Selver Softic

FOUNDATIONS AND APPLICATIONS OF MODERN CRYPTOGRAPHY

PRESA UNIVERSITARĂ CLUJEANĂ

Selver Softic

Foundations and Applications of Modern Cryptography

PRESA UNIVERSITARĂ CLUJEANĂ
2026

Scientific reviewers:

Assoc. Prof. Dr. Vlad Bocăneț

Technical University of Cluj-Napoca

Dipl.-Ing. Safet Softic

CAMPUS 02 University of Applied Sciences, Graz, Austria

Foundations and Applications of Modern Cryptography

FH-Prof. DI Dr. techn. Selver Softic, BSc

CAMPUS 02 University of Applied Sciences, Graz, Austria

ISBN 978-606-37-3020-7

Universitatea Babeș-Bolyai

Presa Universitară Clujeană

Director: Codruța Săcelean

Str. B.P. Hasdeu nr. 51

400371 Cluj-Napoca, România

Tel.: +40 744 687 884

E-mail: editura@ubbcluj.ro

www.editura.ubbcluj.ro | libraria.ubbcluj.ro

Contents

Preface

This work provides a comprehensive introduction to modern cryptography, designed for students and practitioners seeking to understand the cryptographic foundations underlying digital security. The material progresses from historical context and classical methods through contemporary standards (AES, RSA) to practical protocol implementations (TLS, Kerberos, blockchain).

Image and Illustration Credits

All diagrams, illustrations, and visual materials in this manuscript were generated by the author using a combination of manual design and artificial intelligence-assisted tools. Specifically:

- **All images, diagrams and process flowchart**s were created using AI-assisted image generation to ensure technical accuracy and clarity.
- **Data tables and structured comparisons** were formatted by the author for pedagogical effectiveness.
- **Cover design** incorporating the lock-and-circuit motif was generated using AI tools to reflect cryptographic concepts visually.

The use of AI in image generation enhances accessibility and consistency across the manuscript while maintaining the integrity of technical content. All underlying concepts, mathematical formulations, and cryptographic principles are presented with rigorous adherence to established standards (NIST, RFC, academic literature).

Manuscript Information

- **Author**: FH-Prof. DI Dr. techn. Selver Softic, BSc
- **Institution**: CAMPUS 02, University of Applied Sciences, Graz, Austria

- **Date of Publication**: 2026
- **Language**: English
- **Target Audience**: Computer science students, IT practitioners, researchers
- **Copyright Images**: All images are AI generated

Scientific reviewers

Assoc. Prof. Dr.Vlad Bocăneț
Technical University of Cluj-Napoca, Romania

Dipl.-Ing. Safet Softic
CAMPUS 02 University of Applied Sciences, Graz, Austria

1. Introduction to the world of cryptography

1.1. The Strategic Importance of Cryptography

In our digitally connected world, cryptography forms the invisible foundation for trust and security (Katz & Lindell, 2020). From everyday applications such as e-commerce, secure internet banking and encrypted email to revolutionary technologies such as cryptocurrencies, cryptography is indispensable wherever sensitive information is transmitted or stored. It is the science that enables us to protect digital communications from unauthorized access, ensure the integrity of data, and verify the authenticity of communication partners.

Cryptography has evolved from ancient secret-keeping techniques to a cornerstone of cybersecurity in an era of pervasive data breaches and quantum computing threats. As digital transformation accelerates—particularly in cloud-native environments and AI-driven systems—understanding cryptographic primitives is essential for professionals in computer science and IT infrastructure.

This book provides an introduction to the foundational concepts, historical milestones, and modern applications that define digital security today.

Chapter Overview

This overview builds on the manuscript's table of contents, detailing each chapter's core objectives, key subtopics, learning outcomes, and interconnections to modern applications.

Chapter 1: Introduction to the World of Cryptography

Establishes foundational context, tracing cryptography from ancient ciphers (Skytale, Caesar, Vigenère) through WWII Enigma to public-key revolutions like RSA. Defines core goals—confidentiality, integrity, authenticity, non-repudiation—and introduces the Alice-Bob-Eve-Mallory model for threat analysis. Readers gain historical perspective and terminology essential for subsequent technical discussions.

Chapter 2: Symmetric Encryption Methods – The Principle of Shared Secrets

Explores efficient encryption using a single shared key, starting with classical techniques (substitution, transposition, One-Time Pad). Covers modern stream/block ciphers, Shannon's confusion-diffusion principles, block modes (ECB, CBC, CFB, OFB, CTR with pros/cons), and evolution from DES/3DES to AES (Rijndael structure: S-boxes, ShiftRows, MixColumns). Emphasizes key distribution challenges and performance for bulk data in real-time systems.

Chapter 3: Asymmetric Encryption Methods – The Public Key Revolution

Shifts to key pairs solving symmetric limitations, detailing Diffie-Hellman for secure key exchange (discrete logarithm problem) and RSA (factorization trapdoor, key gen/encryption/decryption formulas). Extends to digital signatures for authenticity/non-repudiation, swapping key roles with hashing. Prepares readers for hybrid protocols combining asymmetric handshakes with symmetric payloads.

Chapter 4: Ensuring Data Integrity – Hash Functions and MACs

Beyond secrecy, focuses on tamper-proofing via one-way hashes (pre-image/collision/second-preimage resistance, birthday attacks) and standards (MD5/SHA-1 broken; SHA-2/3 secure). Introduces MACs (HMAC) for authenticity and AEAD modes like GCM (CTR + Galois authentication). Links to practical needs in file verification and protocol integrity.

Chapter 5: Applied Cryptography – Protocols and Systems in Practice

Integrates primitives into systems: TLS/HTTPS (handshake, cipher suites, attacks like BEAST/CRIME/POODLE), Kerberos SSO (KDC/TGT flow), secure passwords (salts, PBKDF2/bcrypt/Argon2), and blockchain (double-spending via PoW, Merkle trees, mining). Highlights real-world deployment, vulnerabilities, and hybrid designs for networks/applications.

1.2. A historical perspective

The history of cryptography stretches back to the origins of writing itself around 3000 BC, mirroring humanity's fundamental instinct to protect secrets from prying eyes.

The development can be divided into several epochs:

Antiquity: The earliest forms of cipherwriting can be traced back to about 3000 BC, for example in the form of secret hieroglyphics.

Systematic procedures emerged later:

Skytale: The Spartans (c. 500 BC) used a wooden stick (Skytale) around which a leather strap was wrapped. The message was written across the windings and was only readable when the tape was wrapped around a stick of identical diameter – an early example of transposition.

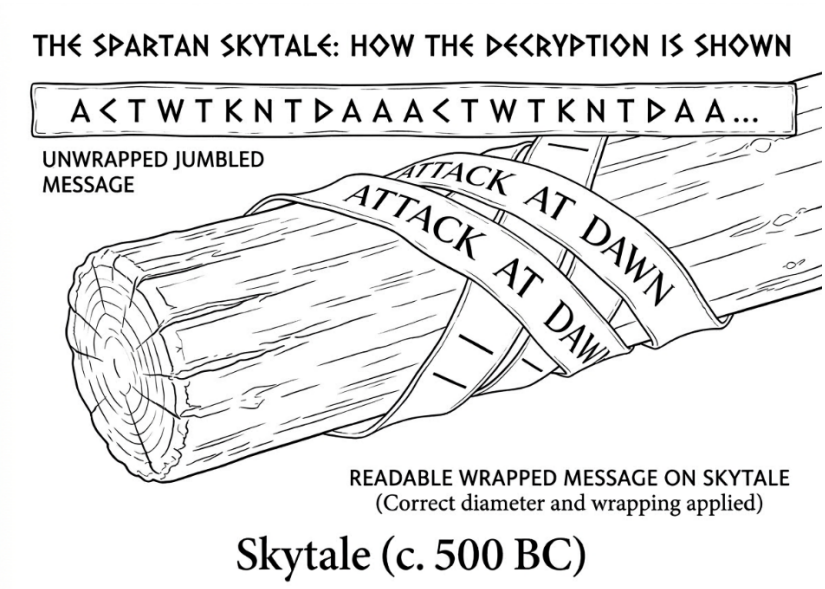
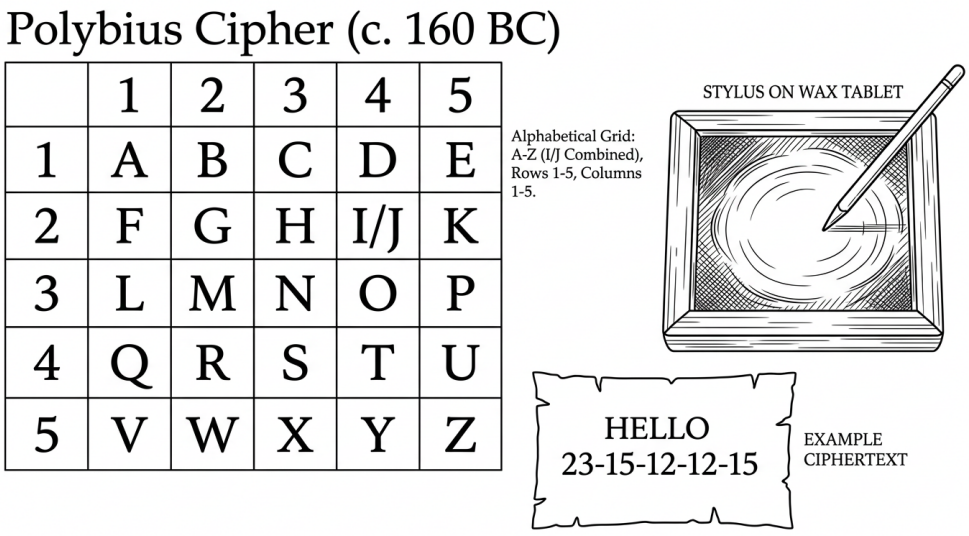


Figure 1. *Skytale introduced by Spartans*

Polybius cipher: The Greek Polybius (c. 160 BC) developed a 5x5 square to convert letters into pairs of numbers.



	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I/J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Figure 2. *Polybius cipher*

Caesar cipher: Julius Caesar (c. 60 BC) used a simple letter shift (e.g. by three places), a method of monoalphabetic substitution.

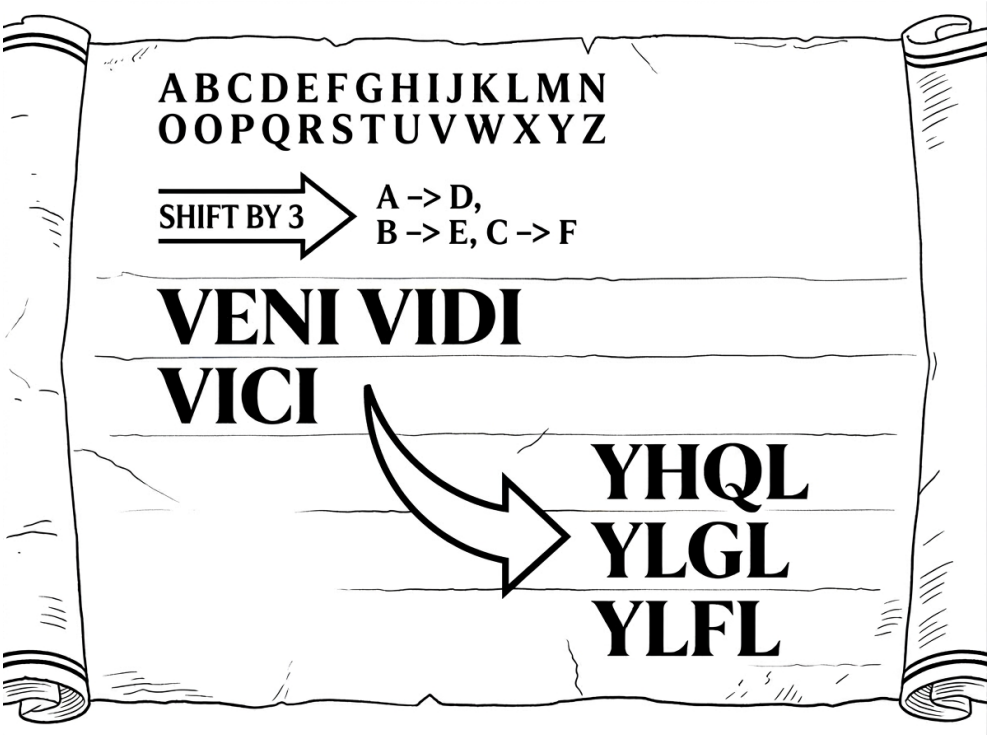


Figure 3. *Caesar cipher*

Middle Ages to early modern times: While simple substitution methods dominated in the early Middle Ages, cryptography experienced an upswing from the 15th century onwards.

Al-Kindi (9th century): The Arab scholar laid the foundation for modern cryptanalysis with the statistical analysis of letter frequencies.

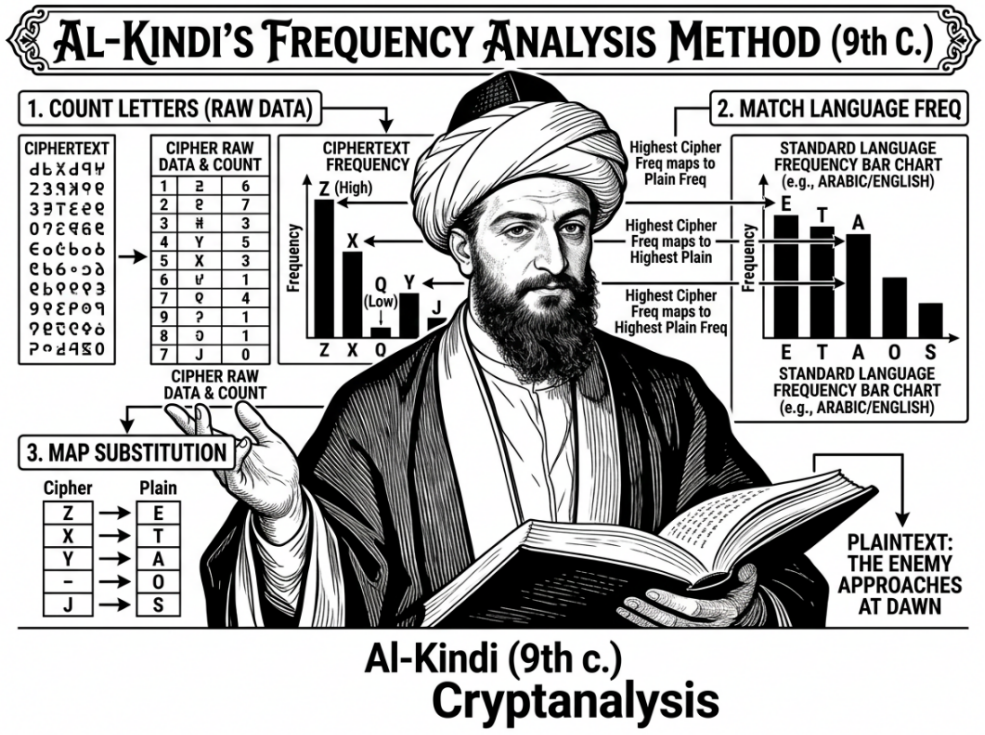


Figure 4. *Al-Kindi's cryptanalysis*

Leon Battista Alberti (15th century): He invented a cipher disk that simplified encryption and enabled polyalphabetic approaches.

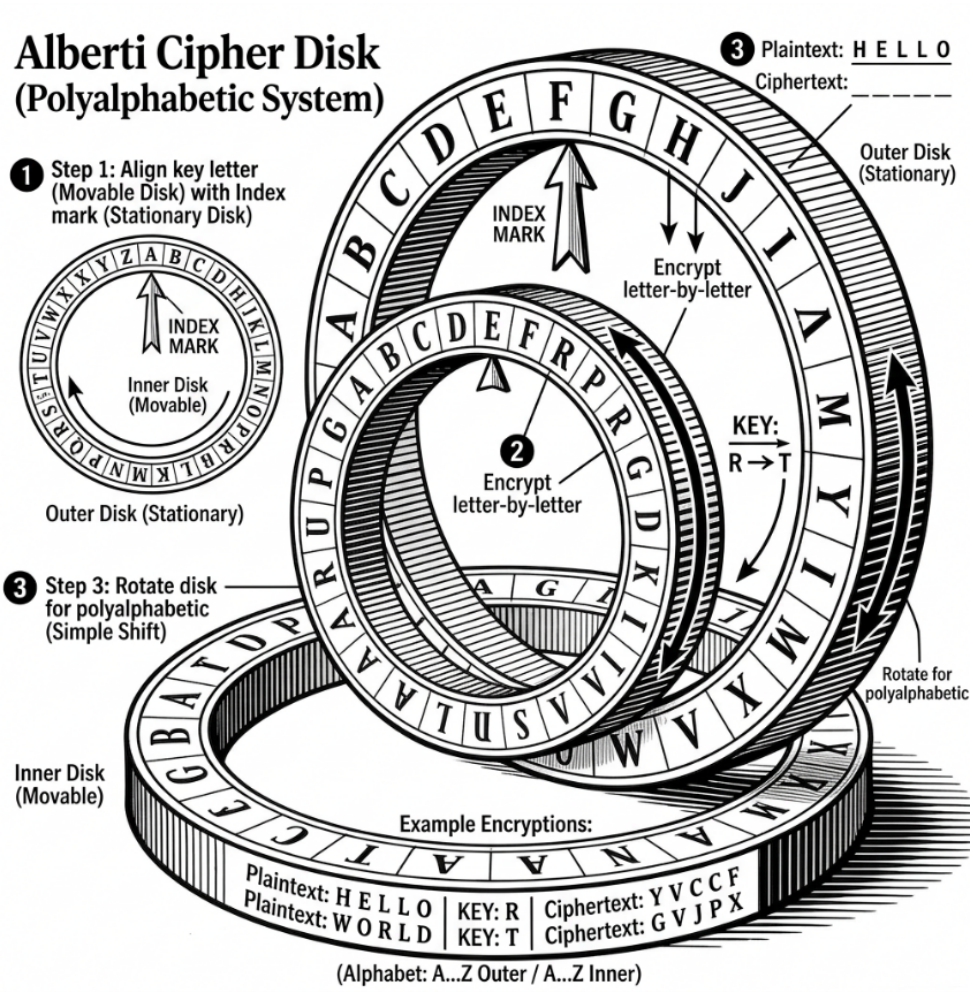


Figure 5. *Alberti's cypher disks*

Blaise de Vigenère (16th century): He developed the Vigenère cipher named after him, a polyalphabetic process that was considered "uncrackable" for centuries.

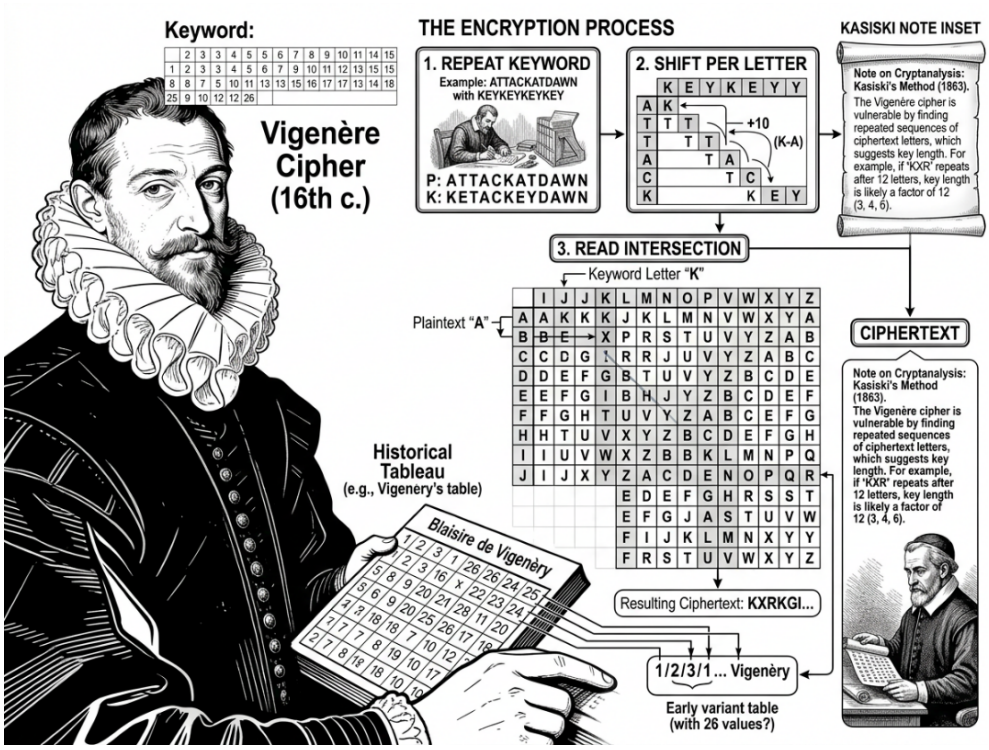


Figure 6. *Vigenère's cipher*

1.2.1. The Enigma machine and the rise of modern cryptanalysis

A particularly important turning point in the history of cryptography occurred during the Second World War with the use of the German Enigma machine. Enigma was an electromechanical cipher machine that used rotating wheels, plugboard substitutions, and reflector mechanisms to produce an enormous number of possible encryption settings. To German operators, this system appeared secure because the internal wiring changed the substitution with every keystroke, making the ciphertext highly dynamic and far more complex than classical monoalphabetic ciphers.

The strength of Enigma did not lie in a single substitution alphabet, but in the combination of several cryptographic transformations. Each pressed key

passed through the plugboard, then through a set of rotating rotors, and finally through a reflector before being mapped back to a ciphertext letter. Because the rotors advanced during operation, the same plaintext letter could be encrypted differently each time it appeared. This principle represented a major step from manual ciphers toward machine-based cryptography and demonstrated how mechanical design could dramatically increase cryptographic complexity.

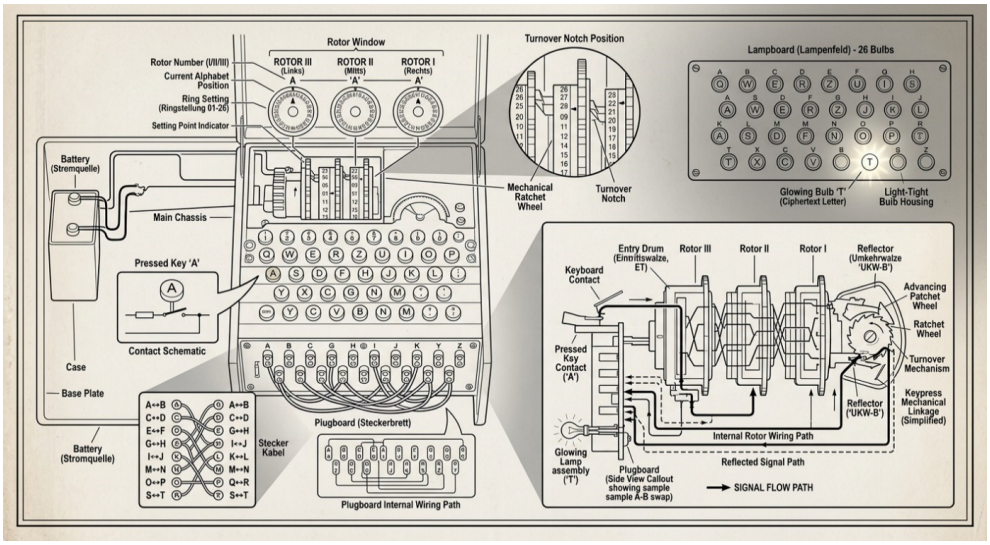


Figure 7. *Eingma machine*

The decisive breakthrough against Enigma was achieved by Allied cryptanalysts, especially at Bletchley Park in the United Kingdom. Building on earlier Polish work, British mathematicians and engineers developed systematic methods for analyzing intercepted messages. Alan Turing and his colleagues designed the Bombe, a machine that helped test possible rotor settings much faster than manual analysis would have allowed. This was not a matter of “guessing the key” in a simple sense, but of exploiting structural weaknesses, operational habits, and probable words in messages, often called cribs.

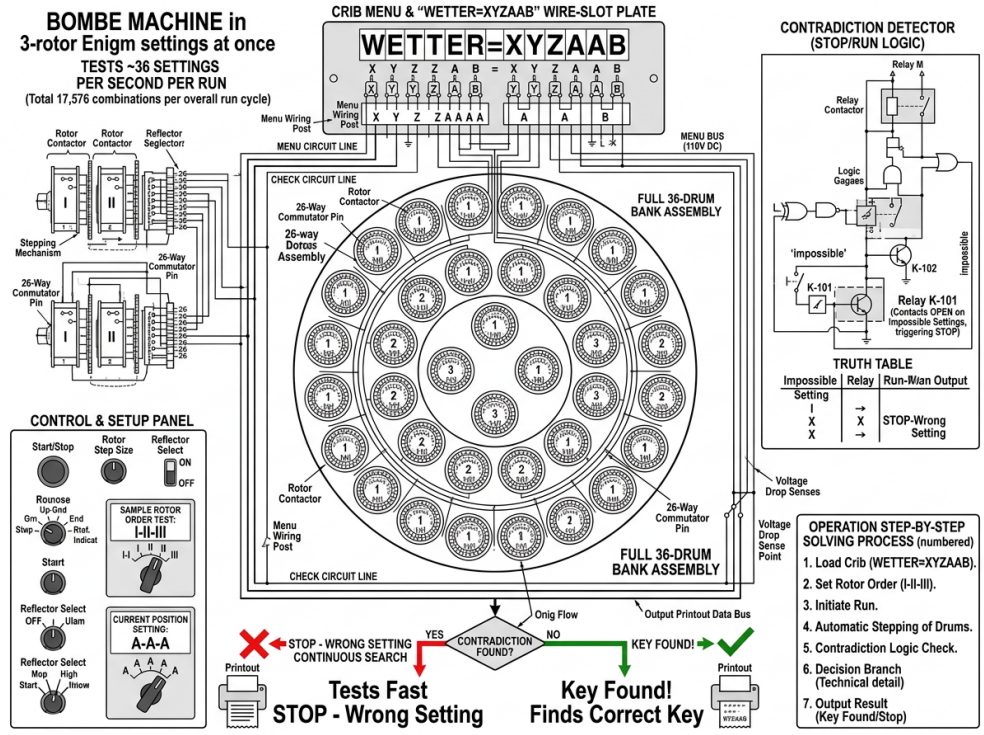


Figure 8. *Bombe Machine for decrypting Enigma*

The successful cryptanalysis of Enigma had enormous strategic consequences. It allowed the Allies to gain insight into German military communications, particularly in naval warfare, where convoy protection in the Atlantic was of critical importance. More broadly, the Enigma story marks the emergence of modern cryptanalysis as an interdisciplinary effort involving mathematics, engineering, statistics, linguistics, and intelligence analysis. It also showed that even very sophisticated cryptographic systems can fail if implementation details, usage procedures, or operational discipline are weak.

1.2.2. From mechanical ciphers to mathematical cryptography

The period after the Second World War marked a profound transformation in cryptography. Earlier systems had often relied on physical devices, manual

procedures, or electromechanical machines. With the rise of digital computing, cryptography increasingly became a mathematical discipline based on formal models, algorithmic design, and computational hardness assumptions. This transition laid the foundation for the forms of cryptography that secure digital communication today.

One major consequence of this change was the increasing importance of algorithm standardization. Instead of relying on secret mechanical constructions, governments, researchers, and standards bodies began to define openly specified cryptographic algorithms whose security depended not on secrecy of design, but on secrecy of the key. This principle is consistent with Kerckhoffs' idea that a cryptographic system should remain secure even if its method is publicly known. In the computer age, this approach enabled broad scientific review and fostered trust in standardized algorithms.

This new era also changed the types of attacks that had to be considered. In classical cryptography, attackers often relied on language patterns, repeated symbols, or human mistakes in manual use. In modern cryptography, attacks increasingly involve computational analysis, algebraic properties, side-channel observations, protocol weaknesses, and brute-force search. As a result, cryptography developed into a field closely linked to computer science, complexity theory, and information security engineering.

The modern period therefore represents more than just a change in tools. It reflects a conceptual shift from secret writing as an art of concealment to cryptography as a rigorous scientific discipline. This scientific foundation later enabled the development of symmetric standards such as DES and AES, as well as the breakthrough of asymmetric cryptography with Diffie-Hellman and RSA.

1.3. Basic Goals and Terms

Cryptology, the science of the "hidden", is divided into two central disciplines:

1. **Cryptography:** From the Greek for "to hide" and "to write", it is the study of encrypting data to ensure secrecy.
2. **Cryptanalysis:** The counterpart to cryptography that deals with breaking encryption methods without knowledge of the key.

Modern cryptography pursues four main protection goals:

1. **Confidentiality:** Ensuring that data can only be read by authorized persons.
2. **Data integrity:** Ensuring that data cannot be altered unnoticed during transmission or storage.
3. **Authenticity:** Verification of the identity of the communication participants (persons, devices or organizations).
4. **Non-repudiation:** Ensuring that an action taken cannot be disputed after the fact.

1.4. The communication model and the actors

To describe cryptographic scenarios, standardized roles are used that allow clear communication about the intentions of the parties involved:

- **Alice and Bob:** The two legitimate participants who want to communicate securely with each other.
- **Eve (from eavesdropper, listener):** A passive attacker who eavesdrops on the communication channel but does not actively alter messages.
- **Mallory (from malicious attacker, malicious attacker):** An active attacker who not only eavesdrops on messages but also modifies, deletes, or injects new messages.

In this model, Alice and Bob exchange messages through an insecure channel that Eve and Mallory have access to. The task of cryptography is to secure this communication against the threats posed by Eve and Mallory. Below, we will analyze the specific procedures that make this possible.

2. Symmetric Encryption Methods: The Principle of Shared Secrets

2.1. Introduction: How it works and challenges

Symmetric cryptography represents the foundational and most intuitive approach to encryption, where the sender (Alice) and receiver (Bob) share a single secret key for both encrypting and decrypting messages.

2.1.1. Key Characteristics

Symmetric cryptography represents the foundational and most intuitive approach to encryption, where the sender (Alice) and receiver (Bob) share a single secret key for both encrypting and decrypting messages. This shared-key model mimics a physical safe with identical copies of the lock: anyone possessing the key can lock or unlock it, ensuring only mutual holders access the contents.

Symmetric algorithms excel in speed and computational efficiency, processing vast data volumes—such as video streams or database backups—at minimal overhead, making them indispensable for bulk encryption in real-time applications like secure file storage or network tunnels. Their performance advantage stems from simple bitwise operations (XOR, shifts, table lookups) rather than complex modular exponentiation required by asymmetric methods.

However, this efficiency creates the core limitation: key distribution. Before secure communication can begin, Alice and Bob must somehow share the secret key through a secure channel. In small, trusted environments this might mean physical delivery or pre-shared keys, but scaling to millions of

internet users makes this impractical. Eve intercepting the key compromises all future sessions—an attack called a compromise-recovery attack.

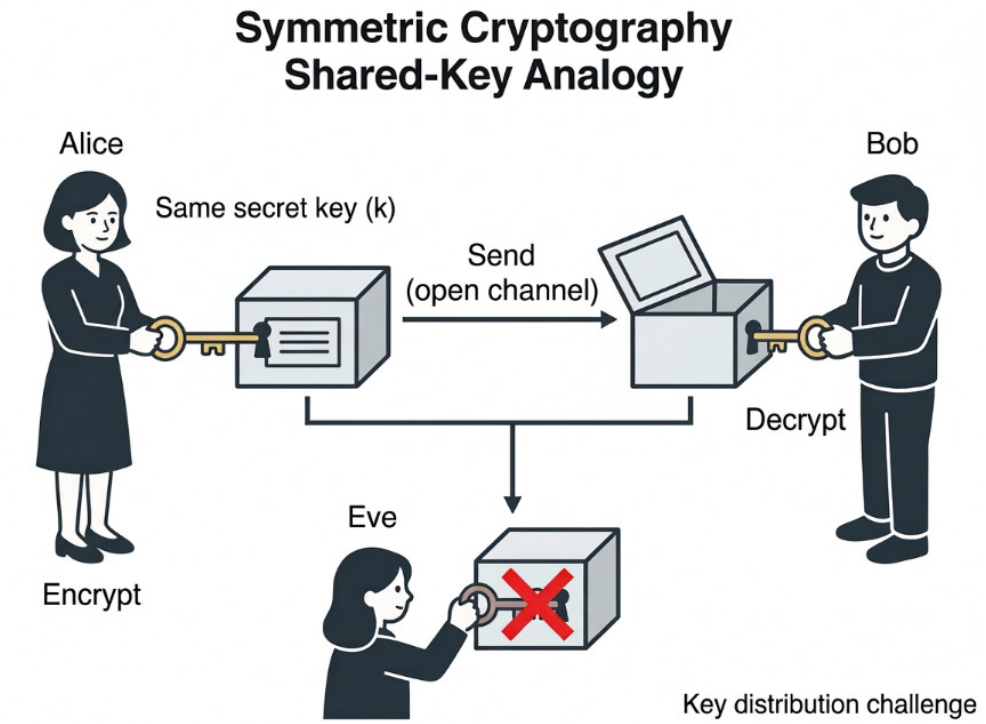


Figure 9. *Symmetric cryptography*

2.1.2. Historical and Technical Context

Rooted in classical methods like the Caesar cipher or Vigenère tableau, modern symmetric ciphers evolved to counter brute-force attacks through expansive key spaces (e.g., AES-256's 2^256 possibilities ≈ 10^77 combinations). They leverage Claude Shannon's two fundamental principles formalized in 1949:

Confusion: Makes the relationship between plaintext/key and ciphertext as complex and nonlinear as possible. Achieved through S-boxes (substitution boxes)—lookup tables designed to resist algebraic cryptanalysis.

Diffusion: Ensures each plaintext bit influences many ciphertext bits (ideally half). Achieved through bit permutations, matrix multiplications, and key mixing.

Modern ciphers iterate these operations through multiple rounds (AES: 10-14, DES: 16), each with round-specific subkeys derived from the master key. This product cipher construction amplifies security exponentially while maintaining efficient implementation.

2.1.3. Core Challenge: Key Distribution

The fundamental problem of symmetric cryptography is elegantly simple: how do two parties establish a shared secret over an insecure channel, (Menezes et al., 1996)? Physical alternatives exist (diplomatic pouches, trusted couriers), but they don't scale to internet communication.

Historical Solutions:

- **STU-III**: US secure telephones distributed monthly key tapes via armored transport
- **KL-7: US** cipher machine with physical keylist distribution
- **One-Time Pad:** Perfect security but requires key as long as message

Mathematical Insight: Perfect forward secrecy requires ephemeral keys per session. Static long-term keys create a single point of failure—compromise reveals all past/future traffic.

This limitation drove the invention of public-key cryptography (Diffie-Hellman, RSA) specifically to solve symmetric key distribution.

2.1.4. Practical Implications

Symmetric methods power 99% of encrypted data volume today:

- **TLS 1.3**: Asymmetric handshake → AES-256-GCM bulk data
- **Disk encryption**: BitLocker, LUKS (AES-XTS)
- **VPNs**: IPsec ESP (AES-CBC/GCM)

Performance Comparison:

- **Asymmetric**: 1000-bit RSA signature = 1-10ms
- **Symmetric**: AES-256-GCM 1MB = 0.1-1ms

Symmetric crypto processes gigabytes per second on modern CPUs with AES-NI acceleration, making it irreplaceable for bulk data.

2.2. Classical Building Blocks: Substitution and Transposition

Modern block ciphers are product ciphers combining multiple simple transformations in sequence. Each round applies:

- **Substitution** (confusion via S-boxes)
- **Permutation** (diffusion via P-boxes)
- **Key mixing** (round subkey XOR)

Security Amplification: n identical rounds with different subkeys provides security far exceeding any single round. DES's 16 rounds resisted differential cryptanalysis; AES's

But what exactly happens in those transformations that create a product cipher:

Substitution: Substitution is the process of replacing characters of plaintext with other characters. The simplest example is the **Caesar cipher**, in which each letter is shifted by a fixed number of positions in the alphabet. However, such monoalphabetic substitutions, in which each letter is always replaced by the same ciphertext letter, are very susceptible to **frequency analysis**. Since the frequency distribution of the letters in a language (e.g. 'E' in German) is preserved, the key can be easily reconstructed.

Transposition: In contrast to substitution, the signs of the plain text are not changed here, but only their positions are reversed. A historical example is the **Scytale** of the Spartans. The purpose of transposition methods is to achieve **diffusion** by distributing the statistical properties of the plaintext over the ciphertext, which makes frequency analysis difficult.

One-Time-Pad (OTP): The OTP, also known as the Vernam cipher, is the only method that is mathematically probably uncrackable. It combines the plaintext bitwise with a key stream via XOR operation. Security is based on three strict conditions:

1. The key must be random.
2. The key must be at least as long as the message to be encrypted.
3. The key may only be used once. Due to these strict requirements for key generation and distribution, OTP is only suitable for very specific use cases in practice.

2.3. Modern Approaches: Stream and Block Ciphers

Modern symmetric ciphers divide into two primary categories: **stream ciphers** and **block ciphers,** each optimized for specific use cases while embodying Shannon's confusion and diffusion principles.

Stream ciphers

These methods imitate the principle of the one-time pad. A relatively short secret key is used to generate a long, pseudo-random key stream. The plaintext is then linked character by character (or bit by bit) to this key stream via XOR. Stream ciphers are very fast and are well suited for real-time transmissions (e.g. mobile communications), where data is generated as a continuous stream. A well-known, but today considered **uncertain** representative is **RC4**. Research showed weaknesses in the key stream from 2001 onwards, especially at the beginning. Certain byte patterns occur more frequently than with a true random sequence, which allows for statistical attacks. For this reason, the use of RC4 is prohibited in modern protocols such as TLS.

Block ciphers

These algorithms divide the data into fixed-length blocks (e.g., 128 bits) and encrypt each block individually. To achieve a high level of safety, modern block ciphers combine the classical principles of substitution and transposition in several consecutive rounds. Claude Shannon formulated two central concepts for this (Shannon, 1949):

1. **Confusion:** Makes the relationship between the ciphertext and the key as complex as possible. This is typically achieved by substitutions (so-called S-boxes).

2. **Diffusion:** Distributes the influence of a single plaintext bit to as many ciphertext bits as possible. This is achieved through permutations (reversals). The repeated use of these two operations in so-called product ciphers ensures a high level of security.

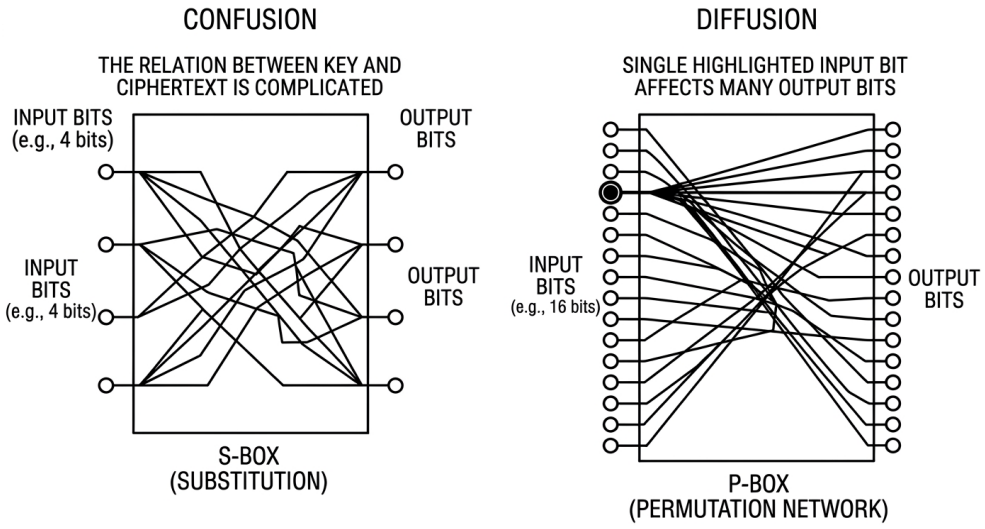


Figure 10. *Principles of confusion and diffusion*

2.4. Operating modes for block ciphers

Block ciphers process fixed-size data chunks (e.g., 128 bits in AES), so encrypting messages longer than one block requires structured approaches called modes of operation.

Purpose and Functionality

Modes define precise rules for chaining or combining successive block encryptions, preventing repetition vulnerabilities and enabling secure handling of arbitrary-length data. They transform a basic block cipher into a versatile tool for streams, files, or network payloads by incorporating feedback from prior blocks or counters, balancing security, speed, and error resilience.

Key Design Principles

Each mode addresses trade-offs in parallelism, error propagation, and pattern leakage: Electronic Codebook (ECB) encrypts blocks independently for simplicity but leaks identical plaintext patterns; Cipher Block Chaining (CBC) XORs each plaintext block with the previous ciphertext (using an Initialization Vector for the first), hiding repeats at the cost of sequential processing; Counter (CTR) mode generates keystreams from incrementing counters for full parallelism and no error propagation, ideal for disk encryption.

Table 1. *Five main block modes*

Mode	Principle of operation (short)	Rating (Advantages/Cons)
ECB	Each block is encrypted independently of the other.	**Advantage:** Parallelizable. **Disadvantage:** Identical plaintext blocks result in identical ciphertext blocks, which keeps patterns visible. **Considered unsafe for most applications (Katz & Lindell, 2020).**
CBC	Each plaintext block is linked to the previous ciphertext block via XOR before encryption.	**Advantage:** Makes patterns unrecognizable. **Disadvantage:** Not parallelizable in encryption. A bit error in the ciphertext corrupts the entire plaintext block and inverts the corresponding bit in the following block.
CFB	Turns the block cipher into a self-adjusting stream cipher. The previous block of ciphertext is encrypted to create a stream of keys.	**Advantage:** Only requires the encryption function of the block cipher. **Disadvantage:** Not parallelizable. Error propagation similar to CBC.
OFB	Turns the block cipher into a stream cipher. A key stream is generated independently of the plaintext.	**Advantage:** Key current can be pre-calculated. Bit errors do not propagate, but only affect the corresponding bit in plain text.
CTR	Creates a key stream by encrypting the values of a counter.	**Advantage:** Fully parallelizable. Direct access (random access) to each block is possible. Bit errors do not propagate.

ECB (Electronic Code Book): The serious weakness of this mode is that the same plaintext blocks lead to the same ciphertext blocks. When encrypting image files, for example, the contours of the original image remain visible in the ciphertext.

CBC (Cipher Block Chaining): This mode solves the problem of ECB by chaining. For the first block, a random **initialization vector (IV)** is required, which ensures that even identical messages result in different ciphertexts for each encryption.

2.5. Modern standards: From DES to AES

Symmetric ciphers reached maturity through standardized algorithms like DES and AES, balancing security, performance, and practicality for widespread adoption.

Feistel networks form the structural backbone of DES and many symmetric ciphers, offering an elegant, invertible design that simplifies implementation and decryption.

2.5.1. Feistel Network Explained

Named after Horst Feistel, this construction splits a 64-bit block into equal left (L) and right (R) halves, processing through multiple (typically 16) iterative rounds. In each round, the right half encrypts a function of itself (f(R, round-key)) and XORs the result with the left half, then swaps halves for the next round:

- $L_{i+1} = R_i$
- $R_{i+1} = L_i \oplus f(R_i, K_i)$

The function f—often combining key-dependent substitution (S-boxes) and permutation—provides confusion and diffusion without one-way traps.

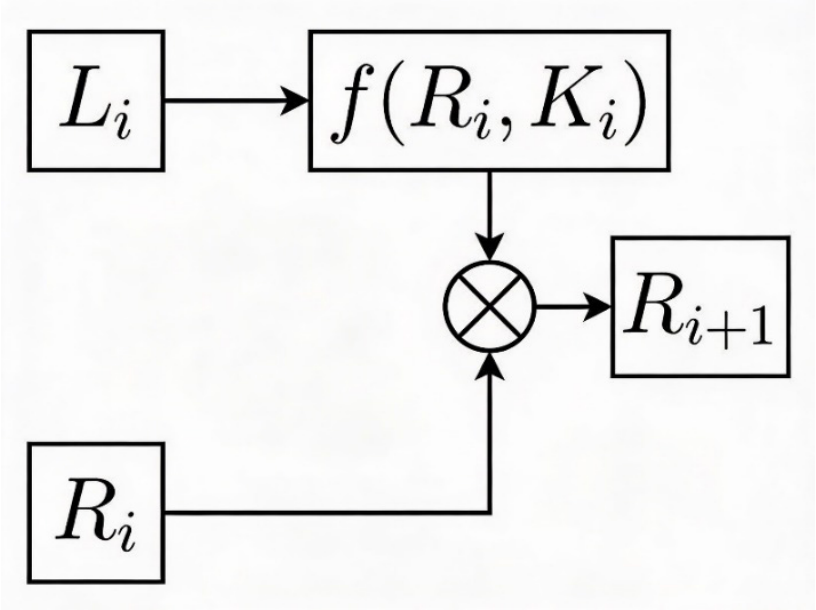


Figure 11. *Basic construction of a Feistel network element*

2.5.2. Decryption Advantage

Feistel's genius lies in symmetric encryption/decryption: decryption mirrors encryption but reverses subkeys (K_{16} to K_1), as XOR's self-inverse property and swapping undo rounds perfectly—no separate decrypt logic needed. This balances security with efficiency, unlike asymmetric networks requiring inverse computations.

2.5.3. Strategic Importance

- **Proven Resilience**: DES S-boxes thwarted differential cryptanalysis (Biham-Shamir, 1990s), validating Feistel's balance of rounds/keys.

- **Iterative Scalability**: More rounds enhance diffusion/confusion margins without redesign.
- **Modern Legacy**: Influences Blowfish, Twofish (AES finalists), Camellia; even non-Feistel AES borrows substitution-permutation principles.

While AES evolved beyond Feistel for larger blocks/keys, the paradigm endures for resource-constrained systems (IoT, smartcards) where decryption symmetry cuts code size. Insert before AES section for contextual flow.

2.5.4. DES: The Pioneering Standard

Standardized by NIST in 1977 after a public competition, the Data Encryption Standard (DES) dominated symmetric encryption for over two decades. It employs a Feistel network—a balanced structure dividing each 64-bit block into left/right halves, processed through 16 iterative rounds where one half encrypts the other via a round-specific subkey (derived from the 56-bit master key).

Each round applies:

- Expansion/Permutation: Expands the right half to interact more broadly.
- S-box Substitution: Eight nonlinear lookup tables (S-boxes) providing confusion, compressing 48 bits to 32.
- P-box Permutation: Rearranges bits for diffusion.
- Subkey XOR: Integrates round key.

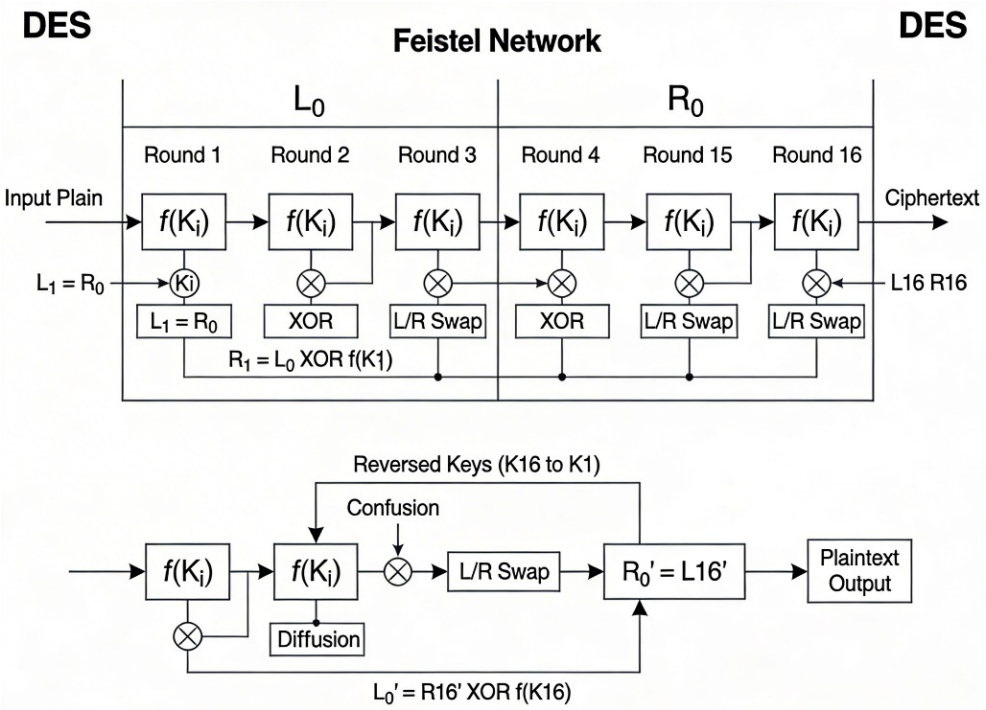


Figure 12. *DES encryption implementation*

Despite initial controversy over its short key (vulnerable to brute-force by 1990s supercomputers), DES proved robust against differential cryptanalysis—its S-boxes were deliberately designed against known attacks. By 1999, brute-force cracked it in days, prompting Triple DES (3DES): E-D-E with three 56-bit keys (effective ~112 bits), using distinct keys (K1/K2/K3) for encryption-decryption-encryption. While secure, 3DES triples computation time (~3x slower than single DES), limiting it to legacy transitions.

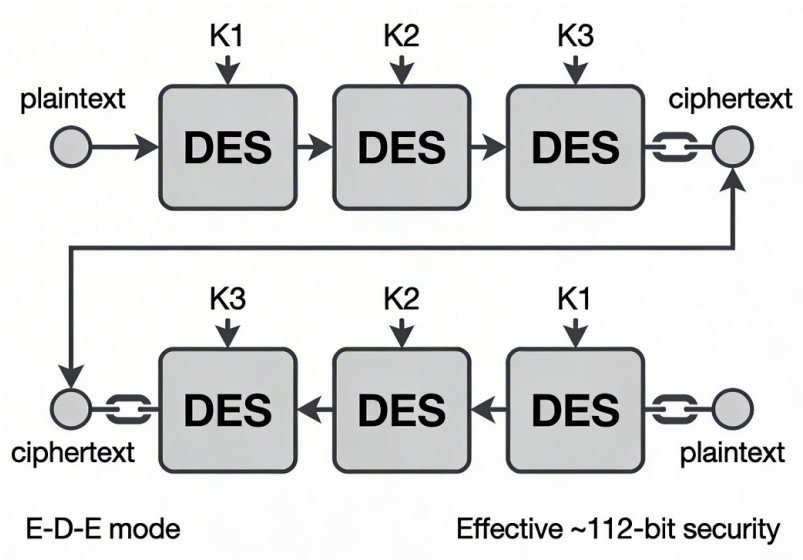


Figure 13. *3DES encryption blocks*

2.5.5. AES: The Modern Global Standard

In 2000, NIST selected Rijndael (by Daemen and Rijmen) as the Advanced Encryption Standard (AES) after evaluating 15 candidates, replacing DES entirely by 2004. AES processes 128-bit blocks with variable key sizes (128/192/256 bits), using 10/12/14 rounds respectively—not a Feistel but a substitution-permutation network on a 4x4 byte state matrix (128 bits). Each round (except final) executes:

1. **SubBytes** (ByteSub): Nonlinear S-box substitution per byte, core confusion layer resisting linear/differential cryptanalysis.
2. **ShiftRows**: Cyclic left-shifts rows (0/1/2/3 positions), providing partial diffusion across columns.
3. **MixColumns**: Galois field matrix multiplication per column, full diffusion—transforms one-bit flip into four-byte changes.
4. **AddRoundKey**: XOR with round-expanded key (via key schedule using S-box/RotWord).

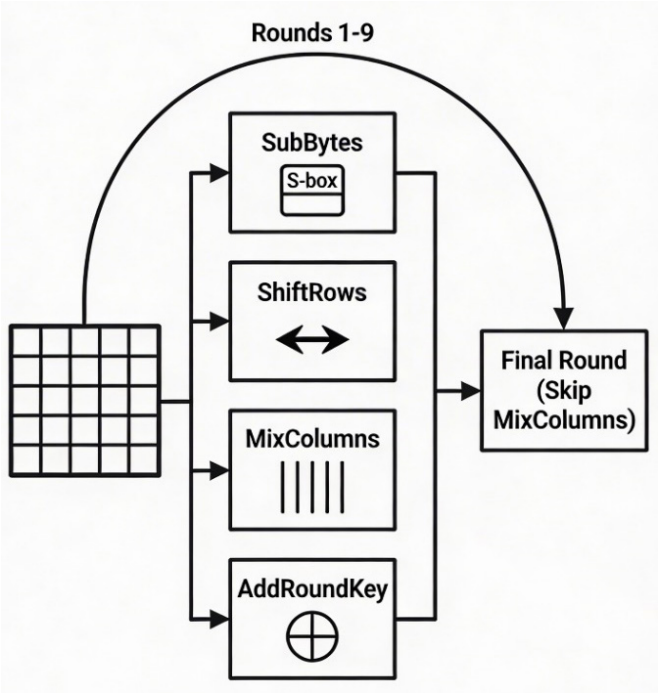


Figure 14. *AES encryption for 128 bit*

The key schedule generates round keys from the cipher key through expansions, rotations, and substitutions, ensuring unique subkeys. Final round omits MixColumns for decryption efficiency.

Table 2. *Comparison of symmetric encryption standards*

Feature	DES	3DES	AES-128	AES-256
Block Size	64 bits	64 bits	128 bits	128 bits
Key Size	56 bits	168 bits (3×56)	128 bits	256 bits
Structure	Feistel (16 rounds)	Feistel (48 rounds)	SPN (10 rounds)	SPN (14 rounds)
Speed (GB/s, modern CPU)	~0.1	~0.05	~5-10	~2-5
Security Status	Broken	Deprecated (NIST 2023)	Secure (quantum-resistant key sizes)	Secure

2.5.6. Evolution and Security Posture

AES resists all known practical attacks, with margins against biclique cryptanalysis exceeding 2^100 operations for AES-128. Quantum threats (Grover's algorithm halves key search to 2^128→2^64 for AES-256) favor longer keys. Hardware acceleration (AES-NI instructions in Intel/AMD since 2010) boosts throughput to 10+ GB/s/core, ubiquitous in VPNs, disk (LUKS), TLS data phase.

Yet AES solves only confidentiality per block—the key distribution quandary persists, birthing public-key cryptography.

3. Asymmetric Encryption Methods: The Public Key Revolution

Symmetric encryption provides fast, efficient bulk data protection through shared secret keys, powering standards like AES via Feistel or SP networks that deliver confusion and diffusion across block modes. From DES's pioneering 56-bit keys—overcome by brute force—to AES's robust 128-256-bit security with hardware acceleration, these methods excel in performance but falter on secure key distribution over untrusted channels, reducing communication security to the key exchange problem.

This limitation spurred the public-key revolution: asymmetric cryptography introduces key pairs—public for encryption, private for decryption—enabling secure key agreement (e.g., Diffie-Hellman) and digital signatures without prior secrets, forming the hybrid backbone of modern protocols like TLS.

3.1. Introduction: A New Paradigm

Asymmetric cryptography, also known as public-key cryptography, elegantly solves the key distribution problem by abandoning the concept of shared secret. Instead of a single key, each participant has a mathematically linked **key pair**: a **public key and** a **private key**. The **"letterbox" analogy** illustrates how it works: the public key is like the address and slot of a letterbox—anyone can use it to insert (encrypt) a message. The private key is the matching key to the mailbox—only the owner can open it and read (decrypt) the messages.

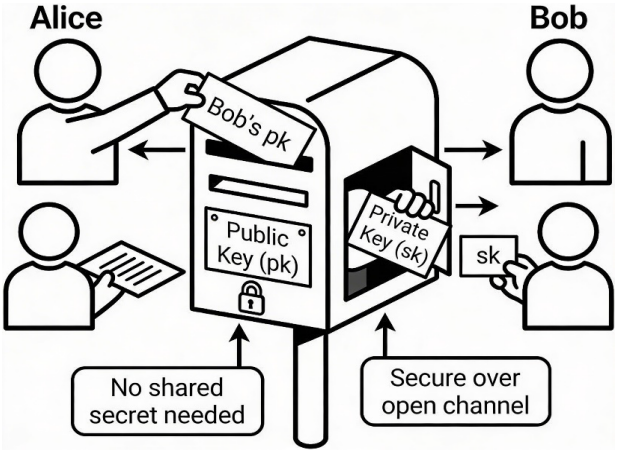


Figure 15. *Principle of asymmetric cryptography*

3.2. The Diffie-Hellman key exchange

The first published asymmetric method was the Diffie-Hellman key exchange in 1976 (Diffie & Hellman, 1976). It is not an encryption method in the true sense of the word, but a protocol that allows two parties (Alice and Bob) to agree on a common secret via an insecure channel intercepted by Eve.

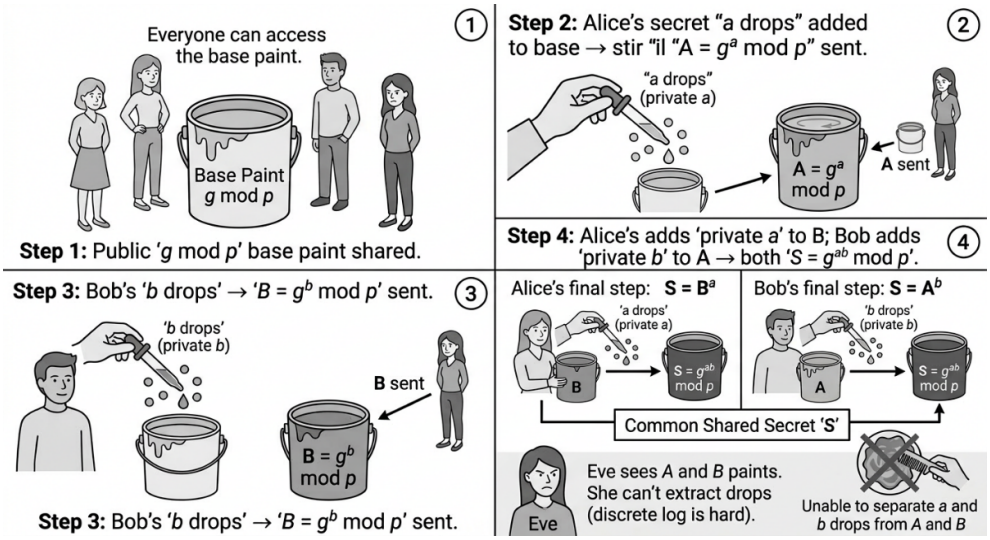


Figure 16. *Diffie-Hellman key exchange*

Procedure:

1. Alice and Bob publicly agree on a large prime number p and an associated basis g (a base g whose powers modulo p produce a wide range of values, making it difficult to calculate the discrete logarithm).
2. Alice chooses a secret random number a, calculates A = g^a mod p and sends A to Bob.
3. Bob chooses a secret random number b, calculates B = g^b mod p and sends B to Alice.
4. Alice calculates the common secret S = B^a mod p.
5. Bob calculates the common secret S = A^b mod p.

Both get the same value S = (g^b)^a mod p = (g^a)^b mod p = g^(ab) mod p. Eve, who only knows p, g, A and B, cannot easily calculate S because she does not know the secret numbers a and b. The certainty of the method is based on the difficulty of calculating the **discrete logarithm**: A can only be determined from A and g with extremely high computational effort.

3.3. The RSA algorithm

Developed in 1977 by Rivest, Shamir and Adleman (Rivest et al., 1977), the RSA algorithm is the most widely used asymmetric method to date and can be used for both encryption and digital signatures.

Mathematical foundations: The security of RSA is based on the practical difficulty of breaking down large numbers into their **prime factors**. While the multiplication of two large prime numbers is simple, the reverse operation (factorization) is extremely computationally intensive. This asymmetry forms a so-called **one-way trapdoor function**: encryption is simple (anyone can do it with the public key), but decryption (the trapdoor) is only possible efficiently with the knowledge of the private key(s) prime factors.

Key generation:

1. Choose two large, random prime numbers p and q.
2. Calculate the module n = p * q.
3. Calculate Euler's phi function φ(n) = (p - 1) * (q - 1).
4. Choose a public exponent e that is coprime to φ(n).
5. Compute the private exponent d as the multiplicative inverse of e modulo φ(n), such that e * d ≡ 1 mod φ(n). The **public key** is the pair (e, n), the **private key** is the pair (d, n).

Encryption and Decryption:

1. Encrypting a message M:C = M^e mod n
2. Deciphering a ciphertext C: M = C^d mod n

3.4. Digital signatures: authenticity and commitment

Asymmetric methods enable the creation of digital signatures (NIST, 2013), which are the digital equivalent of a handwritten signature and ensure authenticity and bindingness. At RSA, the use of the keys is simply "swapped" for this purpose:

Signature creation by the sender (Alice):

1. Alice creates a cryptographic **hash value** of the document to be signed. This hash serves as a unique digital fingerprint.
2. She "encrypts" (signs) this hash value with her **private key**. The result is the digital signature.

Signature verification by the recipient (Bob):

1. Bob receives the document and the signature.

2. He "decrypts" the signature with Alice's **public key** and thus obtains the original hash value that Alice calculated.
3. At the same time, Bob himself calculates the hash value of the received document.
4. If both hash values match, Bob knows that the document is authentic (really Alice's) and unchanged (integrity).

Since only Alice has her private key, only she can create a valid signature. This ensures **non-repudiation**, because Alice cannot later deny that she signed the document.

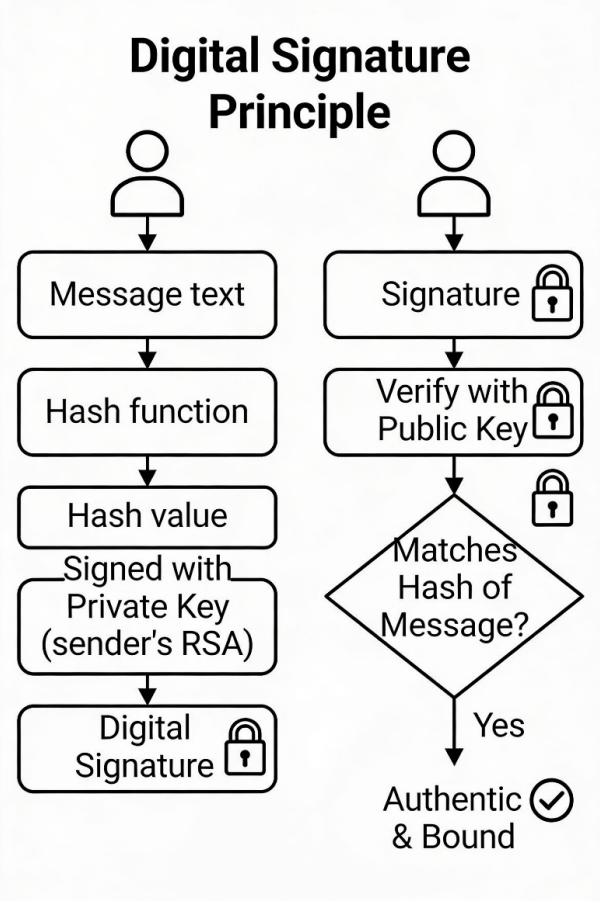


Figure 17. *The principle of digital signature*

4. Ensuring Data Integrity: Hash Functions and MACs

4.1. More than just secrecy

In addition to confidentiality, ensuring **data integrity** is a central protection goal. It must be ensured that data is not manipulated unnoticed on the way from sender to receiver or during storage. While confidentiality protects against "reading", integrity protects against "changing". The primary tool for ensuring integrity is cryptographic hash functions.

4.2. Cryptographic hash functions

A cryptographic hash function generates a fixed-length string of characters, called a hash value, or "digital fingerprint", from any amount of input data (such as a file or message).

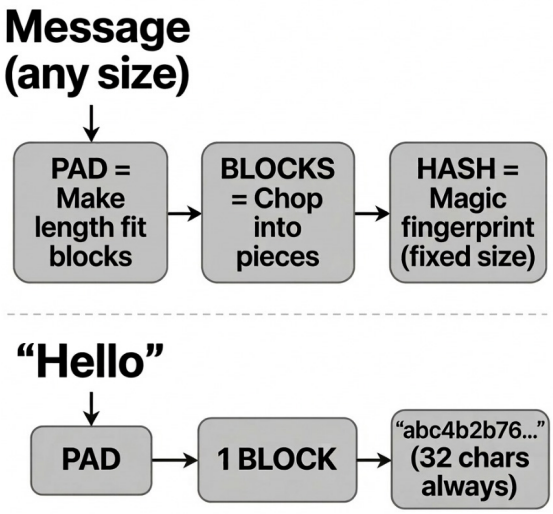


Figure 18. *Cryptographic hash function*

Properties: A secure hash function must meet the following properties:

1. **One-way function:** From a given hash value h(x), it is virtually impossible to reconstruct the original input x.
2. **Collision resistance:** It must be virtually impossible to find two different inputs x and y that produce the same hash value (h(x) = h(y)). These characteristics are summarized in three specific requirements:
    - **Preimage resistance:** For a given hash h, it is difficult to find an x with h(x) = h.
    - **Second Preimage Resistance:** For a given x, it is difficult to find an x' ≠ x with h(x') = h(x).
    - **Collision Resistance:** It is hard to find two arbitrary, different inputs x and x' with h(x') = h(x).

The Birthday Attack: Finding collisions is easier than finding an archetype due to the birthday paradox. While the latter requires an average of 2^n attempts (with a hash length of n bits), a birthday collision attack requires only about 2^(n/2) attempts. For this reason, hash values must be sufficiently long to withstand this attack.

Construction and standards: Many hash functions such as MD5 or SHA-1 are based on the **Merkle-Damgård construction**. Here, the message is divided into blocks and processed iteratively by a compression function.

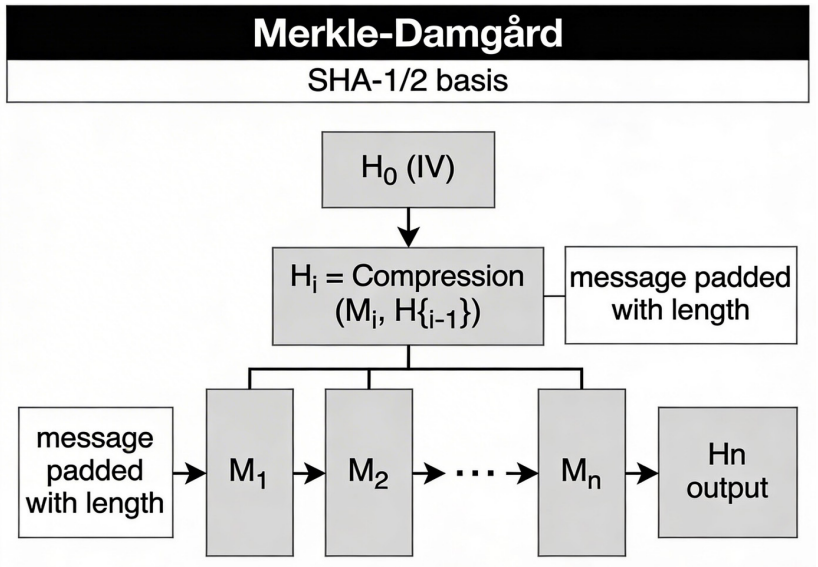


Figure 19. *Example of Merkle- Damgård construct*

Versions:

1. **MD5 (128 bit):** Has long been considered broken and unsafe because collisions can be deliberately generated.
2. **SHA-1 (160-bit):** Also considered insecure and should no longer be used for critical applications such as signatures.
3. **SHA-2 (SHA-256, SHA-512):** Currently considered safe and widely used (NIST, 2015).
4. **SHA-3 (Keccak):** The latest standard, which is based on a completely different internal structure (sponge construction) and is a safe alternative to SHA-2.

4.2.1 Evolution of Cryptographic Hash Functions

Cryptographic hash functions have evolved significantly since the 1990s, progressing from fast but insecure designs to modern standards resistant to all known practical attacks. This subsection examines MD5, SHA-1, SHA-2,

and SHA-3, highlighting their technical characteristics, security postures, and appropriate applications.

MD5 (Message-Digest Algorithm 5, 128-bit)

Technical Design: MD5 processes input in 512-bit blocks through four rounds of 16 operations each, using four 32-bit state registers (A,B,C,D) initialized to constant values. Each round applies bitwise functions (F,G,H,I) with modular addition, rotation, and table lookups.

Advantages:

- Extremely fast execution (optimized assembly implementations)
- Simple algorithm, minimal memory footprint
- Compact 32-character hexadecimal output
- Broad compatibility across all platforms

Applications (non-security only):

- File integrity verification in non-adversarial contexts
- Data deduplication and error detection
- Legacy system interoperability

Security Status: Fundamentally broken. Practical collision attacks demonstrated 2004 (Wang), allowing two arbitrary files with identical MD5 hashes. Birthday complexity reduced from theoretical 2^64 to 2^39 through differential cryptanalysis. NIST deprecated 1996; forbidden in all modern protocols.

SHA-1 (Secure Hash Algorithm 1, 160-bit)

Technical Design: Successor to MD5 with 80 rounds (5x16) and 160-bit state. Similar Merkle-Damgård construction but expanded message schedule and improved constants.

Advantages:

- Faster than SHA-2 family on older hardware
- Resisted practical attacks for 17 years post-publication
- Broad legacy ecosystem support

Applications (phasing out):

- Git repositories (transitioning to SHA-256)
- TLS certificates (Google Chrome blocked 2017)
- Code signing (Microsoft deprecated 2016)

Security Status: Practical collisions demonstrated 2017 (SHAttered attack: two colliding PDFs differing by 152 bytes). Google's coordinated disclosure confirmed real-world exploitability. NIST deprecated 2010; all major browsers/ certification authorities prohibit new SHA-1 certificates.

SHA-2 Family (SHA-224/256/384/512)

Technical Design: Same Merkle-Damgård construction as predecessors but with larger state sizes, improved constants, and expanded message schedules (64 rounds). SHA-256/512 optimized for 32/64-bit architectures respectively.

Table 3. *SHA 2 hash functions overview*

Algorithm	Output Size	Collision Security	Performance (MB/s, modern CPU)
SHA-224	224 bits	112 bits	850
SHA-256	256 bits	128 bits	900
SHA-384	384 bits	192 bits	650
SHA-512	512 bits	256 bits	1200 (64-bit optimized)

Advantages:

- No practical attacks after 20+ years of cryptanalysis
- Hardware acceleration (Intel SHA extensions since 2013)
- Multiple output sizes for diverse requirements
- Quantum-resistant (Grover reduces to 2^128 minimum)

Applications:

- Blockchain (Bitcoin double-SHA256 block headers)
- TLS 1.3 digital signatures/certificates
- File integrity (sha256sum utility)
- Password hashing (PBKDF2-HMAC-SHA256, NIST minimum 600k iterations)

Limitations: Theoretical Merkle-Damgård weaknesses (length extension attacks, Joux multicollisions). Slower than MD5/SHA-1 by design.

SHA-3 (Keccak, Sponge Construction)

Technical Design: First major departure from Merkle-Damgård using sponge construction. Alternates absorbing input into fixed state (1600 bits) with squeezing output. Five-step permutation (θ, ρ, π, χ, ι) applied per round.

Advantages:

- Provably secure sponge construction (no MD weaknesses)
- Algorithmic diversity from SHA-2 (hedge against unknown attacks)
- Configurable output sizes (SHAKE128/256 extendable)
- Excellent side-channel resistance
- Superior hardware efficiency (parallelizable rounds)

Applications:

- NIST FIPS 202 standard (2015)
- Post-quantum signatures (Dilithium uses SHAKE256)
- High-assurance systems (government classified)
- IoT/embedded (compact hardware implementations)

Limitations:

- 30-50% slower in software vs SHA-256
- Larger codebase complexity
- Limited adoption (SHA-2 ecosystem dominance)
- Migration Strategy and Recommendations

Modern systems should follow this priority matrix:

➢ **CRITICAL (Replace Immediately):**

- MD5 → SHA-256 (all uses)
- SHA-1 → SHA-256 (signatures, certificates)

➢ **PREFERRED (General Purpose):**

- SHA-256 (99% use cases)
- SHA-512 (64-bit platforms, maximum security)

➢ **FUTURE-PROOF (High Assurance):**

- SHA3-256/SHAKE256 (diversity)
- BLAKE3 (non-cryptographic speed)

Deployment Reality (2026): SHA-256 dominates 95%+ production systems. SHA-3 adoption growing in new protocols (10% TLS certificates). MD5/SHA-1 persist only in legacy unmaintained systems.

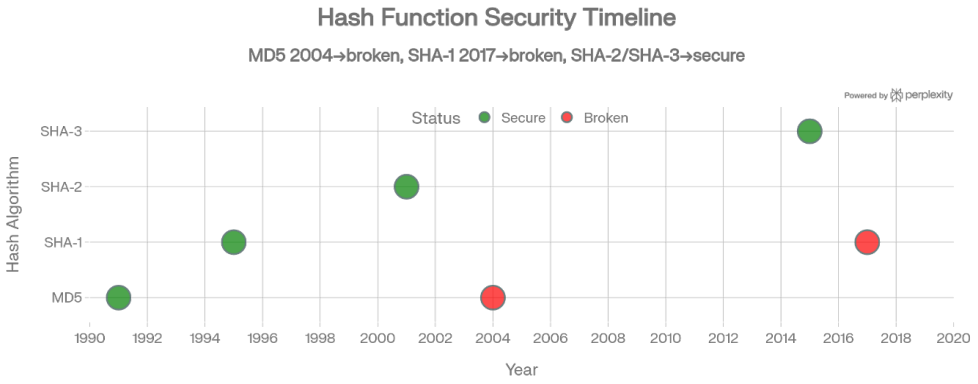


Figure 20. *Hash function security timeline*

This evolution demonstrates cryptographic design principles: larger output sizes alone insufficient (MD5→SHA-1), construction matters (SHA-2→SHA-3), and diversity provides resilience against unknown future attacks.

4.3. Message Authentication Codes (MACs)

A pure hash function only ensures integrity. Anyone can calculate a hash value for a message. To additionally ensure **authenticity**, i.e. to ensure who the message comes from, a **Message Authentication Code (MAC)** is used. A MAC is essentially a hash value that is calculated using a secret key known only to the sender and receiver. Only those who know the key can generate and verify a valid MAC. A widely used and secure construction is the **HMAC (Hash-based MAC),** which combines any hash function (e.g., SHA-256) with a secret key (Krawczyk et al., 1997).

4.4. Authenticated Encryption with GCM

In practice, confidentiality and integrity/authenticity are often needed together. Instead of using encryption and MAC separately, there are modern operating

modes that do both in one step. This concept is known as **Authenticated Encryption (AE).** A prominent representative is the **Galois/Counter Mode (GCM)** (NIST, 2007). GCM combines encryption in CTR mode (for confidentiality) with MAC calculation (for integrity and authenticity) into a single, high-performance and secure method. It is now a central part of modern security protocols such as TLS 1.2 and TLS 1.3 Rescorla (2018).

5. Applied cryptography: protocols and systems in practice

5.1. From building block to system

The cryptographic primitives discussed so far – symmetric and asymmetric ciphers, hash functions and MACs – are the fundamental building blocks of modern security. In practice, however, they are rarely used in isolation. Instead, they are combined into complex security protocols and systems to meet real-world security requirements in networks and applications. These systems orchestrate the various building blocks to ensure comprehensive security.

5.2. Secure transport routes: SSL/TLS

The SSL (Secure Sockets Layer) protocol and its successor TLS (Transport Layer Security) are the basis for secure communication on the Internet, most commonly known through HTTPS.

Purpose and architecture: TLS secures the communication channel between two applications (such as a web browser and a web server) at the transport layer. It is a **hybrid encryption system**:

1. **Asymmetric cryptography** is used during the **TLS handshake** to authenticate the server and securely exchange a session secret.
2. **Symmetric cryptography** is then used for the actual data transfer, as it is significantly faster.

The TLS handshake: This is the initial phase of a TLS connection where the security parameters are negotiated. The main steps are:

1. **Negotiating the algorithms:** The client sends a list of the cryptographic methods it supports (the **cipher suite**). The server selects a suitable suite.
2. **Server authentication:** The server proves its identity by sending its digital **certificate**. This certificate is issued by a trusted Certificate Authority (CA) and binds the server's public key to its domain.
3. **Key exchange:** Client and server use asymmetric methods (e.g. RSA or Diffie-Hellman) to agree on a common secret, the pre-master secret. From this, the symmetrical key material for the session is then derived.

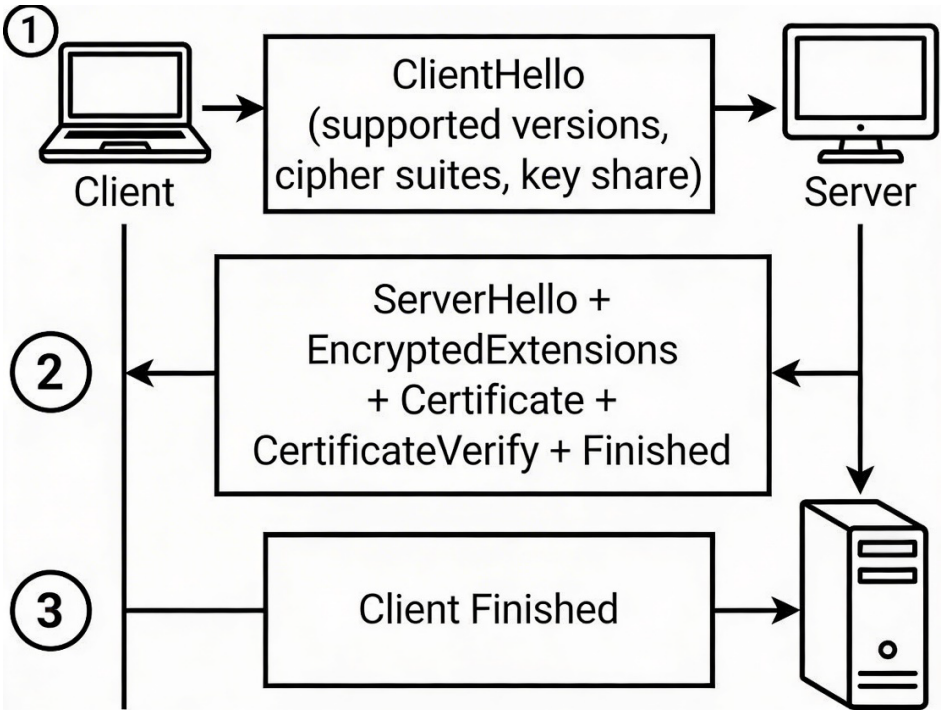


Figure 21. *Negotiation of Cypher Suites in TLS Handshake*

Attacks on TLS: Over the years, several vulnerabilities have been discovered in older TLS versions and their implementations:

Table 4. *Types of attacks on TLS*

Attack	Brief description of the vulnerability	Impact
Renegotiation-Attack	A loophole in the renegotiation of session parameters allowed an attacker to inject their own code at the beginning of a legitimate client request.	Man-in-the-middle attack that allows requests to be manipulated (plain text injection).
BEAST	Exploited a predictability of the IV in CBC mode of TLS 1.0 to decrypt encrypted data (e.g. cookies) byte by byte.	Decryption of sensitive data in an intercepted session (Chosen Plaintext Attack).
CRIME	Abused data compression in TLS. By analysing the change in length of encrypted data, conclusions could be drawn about secret content (e.g. cookies).	Decryption of secret tokens through repeated requests and length analysis.
POODLE	Forces a downgrade to the deprecated SSL 3.0 and exploits a vulnerability in the handling of padding in CBC mode (Padding Oracle).	Allows step-by-step decryption of encrypted data.

5.3. Network Authentication: Kerberos

Kerberos is a widely used authentication service for networks, which emerged as an evolution of the Needham-Schroeder protocol. It is based on symmetric cryptography and enables **single sign-on (SSO).** A user logs in once and can then access various network services without having to re-authenticate.

Core components: The system consists of three main actors:

1. **Client:** The user or machine that wants to access a service.

2. **Service:** The service (such as a file server) that is to be accessed.
3. **Key Distribution Center (KDC):** The trusted third party that knows all secret keys and coordinates authentication. The KDC consists of the Authentication Service (AS) and the Ticket Granting Service (TGS).

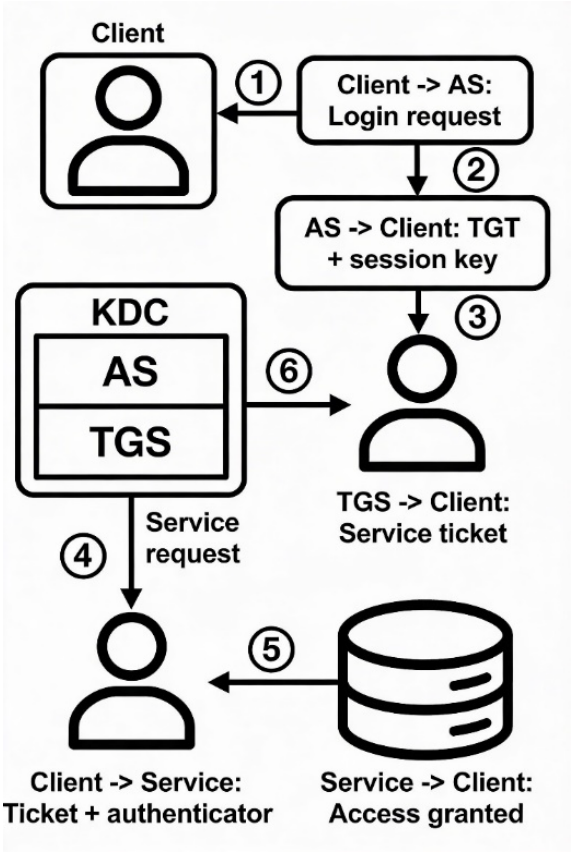


Figure 22. *Kerberos protocol flow*

Protocol flow: The authentication process is a three-step process:

1. **Request Ticket Granting Ticket (TGT):** The client authenticates with its password to the **Authentication Service (AS)** and receives a TGT. This ticket is more or less a ticket for further requests.

2. **Request Service Ticket:** With the TGT, the client contacts the **Ticket Granting Service (TGS)** and requests a specific service ticket for the destination service.
3. **Request service:** The client submits the service ticket to the actual **service**, which uses it to verify the client's identity and grant access.

Evaluation: A major advantage of Kerberos is its high **performance**, as it is based exclusively on fast symmetric cryptography. The biggest disadvantage is that the KDC is a **"single point of failure"** : If the KDC fails, authentication in the network is no longer possible.

5.4. Secure password storage

The fundamental tool for this purpose is a cryptographic hash function (Percival & Josefsson, 2016). When a user creates a password, the system computes a hash value and stores that value rather than the password itself. During login, the entered password is hashed again and compared with the stored hash. If both values match, the system assumes that the correct password was entered. This approach protects against direct disclosure, but by itself it is not sufficient, because attackers can still use precomputed tables or high-speed brute-force attacks against unsalted or weakly hashed passwords.

To make password storage more secure, each password must be combined with a unique random value called a salt. The salt is generated individually for every user and stored together with the password hash. Its purpose is to ensure that identical passwords do not result in identical hash values. This prevents attackers from recognizing shared passwords across accounts and significantly reduces the usefulness of rainbow tables, which are precomputed collections of password-hash mappings.

In addition to salting, secure password storage requires deliberately slow ***key-derivation functions***. General-purpose hash functions such as MD5 or SHA-1 are unsuitable for password storage because they are designed for speed. An attacker with modern hardware can test huge numbers of password guesses per second if such functions are used directly. For this reason, specialized password hashing methods such as PBKDF2, bcrypt, scrypt, and Argon2 are preferred. These methods intentionally increase computational effort and, in some cases, memory consumption, making large-scale guessing attacks much more expensive.

PBKDF2 extends a cryptographic hash function through repeated iteration. Instead of computing the hash only once, it applies the function many thousands or even hundreds of thousands of times. This slows down both legitimate authentication and attacks, but the effect is especially valuable against brute-force attempts. Bcrypt adds an adaptive work factor and was designed specifically for password storage. Scrypt and Argon2 go a step further by making attacks not only processor-intensive but also memory-intensive, which reduces the advantage of specialized hardware such as GPUs and ASICs.

Argon2 is generally regarded as the modern state of the art for password hashing. It was selected as the winner of the Password Hashing Competition and was designed to resist both parallel hardware attacks and trade-offs between memory and time. In practical systems, administrators can configure parameters such as runtime, memory usage, and degree of parallelism, allowing the password hashing process to be adapted to the security requirements and hardware capabilities of the system.

Secure password storage must also be combined with good operational practices. Passwords should be long enough to resist guessing, login attempts should be rate-limited, and multi-factor authentication should be used where possible. From a defensive perspective, the goal is not merely to hide the

password, but to make password recovery so expensive that large-scale attacks become impractical. In this way, password hashing represents an important application of cryptographic principles to everyday security engineering.

5.4.1. Threats to password databases

The compromise of a password database can have serious consequences, especially if the same password is reused across multiple services. Once attackers obtain a list of stored password hashes, they can perform offline attacks without interacting further with the target system. This is a dangerous scenario because the attack is no longer limited by login controls such as account lockouts or rate limiting. The attacker can test password guesses locally and at very high speed.

One common strategy is the **dictionary attack**, in which the attacker tries likely passwords drawn from real-world password lists, common words, keyboard patterns, and predictable variations. Because many users choose weak or familiar passwords, such attacks are often highly effective. Brute-force attacks extend this principle by systematically testing all possible combinations up to a certain length. The feasibility of these attacks depends heavily on the strength of the password and the cost of the hashing function used.

Rainbow tables represent another important attack concept. These are large precomputed tables that link passwords to their corresponding hash values. If a password hash appears in such a table, the attacker can recover the original password much faster than through ordinary brute force. Salting defeats this attack in practice because the same password no longer maps to a single predictable hash value across all users. Each unique salt forces the attacker to recompute guesses separately for every account, which drastically increases the workload.

For this reason, password security must be understood as an economic problem as much as a mathematical one. A strong password hashing scheme does not make attacks theoretically impossible, but it raises the cost in time, memory, and hardware so that the attack becomes unattractive or infeasible at scale. This is why secure password storage is one of the most practical and important uses of applied cryptography

5.5. Decentralized Trust: Blockchain and Bitcoin

Blockchain technology, made famous by cryptocurrencies such as Bitcoin, uses cryptographic concepts to create trust in a decentralized system without a central authority such as a bank.

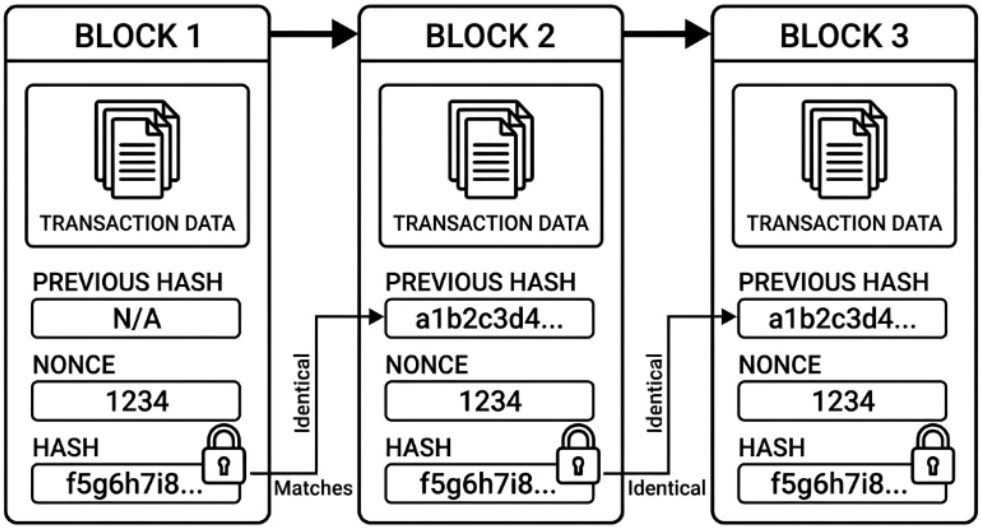


Figure 23. *Blockchain technology*

Blockchain technology became widely known through cryptocurrencies such as Bitcoin, but its deeper significance lies in the way it creates trust in a decentralized environment. In traditional financial systems, trust is usually

provided by a central institution such as a bank, which keeps an authoritative record of balances and transactions. In decentralized systems, however, there is no single trusted authority. The challenge is therefore to ensure that all participants agree on the same transaction history even though they do not necessarily trust one another.

The core problem in this context is the so-called double-spending problem. Since digital information can be copied easily, a malicious user could attempt to spend the same digital currency unit more than once. In a centralized model, a bank prevents this by checking every transaction against its own ledger. In a decentralized network, an alternative mechanism is needed to ensure that one consistent history of valid transactions is accepted by all participants.

The blockchain provides such a mechanism by functioning as a distributed public ledger. Transactions are grouped into blocks, and each block contains a cryptographic reference to the previous one in the form of a hash value. This creates a chronological chain in which every new block depends on the integrity of the previous blocks. If an attacker attempted to modify an older block, its hash would change, thereby invalidating all subsequent links in the chain. This cryptographic linking gives the ledger its tamper-evident character.

Within each block, transactions are commonly organized using a Merkle tree. In a Merkle tree, pairs of transaction hashes are repeatedly hashed together until a single root hash is obtained. This structure allows efficient verification that a transaction belongs to a block without requiring all other transactions in that block to be examined. Merkle trees therefore improve scalability and make blockchain systems more practical for distributed verification.

The integrity of the blockchain is reinforced by a consensus mechanism. In Bitcoin, this mechanism is called Proof of Work. To create a new block, miners must solve a computational puzzle by finding a nonce such that the resulting

hash value satisfies a predefined difficulty condition. Because this process requires significant computing effort, hardware resources, and electrical power, rewriting the transaction history becomes prohibitively expensive. Trust in the ledger is therefore not created by a central authority, but by the high cost of manipulating the network.

Mining also plays an economic role in the system. Miners collect pending transactions, assemble them into candidate blocks, and compete to find the valid Proof of Work. The miner who succeeds first may append the block to the chain and receives a reward. This reward typically consists of newly created currency units and the transaction fees included in the block. In this way, economic incentives are aligned with the maintenance and security of the network.

Blockchain systems show how cryptographic primitives such as hash functions, digital signatures, and tree structures can be combined with distributed consensus to create trust without central control. Their importance extends beyond cryptocurrencies, since the same principles can also be applied to timestamping, integrity assurance, traceability, and decentralized coordination. For this reason, blockchain can be understood as one of the most visible examples of applied cryptography in contemporary digital systems.

5.5.1. Merkle trees and efficient verification

A Merkle tree is a tree-structured arrangement of hash values that allows a large set of data elements to be summarized by a single compact value called the root hash. In blockchain systems, the leaves of the tree are usually the hashes of individual transactions. These leaf hashes are combined pairwise and hashed again, and the process continues upward until only one value remains. This final value, the Merkle root, is stored in the block header and serves as a fingerprint for the complete transaction set.

The practical advantage of this structure lies in efficient verification. If a participant wants to prove that a particular transaction belongs to a block, it is not necessary to transmit the entire list of transactions. Instead, only the transaction itself and the relatively small set of neighboring hashes along the path to the root are needed. By recomputing the path step by step, the verifier can check whether the resulting root matches the one recorded in the block header. This makes verification efficient even in large blocks.

Merkle trees are especially important for lightweight clients that do not store the full blockchain. Such clients can still verify inclusion proofs without downloading every transaction and every block in full. In this sense, the Merkle tree is a good example of how cryptographic design supports scalability: it preserves trust and integrity while reducing storage and bandwidth requirements.

5.5.2. Proof of Work and the cost of consensus

Proof of Work is a consensus mechanism that transforms computational effort into security. Its purpose is not simply to make block creation difficult, but to make dishonesty more expensive than honest participation. In Bitcoin, miners repeatedly vary a nonce in the block header until the resulting hash falls below a target value set by the network. Since cryptographic hash functions behave unpredictably, there is no shortcut to finding such a value other than repeated trial and error.

This design has an important security consequence. In order to alter a previously confirmed transaction, an attacker would have to recompute not only the modified block, but also all subsequent blocks, while simultaneously catching up with and overtaking the honest network. The larger the honest network's total computing power, the more unrealistic this attack becomes. This is why Proof of Work converts energy and hardware investment into a barrier against manipulation.

At the same time, Proof of Work has been criticized because of its high energy consumption. The process deliberately expends computational resources in order to secure the ledger, and this creates economic and environmental costs. For this reason, alternative consensus models such as Proof of Stake have been proposed and adopted in other blockchain systems. Nevertheless, Proof of Work remains historically important because it was the first practical mechanism to enable decentralized consensus at large scale without a central authority.

6. Emerging Directions in Cryptography

Cryptography continues to evolve in response to new computational models, new deployment environments, and new forms of digital risk. Classical goals such as confidentiality, integrity, authenticity, and non-repudiation remain central, but modern systems increasingly demand more than secure communication alone. Today, cryptography is also expected to support privacy-preserving computation, long-term resilience, distributed trust, and verifiable processing across heterogeneous and often untrusted infrastructures (Menezes et al., 1996; Katz & Lindell, 2020).

This shift has been accelerated by several technological developments. Quantum computing challenges the long-term security of widely used public-key algorithms, cloud computing raises questions about who can process sensitive data and under what assumptions, and artificial intelligence creates new privacy and accountability concerns around large-scale data use. In response, modern cryptographic research has expanded toward techniques that protect not only stored and transmitted information, but also the computation performed on that information (Shor, 1994; National Institute of Standards and Technology, 2024a).

This chapter introduces several of the most important emerging directions in contemporary cryptography. It begins with post-quantum cryptography, which addresses the threat of quantum attacks against current public-key systems. It then examines zero-knowledge proofs, homomorphic encryption, and secure multi-party computation, all of which enable new forms of privacy-preserving verification and collaborative processing. Finally, it discusses the growing importance of cryptography in cloud-native and AI-driven environments (Katz & Lindell, 2020).

6.1. Post-Quantum Cryptography

One of the most significant long-term challenges facing cryptography is the development of quantum computing. Many public-key cryptosystems used today, including RSA and elliptic-curve cryptography, rely on mathematical problems that are believed to be infeasible for classical computers to solve efficiently. However, a sufficiently powerful quantum computer could solve integer factorization and discrete logarithm problems in polynomial time using Shor's algorithm, thereby undermining the security assumptions behind many current protocols (Shor, 1994).

This threat does not affect all cryptographic primitives equally. Symmetric algorithms such as AES and cryptographic hash functions such as SHA-256 are more robust against quantum attacks, although they are not completely unaffected. Grover-type speedups imply that larger symmetric parameters, such as AES-256, are preferable for long-term protection in a post-quantum setting (Katz & Lindell, 2020).

Post-quantum cryptography refers to algorithms designed to remain secure against both classical and quantum adversaries while still running on ordinary classical hardware. These schemes are based on mathematical problems for which no efficient quantum algorithms are currently known. Major families include ***lattice-based cryptography***, ***code-based cryptography***, **multivariate cryptography, hash-based signatures,** and **isogeny-based constructions**. Among these, lattice-based approaches have gained particular importance because they combine strong theoretical foundations with comparatively practical efficiency (National Institute of Standards and Technology, 2024a).

A major milestone in this field was reached in 2024, when NIST published its first finalized post-quantum cryptography standards. These include FIPS 203 for ML-KEM, FIPS 204 for ML-DSA, and FIPS 205 for SLH-DSA (National

Institute of Standards and Technology, 2024a, 2024b, 2024c). These standards mark the transition of post-quantum cryptography from a primarily academic topic to a matter of operational planning and systems engineering (National Institute of Standards and Technology, 2024a).

The practical challenge now is migration. Existing software stacks, certificate infrastructures, embedded devices, hardware modules, and communication protocols were largely designed around RSA and elliptic-curve methods. Replacing these components requires more than simply swapping one algorithm for another. Post-quantum schemes often involve larger public keys, larger signatures, and different computational trade-offs, which can affect protocol design, storage, bandwidth, and performance (National Institute of Standards and Technology, 2024a; National Institute of Standards and Technology, 2024b).

For this reason, transitional strategies often rely on hybrid approaches. In a hybrid key exchange or hybrid signature design, a classical algorithm and a post-quantum algorithm are used together so that security does not depend entirely on one family of assumptions during the migration period. This approach is particularly relevant for infrastructures that must remain interoperable while preparing for long-term quantum resilience (National Institute of Standards and Technology, 2024a).

The urgency of post-quantum migration is also linked to the so-called “harvest now, decrypt later” threat. Sensitive encrypted data can be intercepted and stored today, even if it cannot yet be decrypted. If quantum-capable adversaries become available in the future, that historical ciphertext may become readable. This means that organizations responsible for long-lived confidential information must act before large-scale quantum computers actually exist (National Institute of Standards and Technology, 2024a).

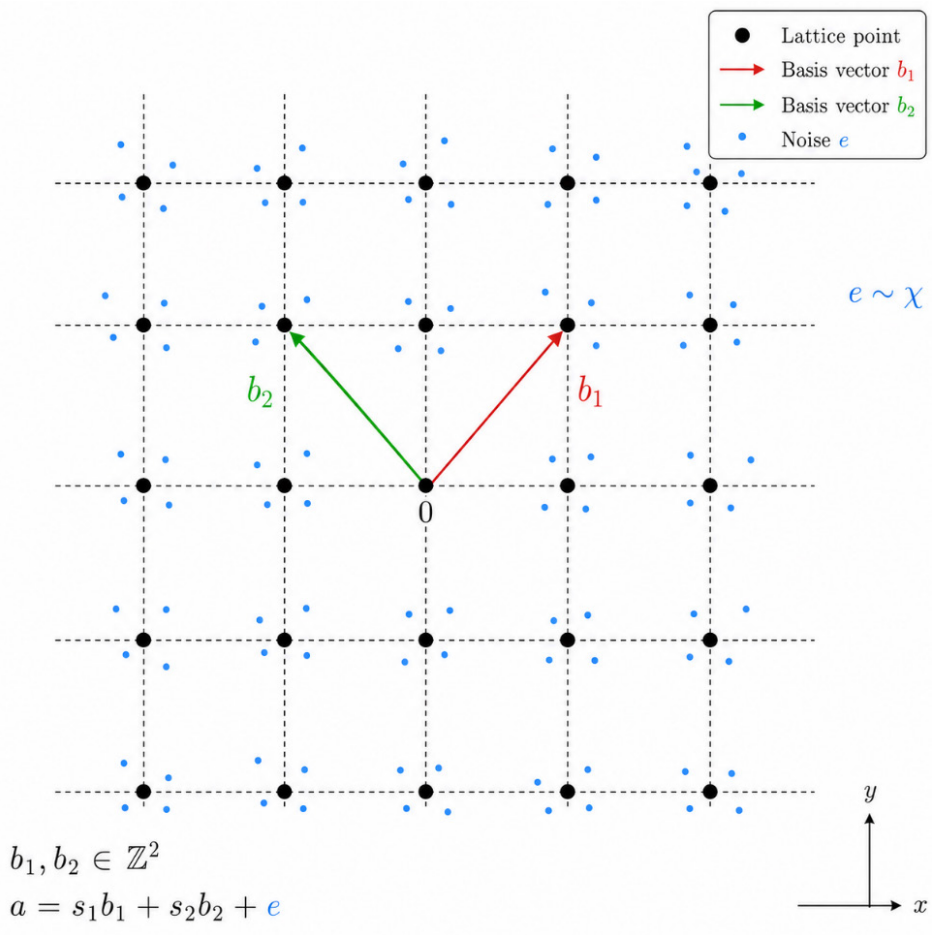


Figure 24. *Lattice-based intuition behind post-quantum cryptography*

6.1.1. Lattice-Based Cryptography

Lattice-based cryptography has emerged as the leading family in practical post-quantum cryptography. A lattice can be understood as a regular grid of points in multidimensional space generated by linear combinations of basis vectors. In cryptographic constructions, problems such as Learning With Errors and Module-LWE introduce small structured noise into algebraic relationships in a way that makes recovering the original secret computationally difficult (National Institute of Standards and Technology, 2024a).

The importance of lattice-based cryptography lies in its balance between security and practicality. Unlike some alternative post-quantum families, lattice-based schemes often support efficient key exchange and signatures while remaining suitable for software implementation. This is one reason why ML-KEM and ML-DSA, the first NIST-standardized post-quantum algorithms, are both based on lattice techniques (National Institute of Standards and Technology, 2024a, 2024b).

Another reason lattice-based cryptography is especially influential is that it also supports more advanced constructions beyond basic encryption and signatures. Many proposals for fully homomorphic encryption, attribute-based encryption, and advanced privacy-preserving systems rely on lattice assumptions. In this sense, lattice cryptography is not only a replacement for classical public-key methods, but also a broader platform for the next generation of cryptographic design (Acar et al., 2018).

6.1.2. Migration Challenges and Organizational Readiness

Moving to post-quantum cryptography is not simply a matter of updating a library or replacing a certificate. In real infrastructures, cryptographic algorithms are deeply embedded in protocols, key management systems, hardware security modules, VPN solutions, identity systems, firmware, and long-lived archives. A successful migration therefore requires inventory, prioritization, testing, and policy adaptation, not just algorithm selection (National Institute of Standards and Technology, 2024a).

Organizations must first identify where vulnerable public-key algorithms are currently used. This includes TLS certificates, VPN tunnels, code-signing workflows, email security, remote management, smart cards, and internal PKI systems. The next step is cryptographic agility, meaning the ability of

a system to replace or combine algorithms without fundamental redesign. Systems that lack cryptographic agility are especially vulnerable because they lock organizations into outdated security assumptions (National Institute of Standards and Technology, 2024a).

Migration also raises pedagogical and governance questions. Engineers need to understand new parameter sizes, new failure modes, and new implementation patterns. Decision makers need to balance interoperability, performance, and long-term security. For this reason, post-quantum migration is not only a technical issue but also a strategic and organizational challenge that will shape information security planning over the coming decade (National Institute of Standards and Technology, 2024a; National Institute of Standards and Technology, 2024b).

6.2. Zero-Knowledge Proofs

Zero-knowledge proofs are among the most conceptually striking developments in modern cryptography. In a zero-knowledge protocol, a prover convinces a verifier that a statement is true without revealing the secret information underlying that statement. The verifier learns only that the claim is valid, and no additional knowledge about the secret should be disclosed (Katz & Lindell, 2020).

This idea is powerful because it separates proof from disclosure. In classical settings, proving possession of knowledge often means revealing at least some part of that knowledge. Zero-knowledge techniques change this relationship. A user may prove possession of a password, a secret key, a credential, or a valid witness for a mathematical statement without disclosing the password, key, or witness itself (Katz & Lindell, 2020).

Zero-knowledge proofs are usually characterized by three core properties. First, they must be complete, meaning that an honest prover can convince an honest verifier. Second, they must be sound, meaning that a dishonest prover cannot successfully prove a false statement except with negligible probability. Third, they must be zero-knowledge, meaning that the verifier learns nothing beyond the truth of the claim (Katz & Lindell, 2020).

In early formulations, many zero-knowledge proofs were interactive, requiring several rounds of communication between prover and verifier. Later developments introduced non-interactive approaches in which a proof can be generated once and verified later without further exchange. This has made zero-knowledge techniques especially useful in distributed systems, where compact proofs of validity are needed without constant interaction (Ben-Sasson et al., 2018; Consensys, 2021).

A particularly important modern distinction is the one between zk-SNARKs and zk-STARKs. zk-SNARKs are succinct, non-interactive proofs that are compact and efficient to verify, but many constructions depend on a trusted setup phase. zk-STARKs, by contrast, emphasize transparency and scalability, avoiding trusted setup in many designs and offering strong scalability for large computations, though often with larger proof sizes (Ben-Sasson et al., 2018; Consensys, 2021).

These techniques have become especially visible in blockchain ecosystems. In public blockchains, zero-knowledge proofs can be used to show that a transaction follows the system rules without revealing sensitive transaction details, or to prove that a large set of off-chain computations was performed correctly before committing only a compact proof on-chain. This makes them valuable both for privacy and for scalability (Consensys, 2021).

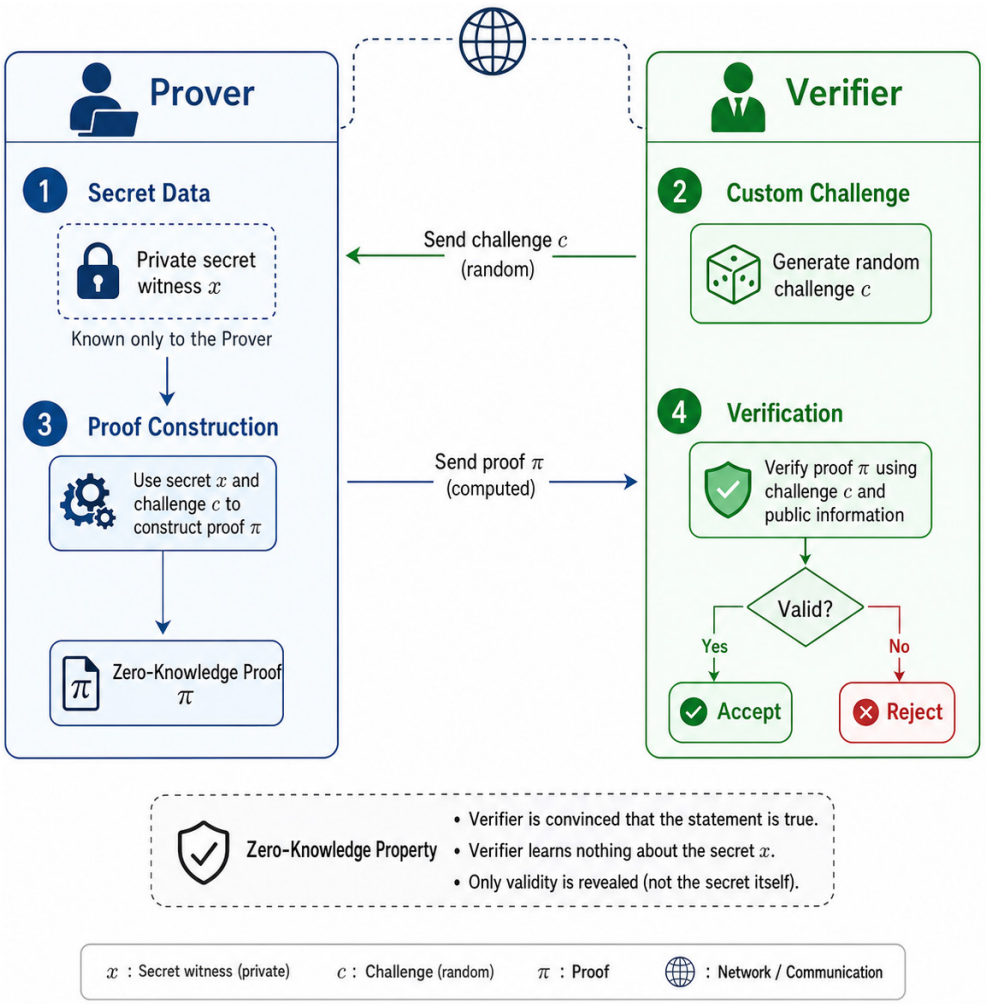


Figure 25. *Basic principle of a zero-knowledge proof between prover and verifier*

6.2.1. Applications of Zero-Knowledge Proofs

One important application area is digital identity. In traditional identity verification, users often reveal more information than is necessary. For example, proving legal age may involve sharing a full identification document containing name, address, birth date, and other details. A zero-knowledge approach can instead allow a

user to prove that they meet an age threshold without revealing the underlying date of birth or any unrelated personal information (Katz & Lindell, 2020).

Another application is privacy-preserving compliance. An organization may need to prove that certain computations, controls, or policy checks were performed correctly without exposing all of its internal data. Similarly, in electronic voting, zero-knowledge techniques can help prove that ballots were formed and counted correctly without disclosing individual votes. These examples demonstrate that zero-knowledge proofs are relevant far beyond cryptocurrency systems (Katz & Lindell, 2020; Consensys, 2021).

In the context of distributed computation, zero-knowledge proofs are also increasingly used as verification tools. Rather than repeating a costly computation, a verifier can check a proof that attests to the correctness of the result. This changes the economics of trust in distributed systems and makes zero-knowledge methods especially attractive for scalable architectures, outsourced computation, and privacy-preserving data services (Ben-Sasson et al., 2018).

6.3. Homomorphic Encryption

Traditional encryption protects data in storage and in transit, but not while it is being processed. In ordinary systems, data must usually be decrypted before meaningful computation can occur. This creates a major problem in outsourced processing environments such as cloud computing, because the service provider performing the computation may then gain access to sensitive plaintext. Homomorphic encryption addresses this challenge by enabling computation directly on encrypted data (Acar et al., 2018).

The core idea is that certain algebraic operations performed on ciphertexts correspond to meaningful operations on the underlying plaintexts. After decryption, the output is the same as if the computation had been applied directly to the original data. This means that a cloud provider can process encrypted information without learning the underlying sensitive values (Acar et al., 2018).

Homomorphic encryption exists in several forms. Partially homomorphic schemes support only one type of operation, such as addition or multiplication. Somewhat homomorphic schemes support a limited number of mixed operations, while fully homomorphic encryption supports arbitrary computations on encrypted data. The fully homomorphic case is the most powerful conceptually, because it enables general-purpose processing without decryption, but it is also the most computationally demanding (Acar et al., 2018).

The importance of homomorphic encryption lies in the way it changes the relationship between confidentiality and utility. Traditionally, systems had to choose between keeping data secret and making it computable. Homomorphic encryption shows that, at least in principle, both goals can coexist. This is especially relevant in fields such as healthcare, finance, government analytics, and privacy-preserving machine learning, where data is highly valuable but also highly sensitive (Acar et al., 2018).

At the same time, the method remains expensive. Ciphertexts are larger than plaintexts, operations are significantly slower, and implementation is complex. For this reason, homomorphic encryption is not yet a universal replacement for ordinary processing, but rather a specialized tool used where confidentiality during computation is more important than raw efficiency (Acar et al., 2018).

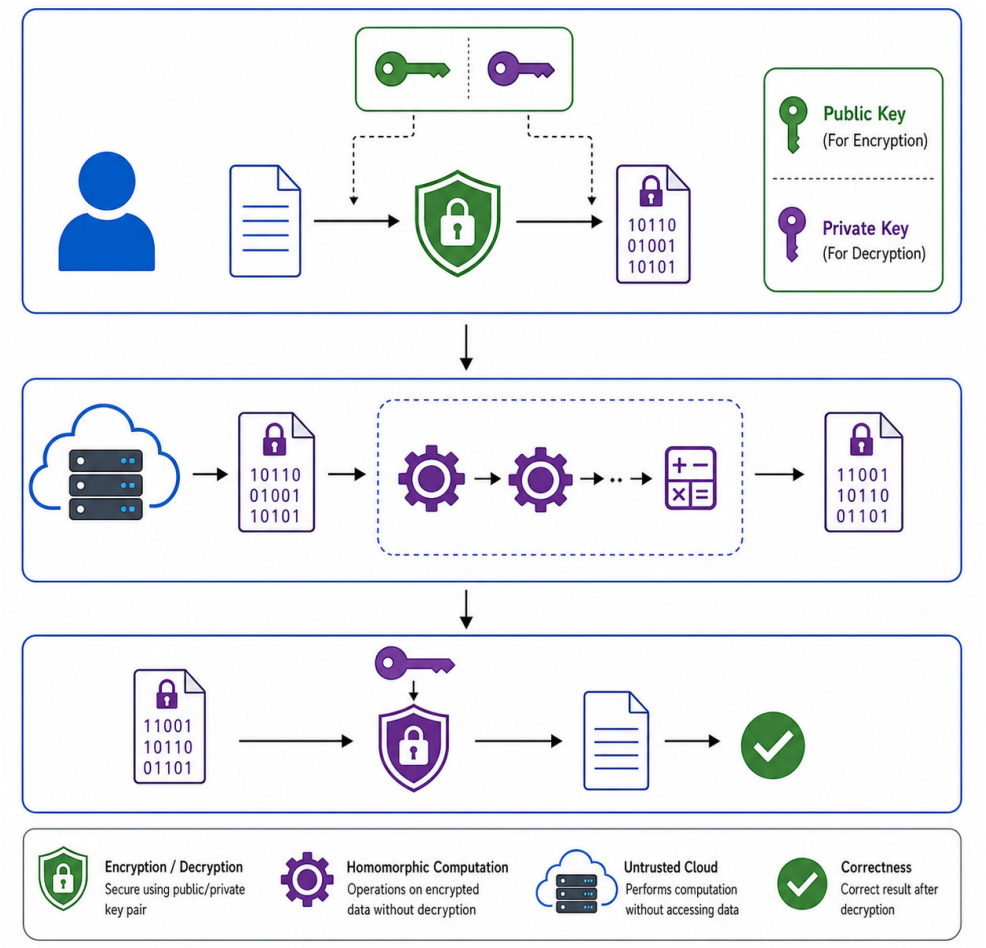


Figure 26. *Cloud computation on encrypted data using homomorphic encryption*

6.3.1. Privacy-Preserving Analytics and Machine Learning

One of the most promising uses of homomorphic encryption is privacy-preserving analytics. A data owner may encrypt a dataset and outsource computation to a cloud provider without exposing the raw data. The provider can perform selected operations on the encrypted inputs and return encrypted results, which only the data owner can decrypt. This model is attractive in sectors that must comply with strict legal or ethical obligations concerning data confidentiality (Acar et al., 2018).

A related area is privacy-preserving machine learning. Organizations increasingly wish to use external infrastructure for model training or inference, but sensitive training data and model inputs create confidentiality concerns. Homomorphic encryption can support limited forms of secure inference and, in some scenarios, parts of secure training workflows. Although the performance costs remain high, the conceptual significance is substantial because it enables useful AI services without requiring full exposure of private data (Acar et al., 2018).

6.4. Secure Multi-Party Computation

Secure Multi-Party Computation, often abbreviated as MPC, enables several parties to compute a joint function over their private inputs without revealing those inputs to one another. Instead of sending raw data to a central authority, each participant contributes encrypted shares or protocol messages, and the result is computed collaboratively according to cryptographic rules. At the end, the parties learn only the agreed output and no additional information about the other participants' private values (Evans et al., 2018).

This model is valuable in many practical settings. Several hospitals may wish to compute aggregate medical statistics without exposing individual patient data. Financial institutions may wish to identify suspicious patterns across organizations without revealing full transaction histories. Companies may want to collaborate on analytics while preserving trade secrets or complying with data protection obligations. In all of these cases, MPC offers a way to cooperate without centralized trust (Evans et al., 2018).

MPC protocols rely on a variety of techniques, including secret sharing, oblivious transfer, garbled circuits, and distributed validation. In secret-sharing approaches, a value is split into several shares distributed among participants so that no single share reveals the original secret. Computation is then performed over

these shares in a structured way, and only the final result is reconstructed. This distributes trust and reduces the need for any single fully trusted processor (Evans et al., 2018).

The major strength of MPC is that it allows confidentiality to be preserved even in collaborative analytical settings. Its main disadvantages are performance overhead, communication complexity, implementation difficulty, and the need for careful threat modeling. In comparison with ordinary centralized computation, MPC is often slower and more operationally demanding. However, where the legal, strategic, or ethical barriers to sharing plaintext data are high, these costs may be justified (Evans et al., 2018).

6.4.1. MPC as a Model of Distributed Trust

Beyond its direct technical applications, MPC is important because it illustrates a broader trend in cryptography: trust can be distributed rather than centralized. Traditional digital systems often rely on a single trusted operator, database, or service provider. MPC shows that protocol design can replace some of this trust by ensuring that no participant sees more than necessary and that correctness emerges from collaboration rather than unilateral control (Evans et al., 2018).

This perspective is increasingly relevant in a world where organizations must collaborate across institutional boundaries. Universities, hospitals, companies, and governments often need to combine information without fully surrendering control over their data. MPC therefore represents not only a cryptographic technique, but also a governance model for privacy-preserving digital cooperation (Evans et al., 2018).

6.5. Cryptography in AI and Cloud Systems

The rapid spread of cloud computing has transformed the operational environment of information security. Data is now frequently stored, processed, and analyzed outside the organization that originally collected it. While this model provides flexibility, elasticity, and cost efficiency, it also introduces new trust assumptions. Cloud customers must rely on third-party infrastructures while still preserving confidentiality, integrity, access control, and compliance. Cryptography is therefore central to cloud security not only in communication but also in storage, authentication, and key management (Katz & Lindell, 2020; Acar et al., 2018).

Artificial intelligence adds a further layer of complexity. AI systems depend on large volumes of training data and often process sensitive personal, financial, or organizational information. These systems also create new attack surfaces, such as model inversion, inference attacks, data leakage, model extraction, and untrusted training pipelines. Cryptographic methods can help mitigate some of these risks by supporting secure computation, privacy-preserving collaboration, and verifiable processing (Acar et al., 2018; Evans et al., 2018).

Several emerging cryptographic methods are especially relevant in this setting. Homomorphic encryption can enable computation on protected datasets. Secure multi-party computation can support collaborative training or analytics without raw data pooling. Zero-knowledge proofs can be used to attest to the correctness of some computational steps or policy properties without exposing the underlying sensitive data. Post-quantum methods are also important because cloud-stored information may need long-term confidentiality against future attackers (National Institute of Standards and Technology, 2024a; Acar et al., 2018; Evans et al., 2018).

This combination of requirements has made cryptographic agility increasingly important. Systems should be designed so that cryptographic mechanisms can be updated, combined, or replaced when new threats emerge or new standards appear. In fast-moving environments such as cloud-native software and AI platforms, inflexible cryptographic design quickly becomes a long-term liability. Agility therefore becomes a strategic design principle rather than an afterthought (National Institute of Standards and Technology, 2024a).

6.5.1. Trustworthy AI and Verifiable Computation

The rise of AI also raises a deeper question: how can users trust the outputs of systems whose internal operations are complex, opaque, or distributed? Cryptography cannot solve every aspect of this problem, but it can contribute important building blocks. Digital signatures ensure the integrity and provenance of models and updates. Zero-knowledge techniques can support selective verification of claims about computation. Secure logging and authenticated data structures can strengthen auditability and tamper evidence (National Institute of Standards and Technology, 2013; Katz & Lindell, 2020).

In this way, cryptography is becoming part of a broader architecture of trustworthy digital systems. Rather than serving only as a shield around data, it increasingly enables controlled sharing, accountable processing, selective disclosure, and resilient infrastructure design. This evolution reflects a major conceptual change: cryptography is no longer just about hiding information, but also about governing how information is used, proven, and trusted in complex digital ecosystems (Katz & Lindell, 2020; Evans et al., 2018).

6.6. Synthesis: From Protection to Trustworthy Computation

The emerging directions discussed in this chapter share a common theme. Classical cryptography focused primarily on secrecy and authentication between communicating parties. Modern cryptography still serves those purposes, but it has expanded into a framework for privacy-preserving verification, secure collaboration, long-term resilience, and trustworthy outsourced computation. Post-quantum cryptography protects future confidentiality, zero-knowledge proofs protect privacy during verification, homomorphic encryption protects data during computation, and secure multi-party computation protects inputs during collaboration (National Institute of Standards and Technology, 2024a; Acar et al., 2018; Evans et al., 2018).

These developments suggest that the future of cryptography lies not only in stronger algorithms, but also in richer models of trust. As quantum computing, cloud infrastructures, distributed ledgers, and AI systems continue to develop, cryptography will increasingly shape how digital systems remain secure, private, and accountable. For this reason, an understanding of emerging cryptographic methods is becoming essential not only for researchers, but also for practitioners designing the next generation of digital infrastructures (Katz & Lindell, 2020; National Institute of Standards and Technology, 2024a).

7. Cryptographic Engineering and Implementation Pitfalls: From Theory to Fragile Practice

7.1. Introduction: The Implementation Gap

Perfect cryptographic algorithms provide no security if deployed incorrectly. History shows that 80-90% of breaches trace back to misconfigurations, not broken mathematics—Heartbleed (Durumeric et al., 2017), POODLE (Google Security Team, 2014), and recent Cloud KMS leaks confirm this pattern. This chapter bridges theory and deployment, examining how even AES-256 or RSA-4096 fail spectacularly through predictable human and systemic errors. Understanding these pitfalls equips practitioners to audit, deploy, and maintain cryptographic systems securely.

7.2. Key Management Failures: The Weakest Link

Keys as the ultimate single point of failure. Despite unbreakable algorithms like AES-256, security collapses instantly when keys leak, repeat, or originate from weak sources. NIST SP 800-57 identifies key management as responsible for 70%+ of cryptographic failures in production systems – more critical than algorithm choice itself (NIST SP 800-57, 2020). This section dissects the predictable failure patterns that turn theoretical perfection into practical catastrophe.

Keys represent concentrated risk—compromise one master key, and all dependent data falls (NIST SP 800-57, 2020).

Common failures include:

- **Inadequate Randomness:** Weak entropy sources produce predictable keys. Dual_EC_DRBG (NSA backdoor, 2013) and Debian OpenSSL bug (2008) generated only 15 bits of entropy, reducing 1024-bit RSA keyspaces to brute-forceable 2^15 possibilities.
- **Hardcoded Secrets:** GitHub repositories routinely leak AWS access keys, API tokens, and TLS private keys. Static analysis finds 10,000+ secrets daily across public repos.
- **Improper Rotation**: Long-lived keys amplify damage. Target's 2013 breach exposed 40 million cards because POS terminals reused static DES keys for months.

Table 5. *Key Management Failure Patterns*

Failure Mode	Example	Consequence	Mitigation
Weak RNG	Dual_EC_DRBG	NSA-accessible keys	Use /dev/urandom, hardware RNG
Hardcoded	GitHub leaks	Full system compromise	Secret scanning, env variables
No Rotation	Target POS	40M cards exposed	Automated 90-day cycles
Key Reuse	TLS session reuse	Forward secrecy loss	Ephemeral keys (DHE/ECDHE)

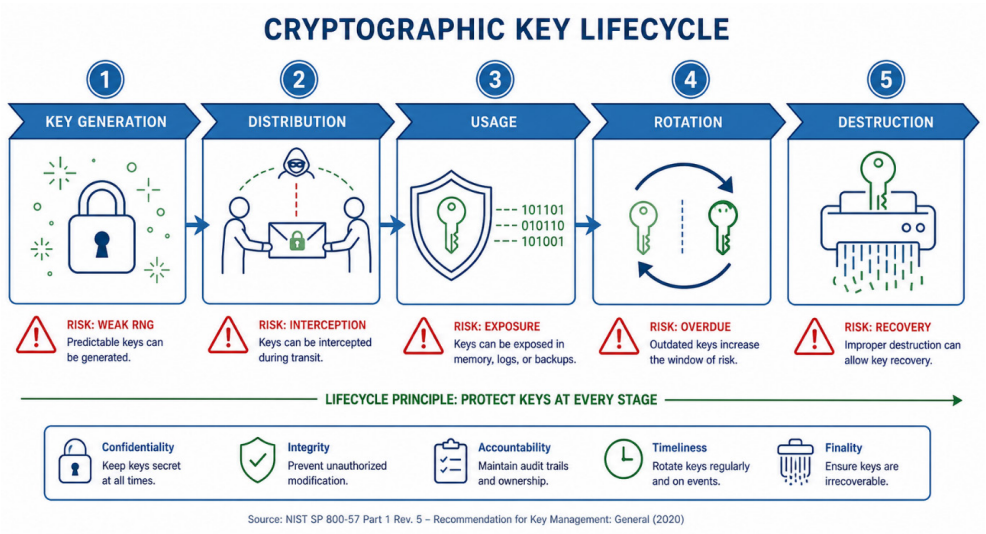


Figure 27. *Carthographic Key Lifecycle*

7.3. Algorithm and Mode Misuse

Secure primitives become liabilities through predictable misuse. Even cryptographers' gold standards fail spectacularly when deployed with outdated modes or ignored best practices. This section exposes how AES-256 in ECB or nonce-reused GCM – despite perfect mathematics – leak data through engineering shortcuts familiar to every attacker.

Even secure primitives fail through misuse:

ECB Mode: Identical plaintext blocks produce identical ciphertext blocks, leaking patterns. Penguin-in-ECB remains the canonical warning.

Nonce/IV Reuse: AES-GCM nonce collisions enable forgery. Cloudflare's 2019 incident decrypted customer HTTPS traffic due to 96-bit nonce exhaustion in 2^32 connections (Cloudflare, 2019).

Padding Oracle: TLS 1.0 CBC flaws (POODLE) exploited error messages revealing padding validity, enabling byte-at-a-time decryption (Google Security Team, 2014).

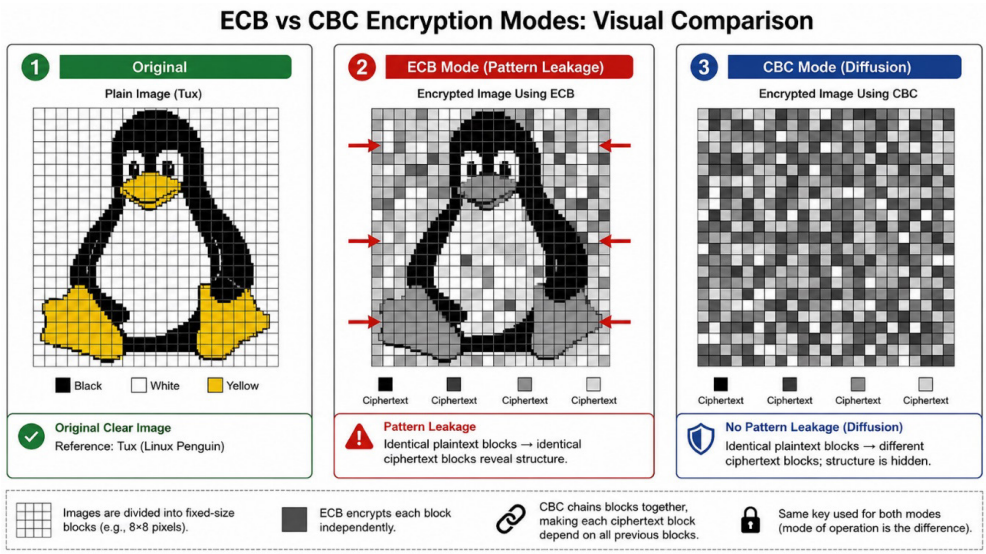


Figure 28. *ECB vs CBC Penguin*

7.4. Side-Channel and Physical Attacks

Mathematics secure against ciphertext analysis fail spectacularly against implementation physics. Timing variations, power spikes, and electromagnetic emissions betray secrets no black-box model predicts. Pioneered by Bernstein (2006), these attacks recover keys from hardware behavior, rendering even quantum-resistant algorithms vulnerable without physical countermeasures.

Cryptanalysis extends beyond ciphertext (Bernstein, 2006):

Timing Attacks: Variable-time comparisons leak information. Lucky Thirteen exploited TLS padding timing differences, recovering 80% of plaintext bytes (AlFardan & Paterson, 2013).

- **Power Analysis**: Differential Power Analysis (DPA) measures CPU power draw during AES S-box lookups, recovering keys in 2^16 traces (Biham & Shamir, 1997).

- **Cold Boot Attacks**: DRAM retains keys for minutes after power-off.
- **Weak Parameters**: NIST finds 70% of sites use <1000 PBKDF2 iterations—crackable on gaming GPUs.
- **No Salting**: Shared hashes enable rainbow table attacks.
- **User Behavior**: Password reuse across 100+ sites means one breach compromises all accounts.

Table 6. *Side-Channel Attack Taxonomy*

Attack Type	Target	Countermeasure	Difficulty
Timing	Conditional branches	Constant-time code	Medium
Power	S-box lookups	Masking, hiding	High
EM	Antenna emissions	Shielding	High
Fault Injection	Glitch circuits	Detection codes	Very High

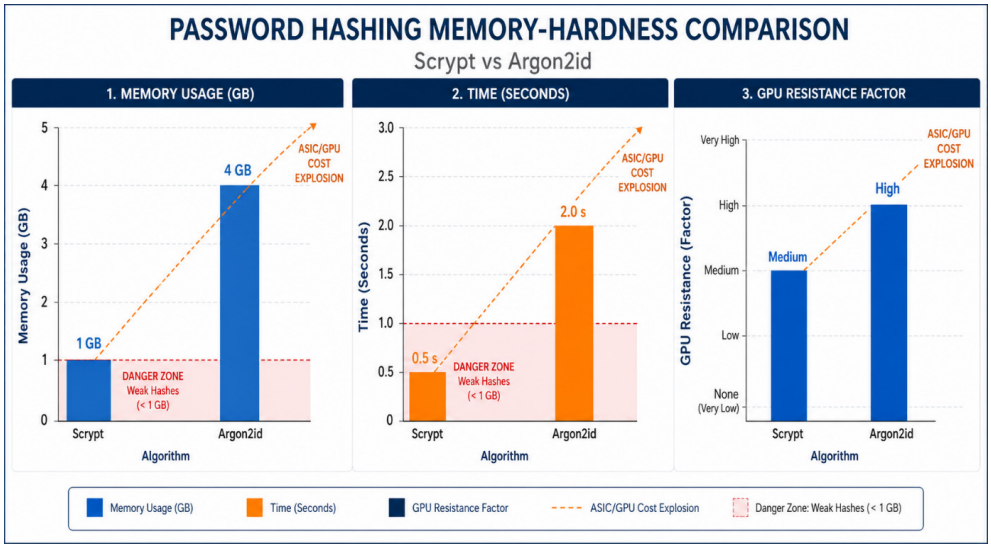


Figure 29. *Password Hashing Memory-Hardness Comparison (Placeholder: Scrypt vs Argon2 cost visualization).*

7.6. Crypto Agility and Supply Chain Risks

Future-proof systems fail through inflexible cryptography and untrusted components. Crypto agility – the ability to swap algorithms without system redesign – becomes mission-critical as NIST sunsets RSA-2048 (2030) and PQC standards mature. Supply chain attacks like XZ Utils (2024) prove even signed dependencies betray trust.

Modern systems must adapt rapidly:

Algorithm Sunset: SHA-1 certificates blocked 2020, yet 0.2% linger. NIST mandates RSA-2048 sunset by 2030.

Supply Chain: XZ Utils backdoor (2024) nearly compromised Linux SSH (Microsoft Security Response Center, 2024). SolarWinds (2020) injected malware into signed updates.

Quantum Migration: NIST PQC standards finalized 2024.

7.7. Best Practices and Audit Checklist

Systematic verification prevents 95% of deployment failures. Even perfect designs fail without disciplined execution. This actionable checklist – distilled from NIST SP 800-57 and ISO 27001 A.10 – transforms theory into production resilience.

CRYPTO AUDIT CHECKLIST

[X] Keys: Hardware RNG, 90-day rotation, HSM storage

[X] Modes: GCM/Chacha20-Poly1305, no ECB/CBC

[X] Timing: Constant-time implementations (libsodium)

[X] Passwords: Argon2id, 2^20+ iterations

[X] Certificates: OCSP stapling, must-staple, CAA records

[X] Agility: Dual crypto stack, automated key roll

[X] Monitoring: Failed auth rate, entropy levels

Closing Synthesis

Cryptography fails not through mathematical weakness, but engineering discipline. Heartbleed (2014) exposed 17% of HTTPS servers because memory safety was ignored (Durumeric et al., 2017). Practitioners must prioritize implementation hygiene over algorithm novelty.

Glossary

AES (Advanced Encryption Standard)
A symmetric block cipher standardized by NIST in 2001, selected from 15 candidates after public competition. Processes 128-bit blocks using 128/192/256-bit keys through 10/12/14 rounds of substitution-permutation operations. Widely used in TLS, disk encryption, and VPNs.

Argon2
Password hashing algorithm and winner of the 2015 Password Hashing Competition. Memory-hard design resists GPU/ASIC parallelization through configurable time, memory, and parallelism parameters. Recommended by OWASP and NIST SP 800-63B.

Birthday Attack
Cryptanalytic technique exploiting the birthday paradox to find hash collisions with ~2^(n/2) effort for n-bit hash. Critical limitation for MD5 (broken) and SHA-1. Requires 256+ bit hashes for modern security margins.

Block Cipher
Symmetric algorithm processing fixed-size data blocks (typically 128 bits). Examples: AES, DES. Requires chaining modes (CBC, GCM) for arbitrary-length messages. Provides confusion (S-boxes) and diffusion (permutations).

CBC (Cipher Block Chaining)
Block cipher mode XORing each plaintext block with previous ciphertext block before encryption. Requires random IV. Hides identical plaintext patterns but sequential-only and error-propagating.

Certificate
Digital document binding public key to entity identity via trusted CA signature. Contains subject name, public key, validity period, and extensions. X.509 format standard for TLS/PKI.

Collision Resistance
Hash property making it computationally infeasible to find two distinct inputs $x \neq y$ where $H(x) = H(y)$. Birthday attack reduces to ~2^(n/2) effort. SHA-256 provides 128-bit collision security.

Confusion
Shannon principle obscuring statistical relationship between plaintext/key and ciphertext. Achieved through nonlinear substitutions (S-boxes). Critical component of modern block ciphers.

Cryptographic Agility
The ability of a cryptographic system to easily replace or combine algorithms without fundamental redesign. Essential for post-quantum migration and adapting to new standards or threats (see Post-Quantum Cryptography

CTR (Counter Mode)
Block cipher mode generating keystream by encrypting incrementing counter values. Parallelizable, no error propagation, provable security when nonce unique. Basis for GCM authenticated encryption.

Decentralized Ledger
Distributed database achieving consensus without central authority. Blockchain primary example using cryptographic hash linking and proof-of-work validation.

DES (Data Encryption Standard)
64-bit Feistel block cipher standardized 1977. 56-bit key vulnerable to brute force (~1999). S-boxes resisted differential cryptanalysis. Predecessor to 3DES and AES.

Diffie-Hellman
1976 key exchange protocol enabling shared secret over insecure channel. Security based on discrete logarithm problem: compute g^(ab) mod p without knowing a or b. Foundation of TLS key agreement.

Diffusion
Shannon principle spreading influence of single plaintext bit across many ciphertext bits. Achieved through permutations and mixing operations. Complements confusion principle.

Digital Signature
Cryptographic primitive providing authenticity, integrity, and non-repudiation. Hash message, encrypt with private key. Verify by decrypting with public key and matching against recomputed hash.

Double-Spending
Cryptocurrency attack spending same digital unit multiple times. Solved by blockchain timestamp ordering and longest-chain consensus rule.

ECB (Electronic Codebook)
Simplest block cipher mode encrypting each block independently. Reveals identical plaintext patterns (image contours visible). Cryptographically unsafe.

Enigma
WWII German rotor cipher machine using plugboard, 3-4 rotating wheels, and reflector. ~10^23 daily settings. Broken via Polish mathematical analysis and British Bombe exploiting cribs.

Feistel Network
Balanced cipher construction processing data through multiple iterative rounds. $L_{i+1} = R_i$, $R_{i+1} = L_i \oplus f(R_i, K_i)$. Symmetric encryption/decryption via key schedule reversal.

Frequency Analysis
Cryptanalytic technique exploiting known language letter frequencies. Al-Kindi (9th century) first systematic application breaking monoalphabetic substitution ciphers.

Fully Homomorphic Encryption (FHE)
Encryption scheme allowing arbitrary computations on encrypted data without decryption. Results decrypt to correct plaintext computation outcomes. Computationally expensive but enables privacy-preserving cloud analytics (see Homomorphic Encryption).

GCM (Galois Counter Mode)
Authenticated encryption mode combining CTR keystream generation with GHASH integrity tag. Parallelizable, single-pass. TLS 1.2/1.3 default cipher suite.

Grover's Algorithm
Quantum algorithm providing quadratic speedup for unstructured search problems. Reduces symmetric key security from n bits to n/2 bits, making AES-256 preferable over AES-128 in post-quantum settings (see Post-Quantum Cryptography).

Harvest Now, Decrypt Later
Attack strategy where encrypted data is collected today for future decryption using quantum computers. Creates urgency for post-quantum migration for long-lived confidential data.

Hash Function
One-way function mapping arbitrary input to fixed-size digest. Cryptographic hashes resist preimage, second-preimage, and collision attacks. Used for integrity, signatures, blockchain.

HMAC (Hash-based MAC)
Message authentication code constructing hash from secret key + message using two nested hashes. Standardized construction provably secure for any secure hash function.

Homomorphic Encryption
Encryption supporting computation on ciphertexts where decrypted results match plaintext computation. Enables cloud processing of sensitive data without exposing plaintext (see Fully Homomorphic Encryption).

Hybrid Cryptography
Combination of classical and post-quantum algorithms during migration. Uses both RSA/EC + ML-KEM/ML-DSA simultaneously to maintain interoperability while building quantum resilience (see Post-Quantum Cryptography).

Initialization Vector (IV)
Random nonce ensuring identical plaintext encrypts differently each time. Critical for CBC, GCM, CTR modes preventing pattern leakage and replay attacks.

Lattice-Based Cryptography
Post-quantum cryptography family using high-dimensional geometric lattices. NIST standards ML-KEM (FIPS 203) and ML-DSA (FIPS 204) are lattice-based. Supports advanced constructions like FHE (see Post-Quantum Cryptography).

Learning With Errors (LWE)
Lattice-based hard problem where small structured noise is added to linear equations. Basis for ML-KEM (FIPS 203) and ML-DSA (FIPS 204). Believed quantum-resistant (see Lattice-Based Cryptography).

Kerberos
Symmetric-key network authentication protocol using trusted KDC. Issues Ticket Granting Tickets (TGT) enabling single sign-on across realm services.

Key Derivation Function (KDF)
Slow, iterated hash construction for password-to-key conversion. PBKDF2, bcrypt, scrypt, Argon2. Prevents GPU-accelerated offline dictionary attacks.

ML-DSA (FIPS 204)
NIST-standardized lattice-based digital signature algorithm (CRYSTALS-Dilithium). First post-quantum signature standard published 2024 (see Post-Quantum Cryptography).

Merkle Tree
Binary tree of pairwise hashes enabling O(log n) membership proofs. Blockchain transaction root allows light clients to verify inclusion without full data.

ML-KEM (FIPS 203)
NIST-standardized lattice-based key encapsulation mechanism (CRYSTALS-Kyber). First post-quantum KEM standard published 2024 (see Post-Quantum Cryptography).

Module-LWE
Ring variant of Learning With Errors using structured modules for efficiency. Used in NIST-standardized ML-KEM/ML-DSA algorithms (see Lattice-Based Cryptography).

Nonce
Number used once preventing replay attacks. Counter (CTR mode), random (CBC), or hybrid (GCM). Must never repeat with same key.

Non-Interactive Zero-Knowledge Proof
Zero-knowledge proof requiring no back-and-forth communication after proof generation. Proof can be verified later by anyone. Basis for zk-SNARKs/zk-STARKs (see Zero-Knowledge Proof).

Non-repudiation
Property preventing signer denying signature creation. Digital signatures using private key provide mathematical proof of authorship.

One-Time Pad (OTP)
XOR plaintext with truly random key ≥ message length, used once. Information-theoretically secure if conditions strictly met. Impractical key distribution.

PBKDF2
Password-Based Key Derivation Function 2. Iterates HMAC thousands of times. NIST SP 800-63B minimum 600,000 iterations SHA-256.

PKI (Public Key Infrastructure)
Framework managing digital certificates and public keys. CA hierarchy establishes chain of trust from root to end-entity certificates.

Polyalphabetic Cipher
Encryption using multiple substitution alphabets. Alberti disk, Vigenère square. Defeats frequency analysis by flattening letter distribution.

Post-Quantum Cryptography
Cryptographic algorithms secure against quantum adversaries while running on classical hardware. NIST FIPS 203/204/205 published 2024: ML-KEM, ML-DSA, SLH-DSA (see Lattice-Based Cryptography).

Preimage Resistance
Given hash h, infeasible to find x where H(x) = h. Foundation of hash security for password storage and digital signatures.

Privacy-Preserving Computation
Cryptographic techniques enabling useful computation on sensitive data without exposing plaintext. Includes homomorphic encryption, MPC, zero-knowledge proofs (see respective entries).

Proof of Work (PoW)
Consensus mechanism requiring computational puzzles for block inclusion. Bitcoin double-spend protection via economic majority hash power.

Prover
Party generating zero-knowledge proof to convince verifier of statement validity without revealing underlying secret (see Zero-Knowledge Proof, Verifier).

Public Key Cryptography
Asymmetric paradigm using key pairs. Public encrypts/sign verifies; private decrypts/signs. Solves symmetric key distribution problem.

Quantum Computing Threat
Grover's algorithm halves symmetric key search (AES-128 → 64-bit security). Shor's algorithm breaks RSA/EC factoring/discrete log. NIST PQC standardization underway.

Quantum-Safe Cryptography
Synonym for post-quantum cryptography. Algorithms resistant to Shor's and Grover's algorithms (see Post-Quantum Cryptography).

Rainbow Table
Time-memory tradeoff attack storing password-hash chains. Salted hashes force recomputation per user defeating precomputation advantage.

RSA
Rivest-Shamir-Adleman public-key algorithm. Security from integer factorization difficulty. n = pq, encrypt m^e mod n, decrypt c^d mod n.

Salt
Random value stored with password hash ensuring identical passwords produce different hashes. Defeats rainbow tables and duplicate detection.

S-box (Substitution Box)
Nonlinear mapping providing confusion. Core component of DES, AES, preventing linear cryptanalysis. Carefully designed against algebraic attacks.

Second Preimage Resistance
Given x, infeasible to find x' ≠ x where H(x) = H(x'). Stronger than collision resistance. Critical for digital signatures.

Secure Multi-arty Computation (MPC)
Protocol enabling multiple parties to compute joint function over private inputs without revealing inputs to each other. Uses secret sharing, garbled circuits (see Secret Sharing).

Secret Sharing
Cryptographic technique splitting secret into multiple shares where only authorized combinations reconstruct the secret. Foundation for MPC and threshold cryptography (see Secure Multi-Party Computation).

SHA-256
Secure Hash Algorithm 256-bit output. NIST standard post-SHA-1 collision breaks. Bitcoin blockchain foundation.

SLH-DSA (FIPS 205)
NIST-standardized stateless hash-based digital signature. Alternative to lattice-based signatures (see Post-Quantum Cryptography).

Shor's Algorithm
Quantum algorithm solving integer factorization and discrete logarithm in polynomial time. Breaks RSA, ECC, DH. Primary motivation for post-quantum cryptography (see Post-Quantum Cryptography).

Skytale
Spartan transposition cipher wrapping leather strip around baton. Readable only with matching diameter. Earliest systematic cipher (~500 BC).

Stream Cipher
Symmetric algorithm encrypting data stream bit/byte-wise via keystream XOR. RC4 (broken), ChaCha20/Salsa20 (secure). Real-time communications.

Succinct Non-Interactive Argument of Knowledge (zk-SNARK)
Compact zero-knowledge proof system. Efficient verification but often requires trusted setup (see zk-STARK).

Succinct Transparent Argument of Knowledge (zk-STARK)
Zero-knowledge proof avoiding trusted setup. Larger proofs but post-quantum secure and scalable (see zk-SNARK).

Substitution Cipher
Plaintext letters/symbols replaced by others. Monoalphabetic (Caesar), polyalphabetic (Vigenère). Vulnerable to frequency analysis.

Symmetric Cryptography
Encryption/decryption using identical secret key. Fast bulk encryption (AES). Key distribution bottleneck solved by asymmetric methods.

TLS (Transport Layer Security)
Successor to SSL providing confidentiality, integrity, authentication for network communications. Hybrid handshake + symmetric data transfer.

Transposition Cipher
Plaintext positions rearranged, characters unchanged. Rail fence, columnar, Skytale. Provides diffusion complementing substitution.

UTXO (Unspent Transaction Output)
Bitcoin accounting model tracking spendable outputs. Double-spending prevented by requiring valid chain of spending transactions.

Verifier
Party checking zero-knowledge proof validity. Learns only statement truth, nothing about underlying secret (see Zero-Knowledge Proof, Prover).

Vigenère Cipher
Polyalphabetic cipher using keyword repeating over plaintext. Each letter shifted by corresponding keyword letter position. "Le Chiffre Indéchiffrable" until Kasiski (1863).

Zero-Knowledge Proof
Cryptographic protocol where prover convinces verifier of statement truth without revealing additional information. Complete, sound, zero-knowledge properties (see Prover, Verifier, zk-SNARK, zk-STARK).

Tables and Figures

Tables

Figures

Bibliography

1. Katz, J., & Lindell, Y. (2020). *Introduction to modern cryptography* (Revised 3rd ed.). CRC Press.

2. Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of applied cryptography*. CRC Press.

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

4. National Institute of Standards and Technology. (2007). *Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC* (NIST Special Publication 800-38D). https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf

5. Krawczyk, H., Bellare, M., & Canetti, R. (1997). *HMAC: Keyed-hashing for message authentication* (RFC 2104). RFC Editor. https://www.rfc-editor.org/rfc/rfc2104.html

6. Rescorla, E. (2018). *The Transport Layer Security (TLS) Protocol Version 1.3* (RFC 8446). RFC Editor. https://www.ietf.org/rfc/rfc8446.txt

7. Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM, 21*(2), 120–126.

8. Diffie, W., & Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory, 22*(6), 644–654.

9. Shannon, C. E. (1949). Communication theory of secrecy systems. *The Bell System Technical Journal, 28*(4), 656–715.

10. National Institute of Standards and Technology. (2015). *Secure Hash Standard (SHS)* (FIPS PUB 180-4). https://doi.org/10.6028/NIST.FIPS.180-4

11. National Institute of Standards and Technology. (2013). *Digital Signature Standard (DSS)* (FIPS PUB 186-4). https://doi.org/10.6028/NIST.FIPS.186-4

12. Percival, C., & Josefsson, S. (2016). *The scrypt password-based key derivation function* (RFC 7914). RFC Editor. https://www.rfc-editor.org/rfc/rfc7914.html

13. National Institute of Standards and Technology. (2024). Module-Lattice-Based Key-Encapsulation Mechanism Standard (FIPS PUB 203). <https://doi.org/10.6028/NIST.FIPS.203>.

14. National Institute of Standards and Technology. (2024). Module-Lattice-Based Digital Signature Standard (FIPS PUB 204). <https://doi.org/10.6028/NIST.FIPS.204>.

15. National Institute of Standards and Technology. (2024). Stateless Hash-Based Digital Signature Standard (FIPS PUB 205). <https://doi.org/10.6028/NIST.FIPS.205>.

16. Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. Proceedings of the 35th Annual Symposium on Foundations of Computer Science, 124–134.

17. Ben-Sasson, E., Bentov, I., Horesh, Y., & Riabzev, M. (2018). Scalable, transparent, and post-quantum secure computational integrity. IACR ePrint Archive.

18. Acar, A., Aksu, H., Uluagac, A. S., & Conti, M. (2018). A survey on homomorphic encryption schemes: Theory and implementation. ACM Computing Surveys, 51(4), 1–35.

19. Evans, D., Kolesnikov, V., & Rosulek, M. (2018). A pragmatic introduction to secure multi-party computation. Foundations and Trends in Privacy and Security, 2(2–3), 70–246.

20. Consensys. (2021). Zero-Knowledge Proofs: STARKs vs SNARKs. <https://consensys.io/blog/zero-knowledge-proofs-starks-vs-snarks>.

21. AlFardan, N. J., & Paterson, K. G. (2013). Lucky Thirteen: Breaking the TLS and DTLS Record Protocols.

22. Bernstein, D. J. (2006). Timing Attacks on Cryptosystems.

23. Biham, E., & Shamir, A. (1997). Differential Fault Analysis.

24. Cloudflare (2019). "Nonce Misuse in Practice".

25. Durumeric, Z., et al. (2017). Neither fire nor ice begets breath: How we crushed OpenSSL Heartbleed.

26. Google Security Team (2014). This POODLE bites: Exploiting the SSL 3.0 fallback.

27. Microsoft Security Response Center (2024). XZ Utils backdoor.

28. NIST (2014). CVE-2014-0160: OpenSSL Heartbleed.

29. NIST (2014). CVE-2014-3566: POODLE SSLv3 vulnerability.

30. NIST SP 800-57 (2020). Recommendation for Key Management.

Closing remarks

Cryptography has evolved from secret-keeping practices spanning millennia to a mathematically rigorous discipline essential for digital trust. This lecture script synthesizes foundational theory with contemporary applications, bridging ancient ciphers (Caesar, Vigenère) and modern algorithms (AES, RSA) to equip readers for real-world deployment in cloud environments, secure protocols, and decentralized systems.

Key Takeaways

1. **Symmetric encryption** (AES, DES) provides high-performance bulk data protection but requires secure key distribution—a problem solved by asymmetric cryptography.

2. **Asymmetric methods** (RSA, Diffie-Hellman) enable secure key exchange and digital signatures, forming the backbone of TLS and modern internet security.

3. **Integrity and authenticity** mechanisms (hash functions, MACs, GCM) complement confidentiality, delivering comprehensive protection against passive and active attacks.

4. **Practical protocols** (TLS/HTTPS, Kerberos, blockchain) orchestrate cryptographic primitives into secure systems, requiring careful attention to implementation details, key management, and evolving threat landscapes.

Future Directions

Post-quantum cryptography, homomorphic encryption, and privacy-preserving cryptographic techniques (zero-knowledge proofs, secure multiparty computation) represent emerging frontiers. As quantum computing advances,

transitional strategies leveraging hybrid approaches will become critical for organizations managing long-term data security.

Acknowledgments

This manuscript builds upon foundational works by Katz & Lindell, Menezes et al., and standards from NIST and the IETF. The pedagogical approach reflects experience teaching cloud computing and cryptography at CAMPUS 02.

About the author

Dr. Selver Softic, a university instructor and researcher s at CAMPUS 02, University of Applied Sciences in Graz, Austria.

Professional Background

Dr. Softic holds a doctoral degree in computer science. His research centers on human-centric digital transformation, information security architectures, and usage of AI systems in business and education. He has extensive experience designing and delivering courses in cloud computing, data science, and IT infrastructure for academic and professional audiences.

Expertise and Research Interests

Dr. Softic regularly participates in international academic conferences, EU mobility programs, and collaborative research initiatives. His work bridges computer science, ethics, and practical security deployment in increasingly digitized organizational landscapes.

ISBN 978-606-37-3020-7