwords are stored as MD5; the portal uses HTTP for the results page; the database user has full rights; there is no MFA; backups are never restored in tests; firewall rules allow any-any inside the LAN; servers are patched irregularly.
2. List assets and classify them (Module 1).
3. List threats and vulnerabilities for each area.
4. Score risk using L x I (Module 8).
5. Recommend controls by module (cryptography, authentication, database, OS, network, web).
6. Prepare a one-page executive summary.

Table 53. Model assessment findings

Finding	Area	Risk (L x I)	Recommended control
MD5 password hashes	Authentication (M12)	5 x 5 = 25	Argon2/scrypt with salt, force reset, MFA
Results page over HTTP	Cryptography (M9-11)	4 x 4 = 16	TLS 1.2+ with valid certificate, HSTS
Database user has full rights	Database (M14)	4 x 5 = 20	Least-privilege accounts, REVOKE excess rights
No MFA	Authentication (M12)	4 x 4 = 16	TOTP or security keys for staff first
Backups never restore-tested	Availability (M8, M14)	3 x 5 = 15	Quarterly restore tests, offline copy
Any-any internal firewall rules	Network (M15)	4 x 4 = 16	Segmentation, default deny, logging
Irregular patching	OS (M14)	4 x 4 = 16	Patch policy with SLAs, vulnerability scanning

Output interpretation	The highest risks are weak password storage and excessive database privilege.
Verification procedure	Check that every recommendation maps to a specific finding.
Common errors	Listing only technical fixes.
Troubleshooting	Include people and process controls.
Security precautions	Fictional scenario; no real system is tested.
Practical conclusion	A good assessment ranks, justifies and recommends.
Student questions	1) Which three fixes come first and why? 2) What residual risk remains? 3) How would you verify each fix?

16.5 Student Research Presentations and Security Project

- **Project:** choose a topic (for example secure student portal design, phishing awareness, encrypted file vault, log-analysis tool, honeypot study in an isolated VM, risk assessment of a small business).
- **Deliverables:** 8 to 12 minute presentation, a 6 to 10 page report and, where relevant, working code with a README.
- **Report structure:** problem, background, method, implementation or analysis, results, security discussion, ethics and legal statement, references (APA 7).
- **Assessment guidance:** technical correctness, clarity, evidence, ethics, originality and teamwork. Weightings are decided by the instructor and are not specified in this book.
- **Rules:** test only owned or authorised systems; cite sources; no plagiarism; no real personal data.

Module 16 Revision

1. Module summary

Web security requires treating all input as untrusted. Prevent SQL injection with parameterised queries, validation and least privilege. Prevent XSS with output encoding, safe APIs and CSP. Always act with authorisation, ethics and professionalism.

2. Key definitions

Term	Definition
SQL injection	Untrusted input altering SQL.
XSS	Untrusted input executed as script in a browser.
CSP	Header limiting script sources.
Responsible disclosure	Private reporting of flaws.

3. Important terminology

Reflected, stored, DOM XSS, parameterised query, OWASP Top 10.

4. Comparison table

Table 54. Comparison for Module 16

	SQLi	XSS
Target	Database	Users' browsers
Cause	Code mixed with data in SQL	Untrusted data in HTML/JS
Fix	Parameterised queries	Output encoding, CSP

5. Quick revision notes

- Never trust input.
- Authorisation before testing.

6. Memory techniques

- SQL: "Parameters, Not Pasting". XSS: "Encode On Output".

7. Important formulas

- None.

8. Common examination mistakes

- Testing without permission.
- Using blacklists as the only defence.

Module 16 Question Bank

9. Short questions (15)

S1. Describe web application architecture.
S2. What is the same-origin policy?
S3. Define SQL injection.
S4. Name two types of SQL injection.
S5. What is a parameterised query?
S6. Why least privilege for DB accounts?
S7. Define XSS.
S8. Differentiate reflected and stored XSS.
S9. What is DOM-based XSS?
S10. What is output encoding?
S11. What is CSP?
S12. Why HttpOnly cookies?
S13. What is responsible disclosure?
S14. Why is authorisation required for testing?
S15. Name two security-research ethics principles.

10. Long questions (8)

L1. Explain SQL injection and how it works.

L2. Explain SQL injection prevention.

L3. Explain XSS types.

L4. Explain XSS prevention.

L5. Explain the OWASP Top 10 relevance.

L6. Discuss legal issues in computer security.

L7. Explain professional and research ethics.

L8. Describe a security assessment of a web portal.

11. Multiple-choice questions (10)

Q1. Primary defence against SQL injection:

A) Hiding the site

B) Colour themes

C) Longer URLs

D) Parameterised queries

Q2. Stored XSS is:

A) Saved and served to others

B) Only in URLs

C) A DoS

D) A hash

Q3. Output encoding turns the character < into:

A) %%

B) <

C) <<

D) !

Q4. CSP is:

A) A cipher

B) A database

C) A password

D) A header restricting content sources

Q5. Testing without permission is:

A) Often illegal

B) Encouraged

C) Required

D) Always legal

Q6. Which is a blind SQLi variant?

A) Padding

B) Rail

C) Time-based

D) Hill

Q7. DOM XSS arises from:

A) Server-only code

B) Client-side script writing untrusted data

C) Hash

D) Firewall

Q8. Responsible disclosure means:

A) Posting publicly at once

B) Ignoring it

C) Reporting privately to the owner

D) Selling the flaw

Q9. Which cookie flag stops JavaScript reading a cookie?

A) Secure

B) Path

C) SameSite

D) HttpOnly

Q10. Client-side validation alone is:

A) Insufficient

B) Illegal

C) Sufficient

D) Required only

12. Scenario-based questions (5)

C1. A search page echoes input. How could XSS occur and how do you fix it?

C2. Login fails when a name has an apostrophe. What does it suggest?

C3. A student finds a flaw. What should they do?

C4. Would you use a WAF alone to fix SQLi?

C5. A developer uses innerHTML for comments. Advise.

13. Practical and problem-solving exercises (3)

X1. Run the SQLi lab and add a validation rule.

X2. Write a CSP for a simple site.

X3. Draft a one-page responsible-disclosure policy for a university.

Model answers (selected short questions)

S3. SQL injection occurs when untrusted input changes the meaning of a SQL statement.

S5. A query with placeholders where input values are sent as data, not as SQL code.

Detailed model answers (selected long questions)

L2. Parameterised queries; allow-list validation; least-privilege database accounts; hidden error details; WAF as a secondary layer; testing in authorised environments.

14. References

OWASP Foundation. (2021). *OWASP Top 10:2021*. https://owasp.org/Top10/

OWASP Foundation. (n.d.). *OWASP Cheat Sheet Series*. https://cheatsheetseries.owasp.org/

Stallings, W., & Brown, L. (2018). *Computer security: Principles and practice* (4th ed.). Pearson.

Easttom, W. (2019). *Computer security fundamentals* (4th ed.). Pearson.

Anderson, R. (2020). *Security engineering: A guide to building dependable distributed systems* (3rd ed.). Wiley.

Back Matter A: Consolidated Practical Laboratory Index

All 32 required laboratory exercises plus four additional exercises (33 to 36) are listed in numerical order with their modules. Programs are in the companion lab archive.

Table 60. Practical laboratory index

No.	Practical	Module	Type
1	Identifying Information Assets	1	Worksheet / reference procedure
2	Analysing the CIA Triad	1	Worksheet / reference procedure
3	Recognising Phishing Emails	1	Worksheet / reference procedure
4	Identifying Malware Indicators	1	Worksheet / reference procedure
5	Classifying Security Attacks	2	Worksheet / reference procedure
6	Caesar Cipher Encryption and Decryption	4	Program output executed
7	General Substitution Cipher	4	Program output executed
8	Playfair Cipher	4	Program output executed
9	Vigenere and Autokey Ciphers	5	Program output executed
10	Rail Fence Cipher	5	Program output executed
11	Hill Cipher Calculations	6	Program output executed
12	Euclidean Algorithm	8	Program output executed
13	Modular Inverse	8	Program output executed
14	Educational RC4 Demonstration	7	Program output executed
15	DES Structure Simulation	7	Program output executed
16	AES Encryption and Decryption (AES-256-GCM)	9	Program output executed
17	RSA Key Generation, Encryption and Decryption	10	Program output executed
18	SHA-256 Hashing and File Integrity	11	Program output executed
19	Digital Signature Creation and Verification	11	Program output executed
20	TLS Certificate Inspection	11	Program output executed
21	Password Security: Salted Hashing	12	Program output executed
22	Authentication and MFA: TOTP	12	Program output executed
23	Safe Buffer Overflow Demonstration	13	Program output executed
24	Database GRANT and REVOKE	14	Worksheet / reference procedure
25	Operating System Permissions	14	Program output executed
26	Firewall Configuration	15	Worksheet / reference procedure
27	Wireshark Traffic Inspection	15	Worksheet / reference procedure
28	Security Log Analysis	15	Program output executed
29	Intrusion Detection Concepts and Honeypot Demonstration	15	Program output executed
30	SQL Injection Demonstration and Prevention	16	Program output executed
31	XSS Prevention with Output Encoding	16	Program output executed
32	Security Assessment of a Fictional University System	16	Worksheet / reference procedure
33	Alphabet Mapping and Modular Arithmetic Warm-up	3	Program output executed
34	Risk Register for a Fictional Organisation	8	Worksheet / reference procedure
35	Caesar Cipher Cryptanalysis by Frequency Analysis	6	Program output executed

No.	Practical	Module	Type
36	Inspecting Authentication Logs	12	Program output executed

Reference procedures (24 database GRANT/REVOKE, 26 firewall, 27 Wireshark, 20 for a live website) require tools that must be installed in your own lab; their expected results are given in the text.

Back Matter B: Important Command Reference

Table 61. Command quick reference

Task	Command	Notes
Check Python	python3 --version	Python 3.8+ needed (pow(e,-1,m))
Create virtual environment	python3 -m venv venv && source venv/bin/activate	Isolate libraries
Install library	python3 -m pip install cryptography	Use inside venv
SHA-256 of a file	sha256sum file	Linux; macOS: shasum -a 256 file
Compile C	gcc -Wall -o prog prog.c	Warnings are teaching signals
View permissions	ls -l	r4 w2 x1
Change permissions	chmod 640 file	Owner rw, group r
Show ACL	getfacl file	acl package
Inspect certificate	openssl x509 -in cert.pem -noout -text	Subject, issuer, validity, SAN
Test TLS server (authorised)	openssl s_client -connect host:443	Needs network
List firewall rules (nftables)	nft list ruleset	Root required
ufw status	ufw status verbose	Ubuntu
Capture filter example	tcp port 80	In Wireshark capture options
Auth log (Debian/Ubuntu)	journalctl -u ssh or /var/log/auth.log	Failed logins
ASLR status	cat /proc/sys/kernel/randomize_va_space	2 = full

Back Matter C: Cryptography Formula Sheet

Table 62. Formula sheet

Topic	Formula / rule
Caesar	C = (P + k) mod 26; P = (C - k) mod 26
General substitution key space	26! (about 2^{88})
Vigenere	$C_i = (P_i + K_{i \bmod m}) \bmod 26$
Autokey	Key stream = primer + plaintext (or ciphertext)
Rail Fence	Write zigzag on n rails; read rows
Hill	C = K P mod 26; P = K^{-1} C mod 26; K^{-1} = det^{-1} adj(K)
Hill validity	gcd(det K, 26) = 1
Stream cipher	C = P xor K; P = C xor K
Feistel	$L_i = R_{i-1}$; $R_i = L_{i-1}$ xor $f(R_{i-1}, K_i)$
Congruence	a = b (mod n) iff n divides (a - b)
Euclid	gcd(a, b) = gcd(b, a mod b)
Bezout	ax + by = gcd(a, b)
Inverse	a^{-1} mod n exists iff gcd(a, n) = 1
Square-and-multiply	Square each bit; multiply by a if bit = 1
Risk exposure	RE = P(loss) x Loss
RRL	$(RE_{before} - RE_{after})$ / cost
AES rounds	10 / 12 / 14 for 128 / 192 / 256-bit keys
RSA	n = pq; phi = (p-1)(q-1); ed = 1 mod phi; C = M^e mod n; M = C^d mod n
Signature	Sign: S = Sign(SK, H(m)); Verify: Verify(PK, m, S)
Password storage	hash = KDF(password, salt)
TOTP	HOTP(K, floor(T / 30))

Back Matter D: Important Algorithms and Pseudocode

Pseudocode summary

```
CAESAR-ENCRYPT(text, k):
  for each letter c in text:
      output chr( (index(c) + k) mod 26 )

PLAYFAIR-ENCRYPT(M, text):
  split text into digraphs; insert X between equal letters; pad with X
  for each pair (a, b):
      same row    -> letters to the right
      same column -> letters below
      otherwise   -> swap columns

EXTENDED-EUCLID(a, b):
  if b = 0: return (a, 1, 0)
  (g, x, y) = EXTENDED-EUCLID(b, a mod b)
  return (g, y, x - (a div b) * y)

MOD-EXP(a, e, n):
  r = 1
  for each bit of e from the left: r = r*r mod n; if bit = 1: r = r*a mod n
  return r

RC4-KSA(key): S = [0..255]; j = 0
  for i in 0..255: j = (j + S[i] + key[i mod len]) mod 256; swap(S[i], S[j])
RC4-PRGA: i = j = 0; loop: i++; j += S[i]; swap(S[i], S[j]); output S[(S[i]+S[j]) mod 256]

DES: IP; 16 x (L,R) = (R, L xor f(R,Ki)); swap; FP
AES-ROUND: SubBytes; ShiftRows; MixColumns (not in last round); AddRoundKey
```

Back Matter E: Security Terminology Glossary

Term	Meaning
Access control list (ACL)	List of permissions attached to an object.
Active attack	Attack that alters resources or disrupts operation.
AES	Advanced Encryption Standard; 128-bit block symmetric cipher.
AEAD	Authenticated Encryption with Associated Data.
Asset	Anything of value requiring protection.
Asymmetric cryptography	Cryptography with a public/private key pair.
Authentication	Verifying a claimed identity.
Authorisation	Granting permitted actions to an authenticated identity.
Availability	Authorised users can access systems when needed.
Backup	Copy of data kept for recovery.
Block cipher	Cipher operating on fixed-size blocks.
Brute force	Trying all possible keys.
Buffer overflow	Writing beyond the bounds of a buffer.
Caesar cipher	Shift cipher; C = (P + k) mod 26.
Certificate authority (CA)	Trusted issuer of digital certificates.
Ciphertext	Encrypted output.
CIA Triad	Confidentiality, Integrity, Availability.
Confidentiality	Preventing unauthorised disclosure.
Control	Safeguard that reduces risk.
Cryptanalysis	Analysis of ciphers to break them.
Cryptography	Science of secure communication.
Denial of Service (DoS)	Attack making a service unavailable.
DES	Data Encryption Standard; 64-bit block, 56-bit key; obsolete.
Digital signature	Private-key-created proof of origin and integrity.
DMZ	Network zone for public-facing servers.
Encryption / decryption	Converting plaintext to ciphertext and back.
Exploit	Method that uses a vulnerability.
Feistel network	Round structure used in DES.
Firewall	Device or software filtering network traffic.
Frequency analysis	Attack using letter statistics.
GCD	Greatest common divisor.
Hash function	One-way function producing a fixed-size digest.
HIDS / HIPS	Host-based intrusion detection / prevention.
Honeypot	Decoy system used to detect or study attackers.
IDS / IPS	Intrusion detection / prevention system.
Integrity	Data is accurate and unaltered.
Key	Secret parameter controlling a cipher.
KDF	Key derivation function.

Term	Meaning
Least privilege	Only the access needed for a task.
MAC (access control)	Mandatory Access Control.
MAC (cryptography)	Message authentication code.
Malware	Malicious software.
MFA	Multi-factor authentication.
Modular arithmetic	Arithmetic with wrap-around at a modulus.
Multiplicative inverse	x with ax = 1 mod n.
Non-repudiation	Preventing denial of an action.
Nonce	Number used once.
Passive attack	Attack that observes without altering.
Phishing	Fraudulent messages that trick users.
Plaintext	Original readable message.
PKI	Public Key Infrastructure.
Privilege	Right to perform an action.
RBAC	Role-Based Access Control.
RC4	Stream cipher; broken.
Replay attack	Re-sending a captured valid message.
Risk	Likelihood x impact of loss.
RSA	Public-key algorithm based on factoring.
Salt	Random value added before hashing.
Session	State of an authenticated interaction.
SHA-256	Member of the SHA-2 hash family.
SIEM	Security information and event management.
Social engineering	Manipulating people to breach security.
SQL injection	Untrusted input altering a SQL query.
Stream cipher	Cipher combining data with a key stream.
Symmetric cryptography	Same key for encryption and decryption.
Threat	Potential cause of harm.
TLS	Transport Layer Security.
TOTP	Time-based one-time password.
Traffic analysis	Studying communication patterns.
Transposition	Rearranging positions of symbols.
Virus	Malware that attaches to files and replicates.
Vulnerability	Weakness that can be exploited.
Worm	Self-spreading malware.
XSS	Cross-site scripting.

Back Matter F: Comprehensive Revision Guide

Module-wise revision sheets appear at the end of each module. This guide collects the most frequently confused ideas.

Table 63. Frequently confused concepts

Often confused	Difference
Threat vs vulnerability	Threat is a source of harm; vulnerability is a weakness.
Virus vs worm vs Trojan	Needs host file; self-spreads; disguised.
Passive vs active attack	Observe vs alter or disrupt.
Service vs mechanism	What protection is provided vs how it is implemented.
Substitution vs transposition	Change letters vs move letters.
Encryption vs hashing	Reversible with key vs one-way.
Symmetric vs asymmetric	One shared key vs key pair.
Stream vs block cipher	Bits/bytes with key stream vs fixed blocks.
DES vs AES	64-bit block, 56-bit key, Feistel vs 128-bit block, 128/192/256-bit key, SPN.
Signature vs encryption	Proves origin and integrity vs hides content.
Authentication vs authorisation	Who you are vs what you may do.
DAC vs MAC vs RBAC	Owner vs system policy vs roles.
IDS vs IPS	Alert vs block.
Reflected vs stored XSS	Immediate echo vs saved and served.
Risk exposure vs RRL	Expected loss vs benefit per cost.

Back Matter G: Midterm Preparation

The calendar places the midterm in Week 9 after revision in Week 8. The topics below are the natural coverage (Weeks 1 to 7, Modules 1 to 8). The format, marks and weightings are set by the instructor and are not specified here.

Table 64. Midterm coverage

Module	Priority topics
1	CIA, terminology, social engineering, malware and virus lifecycle
2	Passive/active attacks, services, mechanisms
3	Cryptography terms, symmetric vs asymmetric, substitution vs transposition
4	Caesar, substitution key space, frequency analysis, Playfair
5	Vigenere, autokey, Rail Fence
6	Hill cipher, invertibility, cryptanalysis
7	RC4 (KSA/PRGA), DES structure and weaknesses
8	Modular arithmetic, Euclid, inverse, risk exposure and RRL

Midterm practice paper

- Short: define threat, vulnerability and risk; differentiate passive and active attacks; list the six security services.
- Calculation: encrypt "SAFE ZONE" with Caesar k = 7, then with Vigenere key KEY (full working).
- Calculation: build a Playfair matrix for key CIPHER and encrypt "MEETING".
- Calculation: find the inverse of 11 modulo 26 and decrypt a Hill pair with key [[3,3],[2,5]].
- Long: explain the DES round structure with a diagram, and state why DES is no longer used.
- Scenario: compute RE and RRL for a loss of PKR 3,000,000 with P = 0.15, reduced to 0.03 by a PKR 150,000 control.

Back Matter H: Final Examination Preparation

Final coverage follows the whole syllabus with emphasis on Modules 9 to 16 (Weeks 10 to 16). Format and weightings follow the instructor's announcement.

Table 65. Final examination emphasis

Module	Priority topics
9	AES rounds, modes, GCM, nonce rules
10	RSA key generation, encryption, padding, key sizes
11	Hash properties, signatures, PKI chain, revocation
12	Password storage, MFA, sessions
13	Buffer overflow and defences
14	GRANT/REVOKE, least privilege, DAC/MAC/ACL/RBAC, hardening
15	Firewalls, IDS/IPS, HIDS, honeypots, segmentation
16	SQL injection, XSS, ethics and law

Practical examination preparation

- Be able to run and explain each lab program and interpret its output.
- Practise by-hand calculations: Caesar, Playfair, Vigenere, Rail Fence, Hill, Euclid, inverse, RSA (small primes).
- Be able to write a GRANT and REVOKE script, a default-deny firewall policy and a parameterised query from memory.
- Be able to inspect a certificate and explain the chain.
- Bring a clear understanding of authorised, ethical testing.

Comprehensive practice tests

Table 66. Suggested practice tests

Test	Content
Test 1 (cryptography)	10 MCQ, 5 short, 3 calculations from Modules 3 to 11
Test 2 (systems and networks)	10 MCQ, 5 short, 2 scenarios from Modules 12 to 16
Test 3 (whole course)	Use the comprehensive MCQ bank below plus two long questions of your choice

Back Matter I: Comprehensive MCQ Bank

1. A hospital loses access to records because of ransomware. Which CIA goal is mainly affected?
A) Integrity
B) Authenticity
C) Confidentiality
D) Availability

2. Which is a transposition cipher?
A) Rail Fence
B) Playfair
C) Vigenere
D) Caesar

3. The key space of the Caesar cipher is:
A) 25
B) 26!
C) 2^56
D) 2^128

4. A Hill matrix with gcd(det,26)=1 is:
A) Prime
B) Symmetric
C) Invertible
D) Singular

5. Which algorithm is withdrawn by NIST?
A) SHA-256
B) RSA-3072
C) AES
D) DES

6. The effective DES key length is:
A) 128 bits
B) 64 bits
C) 56 bits
D) 40 bits

7. AES-128 uses how many rounds?
A) 16
B) 12
C) 10
D) 14

8. Never reuse a nonce in:
A) AES-GCM
B) RSA public keys
C) SHA-256
D) Caesar

9. 7 inverse modulo 26 is:
A) 11
B) 15
C) 19
D) 5

10. RSA decryption uses:
A) Nonce
B) Public exponent
C) Private exponent
D) Salt

11. Digital signatures are created with the:
A) IV
B) Private key
C) Public key
D) Salt

12. A hash function is:
A) Compression only
B) Reversible
C) One-way
D) Keyed encryption

13. Salted slow hashing protects against:
A) Precomputed tables and fast guessing
B) Sniffing
C) XSS
D) DoS

14. Which factor is a fingerprint?
A) Location
B) Possession
C) Knowledge
D) Inherence

15. A stack canary detects:
A) DDoS
B) Phishing
C) SQL injection
D) Stack overwrite before return

16. Which command removes a SQL privilege?
A) REVOKE
B) GRANT
C) DROP USER only
D) COMMIT

17. RBAC attaches permissions to:
A) Roles
B) Packets
C) Hashes
D) Keys

18. A default-deny firewall:
A) Allows unless blocked
B) Encrypts traffic
C) Logs only
D) Blocks unless allowed

19. An IPS differs from an IDS because it:
A) Is a host only
B) Uses no signatures
C) Blocks in-line
D) Only logs

20. XSS is best prevented by:
A) Bigger keys
B) Output encoding
C) Longer passwords
D) Stronger Wi-Fi

21. SQL injection is best prevented by:
A) Hiding the login page
B) HTTP
C) Parameterised queries
D) Frequency analysis

22. Responsible disclosure means reporting:
A) Privately to the owner
B) Publicly at once
C) To attackers
D) Never

23. RE equals:
A) Probability x loss
B) Loss - probability
C) Loss only
D) Probability / loss

24. A replay attack is countered by:
A) Nonces or timestamps
B) Bigger fonts
C) Frequency analysis
D) Tape backups

25. A honeypot should be:
A) Hidden from admins
B) Isolated and monitored
C) Connected to the internal LAN
D) Used for email

26. Traffic analysis is a:
A) Hash
B) Passive attack
C) Active attack
D) Virus

27. TLS certificates are issued by a:

A) Router

B) DNS server

C) Database

D) Certificate authority

28. Which mode gives encryption plus integrity?

A) CBC alone

B) GCM

C) ECB

D) CTR alone

29. Which is NOT a CIA goal?

A) Popularity

B) Integrity

C) Confidentiality

D) Availability

30. Least privilege means:

A) Minimum necessary access

B) Maximum access

C) No logging

D) Shared accounts

Answer key for the comprehensive MCQ bank

1-D 2-A 3-A 4-C 5-D 6-C 7-C 8-A 9-B 10-C 11-B 12-C 13-A 14-D 15-D 16-A 17-A 18-D 19-C 20-B 21-C 22-A 23-A 24-A 25-B 26-B 27-D 28-B 29-A 30-A

Back Matter J: Objective Answer Key and Explanations (Modules 1 to 16)

Each module's ten multiple-choice questions are keyed below with a short explanation. For numerical questions the full working is in the module text or the model answers.

Module 1

Q	Ans	Explanation
1	D	Unauthorised disclosure breaks confidentiality.
2	B	Weakness = vulnerability.
3	A	Worms are standalone.
4	C	Standard definition.
5	C	Physical social engineering.
6	B	Backups restore access.
7	D	It targets people.
8	C	Payload executes last.
9	A	Popularity is not a goal.
10	B	Limits damage from compromise.

Module 2

Q	Ans	Explanation
1	A	Gaining unauthorised access is interception.
2	A	Freshness data stops reuse.
3	B	Audit trails serve all services.
4	C	Signatures bind a sender to a message.
5	D	Observation only.
6	D	Service becomes unusable.
7	B	Counterfeit objects are fabricated.
8	C	Labels classify resources.
9	D	No change to the system.
10	C	Mechanism implements a service.

Module 3

Q	Ans	Explanation
1	C	Security rests on the key.
2	B	A0 B1 ... K10.
3	A	It substitutes letters.
4	B	Uses a key pair.
5	C	-3+26=23.
6	A	Rearranges positions.
7	C	Analysing ciphers to break them.
8	D	Efficient for bulk data.
9	D	Encryption output.
10	B	Public key shares session key.

Module 4

Q	Ans	Explanation
1	B	Shifts 1 to 25.
2	D	D(3)+3=6=G.
3	A	Permutations of 26.
4	A	Statistics survive monoalphabetic encryption.
5	C	Pairs.
6	C	Letter to the right.
7	D	23+5=28 mod 26=2=C.
8	D	13+13=26.
9	A	E.
10	B	Double letters need a filler.

Module 5

Q	Ans	Explanation
1	B	Each key position is a Caesar shift.
2	B	0+11=11=L.
3	A	Positions rearranged.
4	D	Message extends the key.
5	B	Repeated patterns reveal key length.
6	D	Rails count.
7	C	Only positions change.
8	A	Hides letter statistics.
9	C	The primer starts the stream.
10	A	Obsolete.

Module 6

Q	Ans	Explanation
1	D	Needed for decryption.
2	C	15-6=9.
3	B	27 = 1 mod 26.
4	A	By definition.
5	C	Only 25.
6	A	m letters at once.
7	B	No inverse mod 26.
8	C	English E.
9	D	Also weak mathematically.
10	D	Only ciphertext available.

Module 7

Q	Ans	Explanation
1	D	DES uses 16 rounds.
2	B	8 of 64 are parity.
3	D	Key-stream generator.
4	C	XOR cancels the key stream.
5	D	S-boxes.
6	A	64 bits.
7	B	RC4 cipher suites.
8	C	Reverse order.
9	A	Modern authenticated cipher.
10	C	32 to 48 bits.

Module 8

Q	Ans	Explanation
1	D	Euclid trace.
2	A	7x15=105=1 mod 26.
3	C	Coprime.
4	C	Probability times loss.
5	C	Shares the risk.
6	B	135=5x26+5.
7	A	Square-and-multiply gives 21.
8	D	Saves more than costs.
9	B	After controls.
10	B	-7+26=19.

Module 9

Q	Ans	Explanation
1	D	128 bits.
2	D	14.
3	C	S-box.
4	B	ECB leaks patterns.
5	A	AEAD.
6	A	Nonce reuse breaks GCM.
7	C	SPN.
8	B	Protect keys.
9	B	Finite field.
10	D	FIPS 197.

Module 10

Q	Ans	Explanation
1	C	60 x 52.
2	A	Private.
3	A	Factoring hardness.
4	D	7x3=21=1 mod 20.
5	B	2048 bits.
6	C	OAEP.
7	D	Sign with private.
8	B	Slow; use hybrid.
9	A	Same input same output.
10	D	Correct decryption.

Module 11

Q	Ans	Explanation
1	B	256 bits.
2	B	Authentication, integrity, non-repudiation.
3	D	Private.
4	C	MD5.
5	A	Issues certificates.
6	D	Definition.
7	D	It does not guarantee honesty.
8	C	CRL.
9	C	One-way.
10	A	Trusted root.

Module 12

Q	Ans	Explanation
1	D	Precomputation.
2	B	Identity.
3	D	Inherence.
4	A	Slow salted KDF.
5	B	RFC 6238.
6	A	Mitigates cookie theft via XSS.
7	A	SIM swapping.
8	D	Reused passwords.
9	C	After identity.
10	C	Side channel.

Module 13

Q	Ans	Explanation
1	B	No length check.
2	B	Detects stack smashing.
3	C	No code execution from data.
4	B	Randomises layout.
5	D	Unchecked write.
6	A	Correct code.
7	D	Random input.
8	C	Rust.
9	A	Stacks typically grow toward lower addresses.
10	C	Reduces impact.

Module 14

Q	Ans	Explanation
1	D	Removes privileges.
2	C	5 = r-x.
3	B	Central policy.
4	B	Minimum needed.
5	D	Roles.
6	C	Restricts exposed data.
7	B	Restore testing.
8	C	Per-object list.
9	A	r4 w2 x1.
10	D	Time to recover.

Module 15

Q	Ans	Explanation
1	B	Default deny.
2	A	Prevention.
3	D	Exposed services.
4	B	Connection state.
5	C	Host-level.
6	D	Decoy.
7	C	False identity.
8	C	Packet capture and analysis.
9	A	Wrong alert.
10	D	Outbound.

Module 16

Q	Ans	Explanation
1	D	Separates code and data.
2	A	Persistent.
3	B	The HTML entity < is displayed as text.
4	D	Content Security Policy.
5	A	Authorisation is essential.
6	C	Time delays reveal data.
7	B	Client-side sink.
8	C	Private reporting.
9	D	HttpOnly.
10	A	Attackers bypass the browser.

Back Matter K: Project and Presentation Guidelines

- Choose a topic that can be completed ethically on systems you own.
- Structure: title, problem, background, method, results, security discussion, ethics and legal statement, references.
- Presentation: 8 to 12 minutes, clear diagrams, live or recorded demonstration in a safe lab, 3 minutes of questions.
- Evaluation criteria (weightings set by the instructor): technical accuracy, clarity, evidence, ethics, originality, teamwork, references.
- Submit code with a README that lists tools, versions, commands, expected output and safety notes.

Back Matter L: Academic Integrity Statement

Learning information security requires honest work. Submit only your own work, acknowledge sources using APA 7th edition, never copy code or reports without citation, and never fabricate data, results or references. Collaboration is allowed only as permitted by your instructor. Misusing security knowledge, including testing systems without permission, is both an academic offence and, in most jurisdictions, a crime.

Back Matter M: Further Learning Resources

- Official documentation: Python (docs.python.org), the cryptography library (cryptography.io), OpenSSL manual pages.
- Standards and guidance: NIST publications (FIPS 197, 180-4, SP 800-30, 800-61, 800-63B), IETF RFCs (5280, 6238, 8017, 8446).
- OWASP Top 10, OWASP Cheat Sheet Series and OWASP Juice Shop/WebGoat for legal practice.
- Practice platforms that provide authorised targets: TryHackMe, Hack The Box, CyberDefenders (check terms).
- Certification paths: CompTIA Security+, Google Cybersecurity Certificate, ISC2 Certified in Cybersecurity, Cisco CyberOps.
- Tools to explore: Wireshark, nmap (own systems only), Kali or Parrot in isolated VMs, Security Onion, Wazuh.

Back Matter N: References and Bibliography

References follow APA 7th edition. Editions and standards cited are those the author is confident exist; please verify publication details, URLs and availability against your library or the publisher before formal citation.

Anderson, R. (2020). *Security engineering: A guide to building dependable distributed systems* (3rd ed.). Wiley.

Easttom, W. (2019). *Computer security fundamentals* (4th ed.). Pearson.

Paar, C., & Pelzl, J. (2010). *Understanding cryptography: A textbook for students and practitioners*. Springer.

Pfleeger, C. P., Pfleeger, S. L., & Margulies, J. (2015). *Security in computing* (5th ed.). Prentice Hall.

Stallings, W., & Brown, L. (2018). *Computer security: Principles and practice* (4th ed.). Pearson.

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

National Institute of Standards and Technology. (2001). *Advanced Encryption Standard (AES)* (FIPS PUB 197). https://doi.org/10.6028/NIST.FIPS.197

National Institute of Standards and Technology. (2015). *Secure Hash Standard (SHS)* (FIPS PUB 180-4). https://doi.org/10.6028/NIST.FIPS.180-4

National Institute of Standards and Technology. (1999). *Data Encryption Standard (DES)* (FIPS PUB 46-3; withdrawn 2005).

National Institute of Standards and Technology. (2023). *Digital Signature Standard (DSS)* (FIPS PUB 186-5). https://doi.org/10.6028/NIST.FIPS.186-5

National Institute of Standards and Technology. (2012). *Guide for conducting risk assessments* (NIST SP 800-30 Rev. 1). https://doi.org/10.6028/NIST.SP.800-30r1

Dworkin, M. (2007). *Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC* (NIST SP 800-38D). https://doi.org/10.6028/NIST.SP.800-38D

Scarfone, K., & Hoffman, P. (2009). *Guidelines on firewalls and firewall policy* (NIST SP 800-41 Rev. 1).

Joint Task Force. (2020). *Security and privacy controls for information systems and organizations* (NIST SP 800-53 Rev. 5).

Cichonski, P., Millar, T., Grance, T., & Scarfone, K. (2012). *Computer security incident handling guide* (NIST SP 800-61 Rev. 2). https://doi.org/10.6028/NIST.SP.800-61r2

Grassi, P. A., et al. (2017). *Digital identity guidelines: Authentication and lifecycle management* (NIST SP 800-63B).

Kent, K., & Souppaya, M. (2006). *Guide to computer security log management* (NIST SP 800-92).

Scarfone, K., & Mell, P. (2007). *Guide to intrusion detection and prevention systems (IDPS)* (NIST SP 800-94).

Cooper, D., et al. (2008). *Internet X.509 public key infrastructure certificate and CRL profile* (RFC 5280). https://www.rfc-editor.org/info/rfc5280

M'Raihi, D., Machani, S., Pei, M., & Rydell, J. (2011). *TOTP: Time-based one-time password algorithm* (RFC 6238). https://www.rfc-editor.org/info/rfc6238

Popov, A. (2015). *Prohibiting RC4 cipher suites* (RFC 7465). https://www.rfc-editor.org/info/rfc7465

Moriarty, K., Kaliski, B., Jonsson, J., & Rusch, A. (2016). *PKCS #1: RSA cryptography specifications version 2.2* (RFC 8017). https://www.rfc-editor.org/info/rfc8017

Rescorla, E. (2018). *The Transport Layer Security (TLS) protocol version 1.3* (RFC 8446). https://www.rfc-editor.org/info/rfc8446

OWASP Foundation. (2021). *OWASP Top 10:2021*. https://owasp.org/Top10/

OWASP Foundation. (n.d.). *OWASP Cheat Sheet Series*. https://cheatsheetseries.owasp.org/

Python Software Foundation. (n.d.). *Python 3 documentation*. https://docs.python.org/3/

Python Cryptographic Authority. (n.d.). *cryptography documentation*. https://cryptography.io/

Final Message from the Author

You have now travelled from the meaning of confidentiality to the design of a secure web application. Along the way you have shifted letters, multiplied matrices, traced rounds of DES and AES, computed with RSA, hashed files, locked down databases and read firewall logs.

Remember that every technique in this book exists to protect people: their privacy, their money, their health and their trust. Learn it well, practise it ethically, and keep learning, because the threats change even when the principles do not.

Thank you for reading. Now go and build something secure.

Professor Muhammad Umar Naheed

Back Matter O: Subject Index

Page numbers refer to pages in this PDF (see page footers). Terms are listed in alphabetical order.

access control list	145, 180	multi-factor	30, 126, 128, 181
active attack	37, 38, 41, 42, 43, 180 ...	multiplicative inverse	91, 93, 98, 181
AES	48, 50, 51, 54, 55, 57 ...	non-repudiation	39, 40, 41, 42, 43, 44 ...
ASLR	135, 136, 137, 138, 139, 140 ...	nonce	40, 41, 43, 101, 103, 104 ...
asymmetric	46, 47, 48, 50, 51, 52 ...	OAEP	109, 111, 112, 113, 114, 192
audit trail	37, 39, 40, 42, 43, 189	OCSP	119, 122, 123
autokey	65, 66, 67, 68, 69, 70 ...	output encoding	164, 166, 168, 169, 170, 171 ...
availability	24, 26, 27, 28, 30, 31 ...	packet sniffing	38, 152, 153, 161
backup	25, 26, 27, 28, 29, 30 ...	parameterised query	165, 168, 171, 184
brute force	55, 57, 58, 59, 61, 63 ...	passive attack	37, 42, 181, 187
buffer overflow	135, 136, 137, 138, 139, 140 ...	password	27, 28, 30, 32, 35, 36 ...
Caesar cipher	51, 54, 57, 62, 66, 68 ...	phishing	24, 28, 30, 32, 33, 34 ...
certificate	109, 110, 111, 116, 118, 119 ...	PKI	116, 117, 118, 119, 120, 121 ...
certificate authority	119, 124, 180, 188	plaintext	46, 47, 48, 50, 54, 55 ...
CIA Triad	24, 26, 31, 33, 34, 36 ...	Playfair	48, 54, 55, 56, 59, 60 ...
ciphertext	46, 47, 48, 50, 52, 54 ...	Rail Fence	48, 50, 52, 65, 67, 68 ...
confidentiality	24, 26, 27, 28, 30, 31 ...	ransomware	29, 30, 31, 33, 36, 185
Content Security Policy	166, 167, 194	RBAC	143, 145, 146, 148, 149, 150 ...
cross-site scripting	164, 166, 181	RC4	82, 83, 86, 87, 88, 89 ...
cryptanalysis	46, 50, 52, 74, 75, 76 ...	replay	37, 38, 39, 40, 41, 42 ...
cryptography	36, 39, 45, 46, 47, 48 ...	responsible disclosure	167, 171, 173, 187
denial of service	26, 38, 40, 43, 127, 152 ...	REVOKE	104, 111, 118, 119, 143, 144 ...
DES	24, 26, 28, 29, 31, 32 ...	risk exposure	91, 94, 97, 98, 178, 182 ...
digital signature	26, 35, 39, 40, 41, 43 ...	risk register	91, 94, 95, 97, 98, 99 ...
DMZ	152, 153, 154, 156, 160, 161 ...	RSA	46, 48, 51, 74, 89, 91 ...
encipherment	39, 40, 43	S-box	84, 87, 88, 89, 102, 105 ...
Euclidean algorithm	91, 92, 93, 95, 97, 110 ...	salt	112, 117, 126, 127, 128, 129 ...
Feistel	82, 84, 85, 87, 88, 89 ...	session	24, 25, 47, 109, 112, 118 ...
firewall	24, 25, 27, 43, 141, 144 ...	SHA-256	89, 116, 117, 120, 123, 124 ...
frequency analysis	54, 55, 59, 61, 62, 68 ...	social engineering	24, 28, 33, 34, 181, 183 ...
GCM	82, 90, 101, 103, 104, 105 ...	spoofing	35, 152, 153, 161, 162
GRANT	126, 131, 143, 144, 146, 147 ...	SQL injection	144, 159, 164, 165, 168, 170 ...
hash function	116, 117, 118, 119, 120, 121 ...	stack canary	137, 140, 186
Hill cipher	74, 75, 76, 77, 78, 79 ...	stateful inspection	153, 160, 161
honeypot	152, 156, 159, 160, 161, 162 ...	stream cipher	52, 65, 82, 83, 84, 85 ...
incident response	155, 156	SubBytes	101, 102, 103, 105, 106, 107 ...
input validation	135, 137, 140	symmetric	46, 47, 48, 50, 51, 52 ...
integrity	24, 26, 27, 28, 30, 31 ...	threat	24, 26, 27, 29, 30, 32 ...
intrusion detection	152, 153, 154, 155, 156, 157 ...	TOTP	126, 128, 130, 131, 132, 133 ...
least privilege	29, 30, 35, 40, 104, 135 ...	Trojan	29, 30, 33, 34, 146, 182
malware	24, 27, 28, 29, 30, 32 ...	Vigenere	48, 65, 66, 68, 69, 70 ...
masquerade	38, 40, 41, 42	virus	24, 29, 30, 33, 34, 35 ...

MixColumns	74, 101, 102, 103, 105, 106 ...	vulnerability	24, 26, 27, 33, 34, 35 ...
modular arithmetic	46, 48, 49, 50, 91, 92 ...	worm	29, 30, 33, 34, 35, 136 ...