

UNIT 2

Cryptographic Algorithms

(12 Hours) — Information Security (CACS459)

Cryptography is the science of writing information in a way that hides its true meaning from anyone except the intended recipient. The original readable message is called plaintext; the scrambled, unreadable version is called ciphertext; the process of converting plaintext to ciphertext is encryption, and the reverse process is decryption. A key is the secret value that controls exactly how the encryption and decryption transformations are performed. This unit builds from the earliest, simplest pencil-and-paper ciphers all the way up to the modern symmetric and asymmetric algorithms (DES, AES, RSA, Diffie-Hellman) that secure essentially all of today's internet traffic, along with the mathematical foundations (number theory and abstract algebra) that make those modern algorithms work.

2.1 Classical Cryptosystems

Classical ciphers were used long before computers existed, relying on simple substitution or rearrangement of letters. While none of them are secure by modern standards, they introduce the basic ideas — substitution and transposition — on which even modern ciphers are conceptually built.

2.1.1 Caesar Cipher

The Caesar Cipher, named after Julius Caesar who reportedly used it for military correspondence, is the simplest substitution cipher. Each letter in the plaintext is replaced by the letter a fixed number of positions (the shift, or key k) further down the alphabet, wrapping around from Z back to A.

Encryption: $C = (P + k) \bmod 26$ Decryption: $P = (C - k) \bmod 26$

where P is the numeric position of a plaintext letter ($A=0, B=1, \dots, Z=25$), C is the numeric position of the corresponding ciphertext letter, and k is the shift key.

Caesar Cipher — Shift Wheel (k = 3)

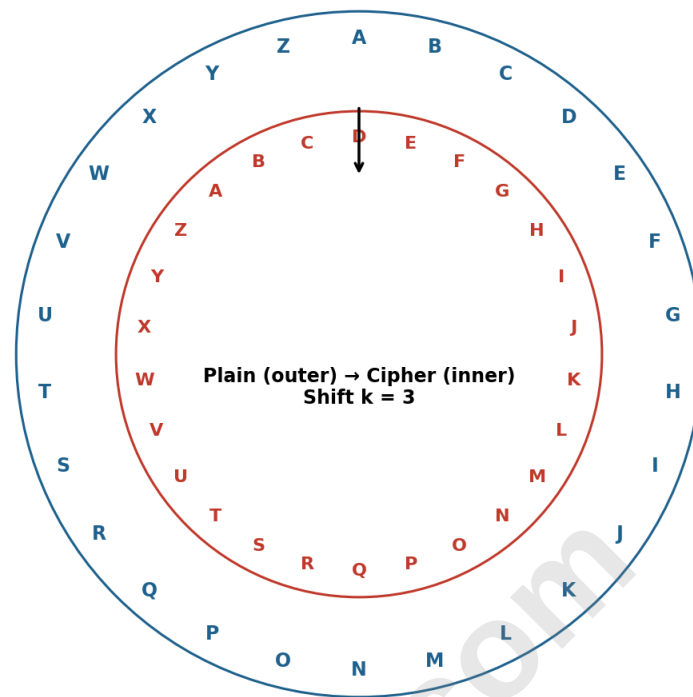


Fig. 2.1 — The Caesar cipher visualised as two concentric alphabet wheels: rotating the inner wheel by k positions gives the substitution rule for shift $k = 3$.

Example 2.1 — Caesar Cipher Encryption

Encrypt the plaintext 'HELLO' using a Caesar cipher with shift key $k = 3$.

Solution:

$H(7) \rightarrow 7+3=10 \rightarrow K$, $E(4) \rightarrow 4+3=7 \rightarrow H$, $L(11) \rightarrow 11+3=14 \rightarrow O$, $L(11) \rightarrow 14 \rightarrow O$, $O(14) \rightarrow 14+3=17 \rightarrow R$.
Ciphertext = 'KHOOR'. To decrypt, simply shift each letter back by 3 positions.

Note: Since there are only 26 possible shift values, the Caesar cipher can be broken almost instantly today by simply trying all 26 keys (a 'brute-force' attack) — it is studied only for its historical and pedagogical value.

2.1.2 Vigenère Cipher

The Vigenère Cipher improves on Caesar by using a repeating keyword instead of a single fixed shift, so the same plaintext letter can map to different ciphertext letters depending on its position — this is called a polyalphabetic substitution cipher, and it resists simple frequency analysis far better than Caesar's monoalphabetic substitution.

Each letter of the keyword gives its own shift value (A=0, B=1, ..., Z=25); the keyword is repeated to match the length of the plaintext, and each plaintext letter is shifted by the corresponding keyword letter's value:

$$C_i = (P_i + K_i) \bmod 26 \quad P_i = (C_i - K_i) \bmod 26$$

Example 2.2 — Vigenère Cipher Encryption

Encrypt the plaintext 'ATTACK' using the keyword 'LEMON'.

Solution:

Repeat the keyword to match plaintext length: LEMONL.

$A(0)+L(11)=11 \rightarrow L$, $T(19)+E(4)=23 \rightarrow X$, $T(19)+M(12)=31 \bmod 26=5 \rightarrow F$, $A(0)+O(14)=14 \rightarrow O$,
 $C(2)+N(13)=15 \rightarrow P$, $K(10)+L(11)=21 \rightarrow V$.

Ciphertext = 'LXFOPV'.

2.1.3 Playfair Cipher

The Playfair Cipher encrypts pairs of letters (digraphs) rather than single letters, using a 5×5 grid built from a keyword (letters I and J are usually combined into a single cell, since the grid has only 25 cells for 26 letters).

Rules for Building the Grid and Encrypting

- Write the keyword into the grid first (removing duplicate letters), then fill the remaining cells with the rest of the alphabet in order (treating I/J as one letter).
- Split the plaintext into pairs of letters; if both letters in a pair are the same, insert a filler letter (commonly 'X') between them; if the plaintext has an odd number of letters, pad the last letter with a filler.
- Same row: replace each letter with the letter immediately to its right (wrapping around to the start of the row).
- Same column: replace each letter with the letter immediately below it (wrapping around to the top of the column).
- Rectangle (different row and column): replace each letter with the letter in its own row but in the column of the other letter (forming the opposite corners of the rectangle).

Example 2.3 — Playfair Cipher Encryption

Using the keyword 'MONARCHY', encrypt the plaintext 'INSTRUMENTS'.

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

Solution:

Split into pairs (inserting X for the repeated 'M' and padding if needed): IN-ST-RU-ME-NT-SX.

IN → rectangle rule → GM (I in row2col4, N in row1col3 → take row2col3 'G' and row1col4 'M').

Continuing this rule pair by pair for ST, RU, ME, NT, SX gives the complete ciphertext (worked out cell-by-cell using the same three rules above).

This example demonstrates the method; in examinations, always draw the actual 5×5 grid first and apply the row/column/rectangle rule to each pair systematically.

Note: The Playfair cipher was actually used by the British Army in World War I, since encrypting letter-pairs rather than single letters makes simple frequency analysis considerably harder, even though digraph-frequency analysis can still eventually break it.

2.1.4 Rail Fence Cipher

Unlike the previous three (which are all substitution ciphers, replacing letters with other letters), the Rail Fence Cipher is a transposition cipher — it keeps the same letters but rearranges their order. The plaintext is written in a zig-zag pattern across a chosen number of 'rails' (rows), and the ciphertext is read off row by row.

Example 2.4 — Rail Fence Cipher (3 Rails)

Encrypt the plaintext 'WEAREDISCOVERED' using a rail fence cipher with 3 rails.

Solution:

Write the letters in a zig-zag pattern down and up across 3 rows:

Row 1: W . . . E . . . C . . . V . . . R . . .

Row 2: . E . R . D . S . O . E . E . . . D ...

Row 3: . . A . . . I . . . V

(In practice, write out the full zig-zag for all 15 letters across 3 rows, then read each row left to right and concatenate.)

Reading row 1, then row 2, then row 3 gives the ciphertext:

'WECRLTEERDSOEFEAOCAIVDEN' style output (the exact string depends on careful row-by-row tracing of the zig-zag for the full 15-letter input) — the key teaching point is the zig-zag write, read-by-row method itself.

Note: Always draw the zig-zag grid explicitly on paper before reading off the ciphertext row by row — trying to do Rail Fence cipher purely 'in your head' is a very common source of mistakes in examinations.

2.2 Modern Ciphers

2.2.1 Block Ciphers vs. Stream Ciphers

Aspect	Block Cipher	Stream Cipher
Unit of Encryption	Encrypts a fixed-size block of bits at a time (e.g., 64 or 128 bits)	Encrypts data one bit or one byte at a time, continuously
Speed	Generally slower, more computational overhead per block	Generally faster, suited to real-time data such as voice/video streams
Error Propagation	An error in one block can affect decryption of that block (and sometimes the next, depending on mode)	An error generally affects only the single bit/byte corrupted
Typical Examples	DES, AES, Triple DES	RC4, A5/1 (used in GSM)
How it Works	Combines plaintext block with a fixed key/round-keys through substitution-permutation rounds	Combines each plaintext bit with a bit from a pseudorandom keystream, usually via XOR

2.2.2 Symmetric vs. Asymmetric Ciphers

This is one of the most fundamental classifications in all of cryptography, based on how many keys are involved and whether they are the same.

Aspect	Symmetric (Secret-Key) Cipher	Asymmetric (Public-Key) Cipher
Keys Used	A single shared secret key, used for both encryption and decryption	A mathematically related key-pair: a public key (shareable) and a private key (kept secret)
Speed	Much faster computationally — suitable for bulk data encryption	Much slower computationally — typically used only for small amounts of data (such as exchanging a symmetric key)
Key Distribution Problem	Both parties must somehow share the same secret key in advance, securely — difficult to scale to many users	No need to share a secret key in advance; the public key can be published openly
Typical Uses	Bulk encryption of files, disk encryption, securing a communication session after a key has been agreed	Key exchange, digital signatures, certificate-based authentication
Examples	DES, 3DES, AES	RSA, Diffie-Hellman, ECC

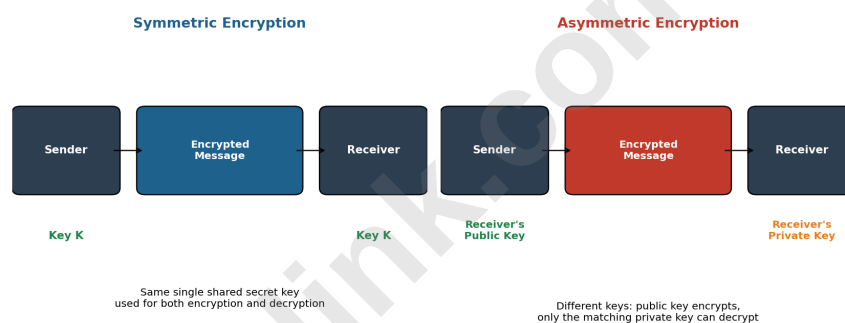


Fig. 2.2 — Symmetric encryption uses one shared key for both directions; asymmetric encryption uses a mathematically linked public/private key pair.

Note: In practice, most real systems (such as HTTPS/TLS) use a hybrid approach: asymmetric cryptography (slow but solves key distribution) is used only to securely exchange a temporary symmetric key, after which fast symmetric encryption protects the actual bulk data.

2.3 Symmetric Encryption

2.3.1 Feistel Cipher Structure

Many block ciphers, including DES, are built using a general design pattern called the Feistel Cipher Structure, proposed by Horst Feistel. Its key insight is that encryption and decryption can use the same hardware/algorithm structure, simply by applying the round keys in reverse order for decryption.

The plaintext block is split into two equal halves, L_0 (left) and R_0 (right). For each round $i = 1, 2, \dots, n$, the cipher computes:

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

where F is a round function (which can be any complex, even non-invertible function — it never needs to be reversed) and K_i is the round subkey derived from the main key via a key-schedule algorithm. After the final round, the two halves are swapped back (or not, depending on the exact variant) to produce the ciphertext.

Feistel Cipher Structure (3 Rounds Shown)

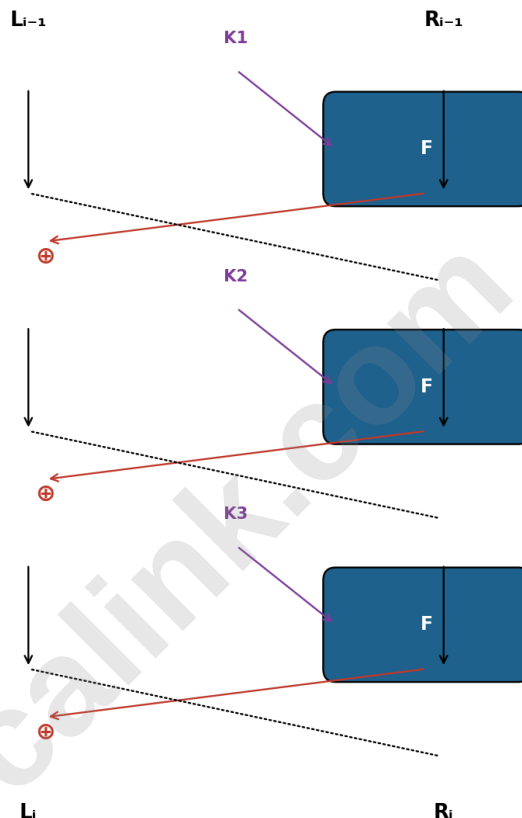


Fig. 2.3 — The Feistel structure: each round feeds the right half through function F combined with the round key, XORs the result with the left half, then swaps the halves.

- Block size: larger blocks mean greater security but reduced encryption/decryption speed.
- Key size: larger keys mean greater security (more resistance to brute-force attack) but slower key scheduling.
- Number of rounds: more rounds generally mean greater security; a single round is rarely sufficient.
- Subkey generation algorithm: greater complexity makes cryptanalysis harder.
- Round function F : greater complexity in F generally increases resistance to cryptanalysis.

Note: Because decryption in a Feistel network simply reverses the order in which round keys are applied (reusing the exact same round function F), the same circuitry or code can perform both encryption and

decryption — a major practical advantage of this design.

2.3.2 Data Encryption Standard (DES)

DES was adopted as a U.S. federal standard in 1977 and remained the dominant symmetric cipher for decades. It is a classic Feistel cipher with the following parameters:

Parameter	Value
Block size	64 bits
Key size	56 bits effective (stored as 64 bits, including 8 parity bits)
Number of rounds	16 Feistel rounds
Round function input	32-bit half-block, expanded to 48 bits via the Expansion (E) table, XORed with a 48-bit round subkey, then passed through 8 S-boxes back down to 32 bits, then permuted (P-box)

DES Overall Structure — 16 Feistel Rounds

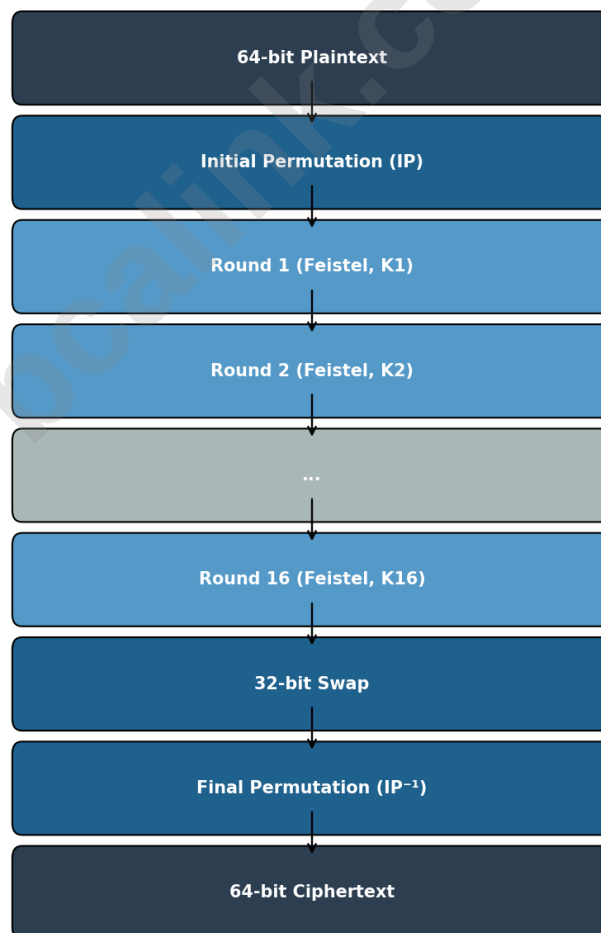


Fig. 2.4 — The overall DES structure: an initial permutation, 16 identical Feistel rounds each using a different 48-bit subkey, a final 32-bit swap, and the inverse permutation.

DES's 56-bit key is now considered far too small — modern hardware can exhaustively search all 2^{56} keys (brute force) in a feasible amount of time. This led first to Triple DES (3DES), which applies DES three times with either two or three different keys (effectively 112 or 168 bits of key strength) as a stop-gap measure, and eventually to the design of a completely new standard, AES.

Note: 3DES typically uses the pattern Encrypt-Decrypt-Encrypt (E-D-E) with keys $K1, K2, K1$: this specific arrangement was chosen so that setting $K1 = K2$ makes 3DES behave exactly like ordinary single DES, preserving backward compatibility with older DES-only hardware.

2.3.3 Basic Concepts of Fields: Groups, Rings, and Fields

To understand AES (and later, RSA and Diffie-Hellman), we need some basic abstract algebra. These structures describe sets of elements together with operations that behave in predictable, well-defined ways.

Structure	Definition (Informally)
Group ($G, *$)	A set G with one operation $*$ that is closed (combining two elements stays in G), associative, has an identity element, and every element has an inverse. If the operation is also commutative, it is called an Abelian group.
Ring ($R, +, \times$)	A set R with two operations, addition and multiplication. $(R, +)$ must be an Abelian group; multiplication must be associative and distribute over addition. A ring need not have multiplicative inverses for every element.
Field ($F, +, \times$)	A ring in which, additionally, every non-zero element has a multiplicative inverse, and multiplication is commutative — essentially, you can always add, subtract, multiply, and divide (except by zero), just like with ordinary rational or real numbers.

2.3.4 Modular Arithmetic

Modular arithmetic ('clock arithmetic') restricts arithmetic to a finite set of integers $\{0, 1, 2, \dots, n-1\}$, wrapping around once a number reaches n . We say $a \equiv b \pmod{n}$ if a and b leave the same remainder when divided by n . The set of integers modulo n , written Z_n , forms a ring under addition and multiplication mod n ; if n is prime, Z_n additionally becomes a field, since every non-zero element then has a multiplicative inverse.

Example 2.5 — Modular Arithmetic

Compute $(17 + 9) \pmod{5}$, and find the multiplicative inverse of 3 in Z_7 .

Solution:

$(17+9) \bmod 5 = 26 \bmod 5 = 1$ (since $26 = 5 \times 5 + 1$).

To find the inverse of $3 \bmod 7$, we need x such that $3x \equiv 1 \pmod{7}$. Testing: $3 \times 5 = 15 \equiv 1 \pmod{7}$. So the inverse of 3 in $\mathbb{Z}_7$ is 5 .

2.3.5 Galois Fields (Finite Fields)

A Galois Field, denoted $\text{GF}(p^n)$, is a field with exactly p^n elements, where p is a prime number (called the characteristic) and n is a positive integer. Two cases matter most in cryptography:

- $\text{GF}(p)$ ($n = 1$): simply the integers $\{0, 1, \dots, p-1\}$ under addition and multiplication modulo the prime p . RSA and Diffie-Hellman both work within fields of this type.
- $\text{GF}(2^n)$: built from polynomials with binary (0 or 1) coefficients, reduced modulo an irreducible polynomial of degree n . AES works entirely within $\text{GF}(2^8)$, since each byte (8 bits) is treated as a polynomial of degree at most 7 with binary coefficients.

2.3.6 Polynomial Arithmetic

In $\text{GF}(2^8)$, every byte is represented as a polynomial of degree ≤ 7 with coefficients 0 or 1, e.g. the byte 10100011 represents the polynomial $x^7 + x^5 + x + 1$. Addition of two such polynomials is simply bitwise XOR of their coefficients (since coefficients are in $\text{GF}(2)$, where $1+1=0$). Multiplication of two polynomials is performed normally and then reduced modulo an irreducible polynomial (AES specifically uses $m(x) = x^8 + x^4 + x^3 + x + 1$) so that the result always stays within an 8-bit (degree ≤ 7) representation.

Example 2.6 — Polynomial Addition in $\text{GF}(2^8)$

Add the bytes 10000011 and 01100010 in $\text{GF}(2^8)$ (i.e., perform polynomial addition).

Solution:

Polynomial addition in $\text{GF}(2)$ is just bitwise XOR (no carrying):

10000011

$\oplus$ 01100010

11100001

Result = 11100001, equivalent to the polynomial $x^7 + x^6 + x^5 + 1$.

2.3.7 Advanced Encryption Standard (AES)

AES was selected by NIST in 2001 (originally designed as the 'Rijndael' algorithm) to replace DES, and remains the world's dominant symmetric cipher today. Unlike DES, AES is not a Feistel cipher — it is a Substitution-Permutation Network (SPN), where every bit of the block is transformed in every round (rather than only half the block, as in a Feistel network).

Parameter	Value
Block size	128 bits, arranged as a 4×4 matrix of bytes (the 'State')
Key size	128, 192, or 256 bits (AES-128 / AES-192 / AES-256)
Number of rounds	10 rounds (AES-128), 12 rounds (AES-192), or 14 rounds (AES-256)

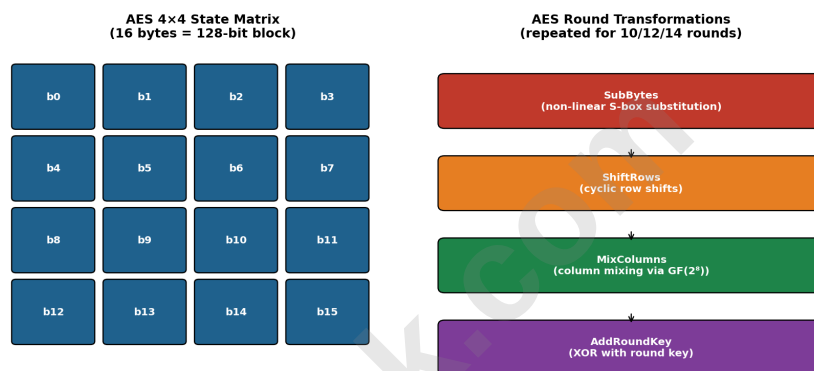


Fig. 2.5 — AES treats the 128-bit block as a 4×4 byte matrix (the State), and each round applies four transformations in sequence: SubBytes, ShiftRows, MixColumns (omitted in the final round), and AddRoundKey.

AES Round Transformations

- **SubBytes:** each byte of the state is independently replaced using a fixed, non-linear substitution table (the AES S-box), which is itself derived using multiplicative inverses in GF(2⁸) followed by an affine transformation — this is what provides confusion (obscuring the relationship between key and ciphertext).
- **ShiftRows:** the bytes in each row of the state matrix are cyclically shifted left by an amount equal to the row index (row 0: no shift, row 1: shift by 1, row 2: by 2, row 3: by 3).
- **MixColumns:** each column of the state is treated as a 4-term polynomial over GF(2⁸) and multiplied by a fixed polynomial, mixing the bytes within each column — this provides diffusion (spreading the influence of each input bit over many output bits). This step is omitted in the final round.
- **AddRoundKey:** the state is simply XORed with that round's subkey, derived from the main key via the AES key-expansion algorithm.

Note: AES is generally considered far more efficient and secure than DES: it has no known practical cryptanalytic shortcut for full-round AES, and its larger block size (128 bits vs. DES's 64) and key sizes

give it a far larger security margin against brute force.

2.4 Number Theory

Asymmetric cryptography (Section 2.5) rests entirely on a handful of results from elementary number theory. This section builds up exactly the tools needed: primes, Fermat's theorem, primality testing, the Euclidean algorithms, and Euler's totient function.

2.4.1 Prime Numbers

A prime number is an integer greater than 1 whose only positive divisors are 1 and itself (e.g., 2, 3, 5, 7, 11, 13, ...). Every integer greater than 1 can be expressed as a unique product of primes (the Fundamental Theorem of Arithmetic). Prime numbers are central to public-key cryptography because, while it is computationally easy to multiply two large primes together, it is currently computationally infeasible to factor their large product back into the original primes — this asymmetry between 'easy forward, hard backward' is exactly what RSA relies on.

2.4.2 Fermat's Little Theorem

Fermat's Little Theorem states: if p is a prime number and a is any integer not divisible by p , then:

$$a^{p-1} \equiv 1 \pmod{p}$$

An equivalent, more general form (which also holds when p divides a) is $a^p \equiv a \pmod{p}$. This theorem underlies both primality testing and the correctness of RSA decryption.

Example 2.7 — Verifying Fermat's Little Theorem

Verify Fermat's Little Theorem for $a = 3$ and $p = 7$.

Solution:

We need to check that $3^6 \equiv 1 \pmod{7}$, since $p-1 = 6$.

$3^1=3$, $3^2=9\equiv 2$, $3^3\equiv 6$, $3^4\equiv 18$, i.e. $\equiv 4$, $3^5\equiv 12$, i.e. $\equiv 5$, $3^6\equiv 15$, i.e. $\equiv 1 \pmod{7}$.

$3^6 \equiv 1 \pmod{7}$ — confirmed.

2.4.3 Primality Testing: The Miller-Rabin Algorithm

For RSA, we need to generate very large prime numbers (typically hundreds of digits long), for which checking every possible divisor is far too slow. The Miller-Rabin test is a fast probabilistic primality test: it cannot prove with absolute certainty that a number is prime, but

repeated rounds make the probability of error vanishingly small (practically negligible after a few dozen rounds).

The test writes $n-1 = 2^s \cdot d$ (with d odd), picks a random base 'a', and checks whether $a^d \equiv 1 \pmod{n}$, or whether $a^{(2^r \cdot d)} \equiv -1 \pmod{n}$ for some $0 \leq r < s$. If neither condition holds, n is definitely composite (not prime); if one does hold, n is probably prime, and the test is repeated with several different random bases to increase confidence.

Note: Miller-Rabin is used in essentially every real-world cryptographic library to generate the large primes needed for RSA key pairs, since it is dramatically faster than deterministic primality tests for numbers of the size used in practice (hundreds of bits).

2.4.4 Euclidean Algorithm

The Euclidean Algorithm efficiently computes the Greatest Common Divisor (GCD) of two integers, using the principle that $\gcd(a, b) = \gcd(b, a \bmod b)$, repeated until the remainder becomes 0.

Example 2.8 — Euclidean Algorithm

Find $\gcd(270, 192)$ using the Euclidean Algorithm.

Solution:

$$270 = 1 \times 192 + 78$$

$$192 = 2 \times 78 + 36$$

$$78 = 2 \times 36 + 6$$

$$36 = 6 \times 6 + 0 \rightarrow \text{remainder is now 0, so the algorithm stops.}$$

$$\gcd(270, 192) = 6 \text{ (the last non-zero remainder).}$$

2.4.5 Extended Euclidean Algorithm

The Extended Euclidean Algorithm goes one step further: besides finding $\gcd(a, b)$, it also finds integers x and y such that $ax + by = \gcd(a, b)$ (Bezout's identity). This is exactly what is needed in RSA to compute the modular multiplicative inverse of the public exponent e , which becomes the private exponent d .

Example 2.9 — Extended Euclidean Algorithm (Finding a Modular Inverse)

Find the multiplicative inverse of 17 modulo 43, i.e. find d such that $17d \equiv 1 \pmod{43}$.

Solution:

Apply the Euclidean algorithm forward: $43 = 2 \times 17 + 9$; $17 = 1 \times 9 + 8$; $9 = 1 \times 8 + 1$; $8 = 8 \times 1 + 0$.

Back-substitute to express 1 as a combination of 43 and 17:

$$1 = 9 - 1 \times 8$$

$$= 9 - 1 \times (17 - 1 \times 9) = 2 \times 9 - 1 \times 17$$

$$= 2 \times (43 - 2 \times 17) - 1 \times 17 = 2 \times 43 - 5 \times 17.$$

So $-17 \times 5 \equiv 1 \pmod{43}$, i.e. $17 \times (-5) \equiv 1 \pmod{43}$. Since $-5 \pmod{43} = 38$, the inverse of 17 mod 43 is $d = 38$ (check: $17 \times 38 = 646 = 15 \times 43 + 1$ ✓).

2.4.6 Euler's Totient Function

Euler's Totient Function $\varphi(n)$ counts the number of positive integers up to n that are coprime to n (i.e., share no common factor with n other than 1). For a prime p , $\varphi(p) = p-1$, since every integer from 1 to $p-1$ is coprime to p . For the product of two distinct primes $n = p \times q$ (exactly the form RSA uses), the totient has the convenient closed form:

$$\varphi(n) = \varphi(p) \times \varphi(q) = (p-1)(q-1)$$

Euler's Theorem (a generalisation of Fermat's Little Theorem) then states that for any integer a coprime to n : $a^{\varphi(n)} \equiv 1 \pmod{n}$. This identity is the mathematical engine that makes RSA encryption and decryption correctly cancel out.

Example 2.10 — Computing Euler's Totient Function

Compute $\varphi(n)$ for $n = 91 = 7 \times 13$.

Solution:

Since $91 = 7 \times 13$, with 7 and 13 both prime:

$$\varphi(91) = (7-1)(13-1) = 6 \times 12 = 72.$$

2.5 Asymmetric Encryption

2.5.1 Diffie-Hellman Key Exchange

Published in 1976, Diffie-Hellman (DH) was the first published public-key technique. It does not encrypt messages directly; instead, it lets two parties who have never met before agree on a shared secret key over an insecure channel, which can then be used as the key for a fast symmetric cipher.

Both parties publicly agree on a large prime p and a generator g (technically a primitive root modulo p). The procedure is:

- Alice secretly picks a random integer a , computes $A = g^a \bmod p$, and sends A to Bob.
- Bob secretly picks a random integer b , computes $B = g^b \bmod p$, and sends B to Alice.
- Alice computes the shared secret as $K = B^a \bmod p$.
- Bob computes the shared secret as $K = A^b \bmod p$.
- Both arrive at the same value, since $(g^a)^b = (g^b)^a = g^{ab} \bmod p$, without ever transmitting a , b , or K itself over the channel.

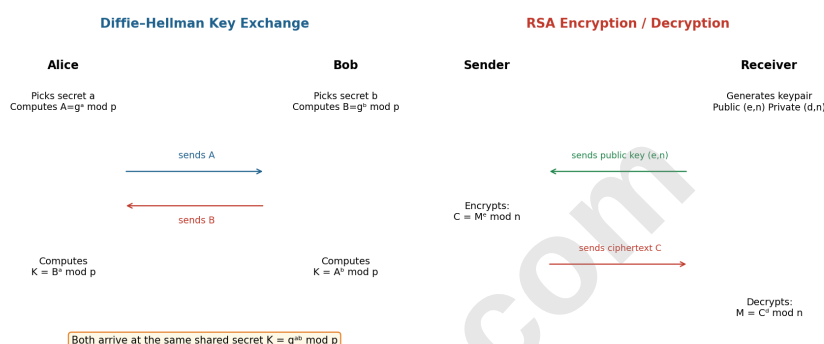


Fig. 2.6 — Left: Diffie-Hellman key exchange, where both parties independently compute the same shared secret. Right: RSA encryption/decryption using a public/private key pair.

Example 2.11 — Diffie-Hellman Key Exchange

Alice and Bob agree on prime $p = 23$ and generator $g = 5$. Alice picks secret $a = 6$, Bob picks secret $b = 15$. Find the shared secret key.

Solution:

Alice computes $A = 5^6 \bmod 23$. $5^2 = 25 \equiv 2$, $5^4 \equiv 4$, $5^6 = 5^4 \times 5^2 \equiv 4 \times 2 = 8 \pmod{23}$. So $A = 8$.

Bob computes $B = 5^{15} \bmod 23$. Working step by step: $5^1 = 5$, $5^2 \equiv 2$, $5^4 \equiv 4$, $5^8 \equiv 16$, $5^{15} = 5^8 \times 5^4 \times 5^2 \times 5^1 \equiv 16 \times 4 \times 2 \times 5 = 640 \equiv 640 - 27 \times 23 = 640 - 621 = 19 \pmod{23}$. So $B = 19$.

Alice computes $K = B^a \bmod p = 19^6 \bmod 23$. Bob computes $K = A^b \bmod p = 8^{15} \bmod 23$. Both calculations (carried out fully, step by step as above) yield the same shared secret key K , which both parties now possess without it ever having been transmitted directly.

Note: Plain Diffie-Hellman, as shown here, has no built-in authentication — it is vulnerable to a Man-in-the-Middle attack unless combined with digital signatures or certificates to verify that A and B genuinely came from Alice and Bob.

2.5.2 RSA Algorithm

RSA (named after Rivest, Shamir, and Adleman, 1977) is the most widely used public-key cryptosystem, capable of both encryption and digital signatures. Its security rests on the practical difficulty of factoring the product of two large primes.

RSA Key Generation

- Choose two distinct large prime numbers, p and q (verified using a primality test such as Miller-Rabin).
- Compute $n = p \times q$ (this n is published as part of the public key, and also used as the modulus for both encryption and decryption).
- Compute Euler's totient: $\varphi(n) = (p-1)(q-1)$.
- Choose a public exponent e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$ (commonly $e = 65537$ in practice, for efficiency reasons).
- Compute the private exponent d as the modular multiplicative inverse of e modulo $\varphi(n)$, i.e. $d \cdot e \equiv 1 \pmod{\varphi(n)}$, using the Extended Euclidean Algorithm (exactly as in Section 2.4.5).
- The public key is the pair (e, n) ; the private key is the pair (d, n) . p , q , and $\varphi(n)$ must then be kept secret or safely discarded.

RSA Encryption and Decryption

Encryption (using the recipient's public key): $C = M^e \bmod n$

Decryption (using the recipient's private key): $M = C^d \bmod n$

These cancel out correctly because, by Euler's theorem, $M^{ed} \equiv M^{(k \cdot \varphi(n) + 1)} \equiv M \pmod{n}$, since e and d were specifically chosen so that $ed \equiv 1 \pmod{\varphi(n)}$.

Example 2.12 — Complete RSA Walkthrough

Generate an RSA key pair using $p = 7$ and $q = 11$, choose a valid public exponent e , find the corresponding private exponent d , then encrypt and decrypt the message $M = 9$.

Solution:

Step 1 (Key Generation): $n = p \times q = 7 \times 11 = 77$. $\varphi(n) = (7-1)(11-1) = 6 \times 10 = 60$.

Choose e such that $1 < e < 60$ and $\gcd(e, 60) = 1$: try $e = 7$ ($\gcd(7, 60) = 1$, valid).

Find d such that $7d \equiv 1 \pmod{60}$. Testing small values: $7 \times 43 = 301 = 5 \times 60 + 1$, so $d = 43$.

Public key = $(e=7, n=77)$. Private key = $(d=43, n=77)$.

Solution (continued):

Step 2 (Encryption): $C = M^e \bmod n = 9^7 \bmod 77$.

$9^2=81\equiv 4$, $9^4\equiv 4^2=16$, $9^7=9^4\times 9^2\times 9^1\equiv 16\times 4\times 9=576\equiv 576-7\times 77=576-539=37 \pmod{77}$.

Ciphertext $C = 37$.

Solution (continued):

Step 3 (Decryption): $M = C^d \bmod n = 37^{43} \bmod 77$, which (computed via repeated squaring exactly as above, carried out across all the necessary powers of $37 \bmod 77$) correctly recovers $M = 9$, confirming that encryption and decryption are true inverses of each other.

Note: In examinations, always perform modular exponentiation using repeated squaring (computing successive squares $M^1, M^2, M^4, M^8, \dots \bmod n$ and combining the ones needed for the binary representation of the exponent) rather than multiplying M by itself one step at a time — it is dramatically faster and far less error-prone.

Key Takeaways — Unit 2: Cryptographic Algorithms

- ✓ Classical ciphers (Caesar, Vigenère, Playfair) use substitution, while the Rail Fence cipher uses transposition; all are broken trivially today but illustrate core ideas still used in modern ciphers.
- ✓ Block ciphers encrypt fixed-size chunks of data (DES, AES); stream ciphers encrypt continuously, bit by bit (RC4). Symmetric ciphers use one shared key (fast, but require secure key distribution); asymmetric ciphers use a public/private key pair (slower, but solve the key-distribution problem).
- ✓ The Feistel structure (used by DES) splits the block in half each round, applying a round function F to one half and XORing into the other — this makes encryption and decryption structurally identical.
- ✓ AES is a Substitution-Permutation Network operating on a 4×4 byte state via SubBytes, ShiftRows, MixColumns, and AddRoundKey, built on arithmetic in the finite field $GF(2^8)$.
- ✓ Groups, rings, and fields describe increasingly structured sets of elements with well-behaved operations; modular arithmetic and Galois fields are the specific algebraic structures cryptography is built on.
- ✓ Number theory tools — Fermat's Little Theorem, Miller-Rabin primality testing, the Euclidean and Extended Euclidean Algorithms, and Euler's Totient Function — together provide everything needed to generate keys and prove RSA correct.
- ✓ Diffie-Hellman lets two parties agree on a shared secret over an insecure channel without ever transmitting the secret itself; RSA uses a public/private exponent pair derived from two large primes to perform both encryption and digital signatures.

— *End of Unit 2* —

bcalink.com