Securing Applications and Data

AP Cybersecurity

Application and Data Vulnerabilities and Attacks

Applications 应用程序 are the programs that run on computers, and **data** is what they process - both are prime targets. If files are stored **unencrypted**, anyone with access to the drive can read them. If a normal user is given **administrative** 管理员 privileges, an adversary who steals that account gains sweeping power.

The biggest application danger is bad **user input**. When a program does not check what a user types, an adversary can slip in commands - an **injection attack** 注入攻击. **Data validation** 数据验证 (checking input meets expected rules) is the defense. Key attacks:

- **SQL injection** 数据库注入 - inserting **SQL** commands into an input field to read or change a database.
- **Cross-site scripting (XSS)** 跨站脚本 - injecting malicious script into a website that runs in another user's browser.
- **Buffer overflow** 缓冲区溢出 - sending more data than a memory **buffer** 缓冲区 can hold, so it overflows into nearby memory and may run the adversary's code.
- **Directory traversal** 目录遍历 - using `../` sequences in a URL to reach files outside the intended folder, such as `/etc/passwd`.

We rate data risk by sensitivity: unencrypted military plans are **high** risk; customer data with a weak key is **moderate**; low-value data with short keys is **low**.

Protecting Applications and Data: Managerial Controls and Access Controls

Data is classified by its **state** - **at rest** 静态数据 (stored on a drive), **in transit** 传输中数据 (moving between devices), and **in use** 使用中数据 (being processed). Data at rest and in transit can be **encrypted** so a thief cannot read it; data in use must be decrypted, so **access controls** guard it instead.

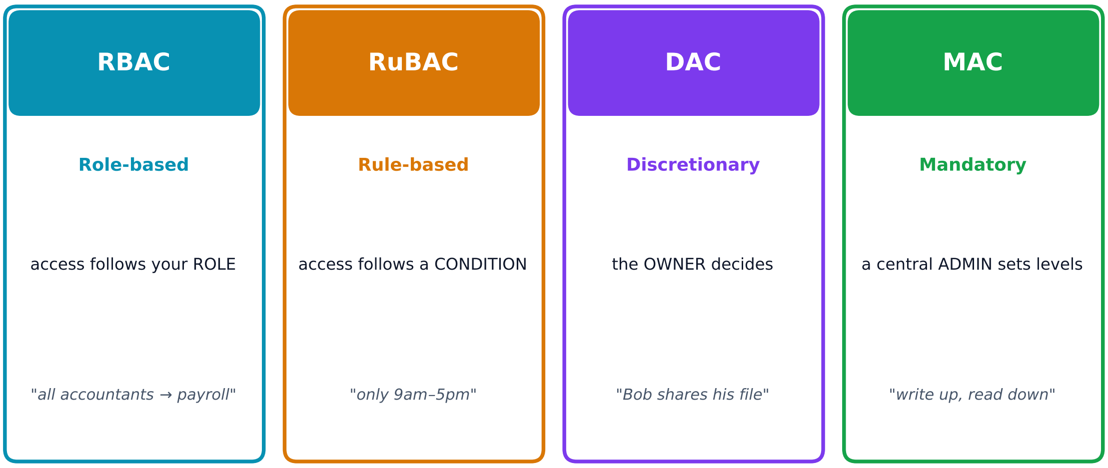
Some data types are **regulated** 受监管 - the law dictates how they must be stored, transmitted and handled - so an organisation must achieve **compliance** 合规 by matching its controls to the rules. The exam expects you to pair each data type with its governing law:

Regulated data	What it is	Governing law
personally identifiable information (PII) 个人身份信息	anything identifying a person: name, address, SSN, biometrics, date of birth	The Privacy Act (1974); COPPA for under-13s
protected health information (PHI) 受保护健康信息	health, treatment and healthcare-payment records	**HIPAA** (1996)
payment card information (PCI) 支付卡信息	card number, expiry, CVV, cardholder name	**PCI-DSS**

An organisation that collects regulated data must **label** it and hold **policies** that keep its storage, transmission and handling compliant - the higher the sensitivity, the higher the required degree of security.

Access control decides which **subjects** (users) may perform which **operations** on which **objects** (files). Four models:

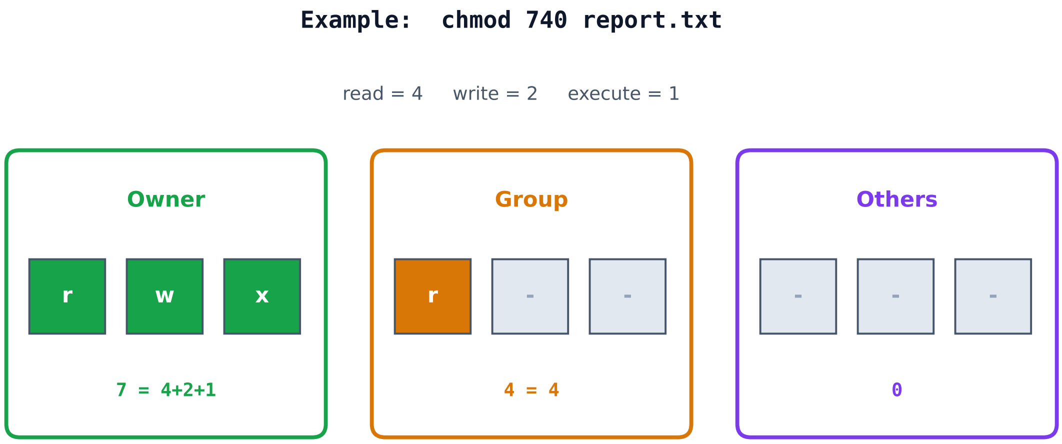
- **Role-based (RBAC)** 基于角色的访问控制 - access follows your **role** (all ”accountants” reach the payroll software).
- **Rule-based (RuBAC)** 基于规则的访问控制 - access follows **conditions** (only during business hours), layered on another model.
- **Discretionary (DAC)** 自主访问控制 - the **owner** of a file decides who else may use it.
- **Mandatory (MAC)** 强制访问控制 - a central administrator sets strict levels; the **Bell-LaPadula** model summarises it as ”write up, read down”.



Four access-control models decide who reaches which object, and how

A guiding idea across all models is the **principle of least privilege** 最小权限原则 - give each entity exactly the access it needs and no more.

On a **Linux** system, each file has three permissions - **read (r), write (w), execute (x)** - for three groups: the **owner**, the **group**, and **others**. The `chmod` command sets them with numbers, adding 4 (read) + 2 (write) + 1 (execute). So `chmod 640` means owner read+write (6), group read (4), others nothing (0).



Linux file permissions: read/write/execute for owner, group, and others

Worked example. A principal wants only herself to read *and* edit a file, her staff group to read it, and no one else to touch it. Read+write = 4+2 = **6** for the owner, read = **4** for the group, nothing = **0** for others, giving `chmod 640 file`. The listing then shows `-rw-r-----`. To also let the owner run the file as a program you would add execute (7 = 4+2+1), giving `chmod 740`.

Protecting Stored Data with Cryptography

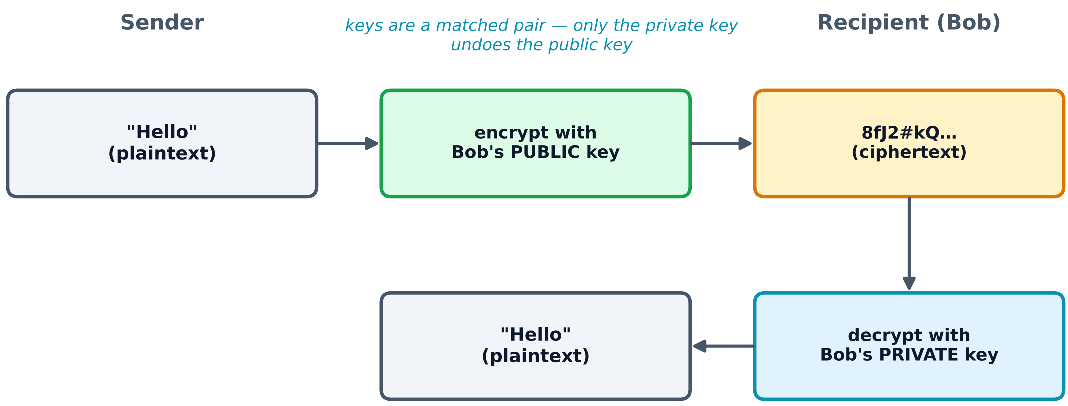
Cryptography 密码学 hides information. An **encryption** algorithm combines the **plaintext** 明文 with a **key** 密钥 to produce **ciphertext** 密文; **decryption** reverses it. The **keyspace** 密钥空间 is the number of possible keys - the bigger it is, the longer an adversary needs to guess. An **n-bit** key has a keyspace of 2^n.

Symmetric encryption 对称加密 uses the **same key** to encrypt and decrypt. The standard is **AES** 高级加密标准, a **block cipher** 分组密码 that works on 128-bit blocks and secures Wi-Fi, browsing, and stored files. Because both sides need the same secret key, sharing that key safely is the challenge.

The Enigma machine scrambled messages with rotors —an early, breakable example of encryption

Asymmetric Cryptography

Asymmetric encryption 非对称加密 solves the key-sharing problem with a **key pair** 密钥对 - a **public key** 公钥 anyone may see and a **private key** 私钥 kept secret. The keys are mathematical inverses: whatever one locks, only the other unlocks. To send you a secret, I encrypt with **your public key**, and only **your private key** can decrypt it - so we never had to share a secret in advance.



Asymmetric encryption: encrypt with the public key, decrypt with the private key

Longer keys mean larger keyspaces and more security, but slower encryption. Common asymmetric algorithms are **RSA** and **elliptic curve cryptography (ECC)** 椭圆曲线密码学, used in digital signatures and certificates. Remember: you can only compare key lengths **within the same algorithm** - an RSA 4096-bit key is not directly comparable to an AES 256-bit key.

Protecting Applications

Two design principles keep applications safe from the start. **Secure by design** 安全设计 builds security into every phase of development, not as an afterthought. **Secure by default** 默认安全 means the product ships with its security features already **enabled** - safe straight out of the box.

Secure by design rests on three principles a company must adopt: (1) **take ownership of its customers' security outcomes** rather than shifting blame onto users, (2) embrace **radical transparency and accountability** –sharing security-relevant news and updates quickly so everyone becomes safer, and (3) build the **organisational structure and leadership** that makes security a first-class goal.

The key defense against injection attacks is **input sanitization** 输入清理. Special **control characters** 控制字符 - the single quote, double quote, and semicolon - can be used to manipulate a system, so a good program removes or rejects them before processing. Sanitization protects against SQL injection, XSS, and directory-traversal attacks alike.

Detecting Attacks on Data and Applications

To detect data attacks, systems perform **accounting** 审计记录 - logging who accessed what and when. But logs are huge, so **log analysis** must be **automated** to run at a useful speed; a human reading raw logs is far too slow. A clever complement is a **honeypot** 蜜罐 - a fake file that looks valuable; since no one has a real reason to open it, any access is a clear, **near-instantaneous** sign of an attack. Watch especially for attempts to **delete or copy sensitive files**. **Cryptographic hashes** also help: re-hash a file and compare - if the digest changed, the file was altered.

Choosing detective controls means weighing **cost** (honeypots are cheap; a **data loss prevention (DLP)** 数据泄露防护 service is powerful but pricey) against the sensitivity of the data. To read a specific attack from logs, look for its signature: SQL injection shows `OR 1=1` and `--`; XSS shows `<script>` tags; directory traversal shows `../` sequences; a buffer overflow shows unusually long input strings.

Exam tips

- Match each application attack to its **evidence in a log**: `OR 1=1` / `--` = **SQL injection**; `<script>` = **XSS**; `../` = **directory traversal**; very long input = **buffer overflow**.
- Learn the four access-control models by their **decider**: RBAC = your role, RuBAC = a condition, DAC = the file's owner, MAC = a central admin. **Least privilege** underlies them all.

- Read Linux permissions by adding **4+2+1** per group - `chmod 750` = owner rwx (7), group r-x (5), others none (0). Practice converting both ways.
- **Symmetric** = one shared key (fast, AES); **asymmetric** = a public/private key pair (solves key sharing, RSA/ECC). Encrypt with the **recipient's public key**.
- **Input sanitization** is the single best answer for preventing injection attacks; a **honeypot** is the classic cheap detective control.