Solutions

Each answer shows the steps, then the **result**.

2.1

- **plaintext** — the original readable data
- **ciphertext** — the scrambled, unreadable form after encryption

2.2

- **symmetric** uses the **same** key for both; **asymmetric** uses a **pair** (public to encrypt, private to decrypt)

2.3

- **key distribution** — sharing the secret key safely with the other party

2.4

- an **identity** (domain/organisation) and its **public key** (signed by a CA)

3.1

- Bob encrypts with **Alice's public key**
- only **Alice's private key** can decrypt it
- since Alice keeps her private key secret, an interceptor with only the public key and ciphertext cannot read the message

3.2

- asymmetric is **secure for key exchange** but **slow**
- symmetric is **fast** but needs a shared key
- so asymmetric is used once to share a **session key**, then fast symmetric encryption carries the bulk data

3.3

- any two: **encryption** of data in transit; **authentication** of the server (via certificate); **integrity** (detecting tampering)

3.4

- any one: the certificate is **in date**; the **subject name matches** the site's URL; it is **signed by a trusted CA** / chains to a trusted root

3.5

- the sender hashes the message and encrypts that hash with their **private key** to form the signature
- the receiver decrypts the signature with the sender's **public key** to get the original hash, and **re-hashes** the received message
- if the two hashes **match** the message is unchanged, and if they differ it was altered in transit

3.6

(a) the shop publishes its **public** key and keeps its **private** key secret

- the customer's browser encrypts the card details with the shop's **public** key
- only the matching **private** key can decrypt them, so only the shop can read the message

(b) the private key is the only thing that can decrypt messages sent to the shop and the only thing that can create its signature

- publishing it would let anyone read customers' data and impersonate the shop; the shop publishes the **public** key instead, inside its certificate

(c) the certificate contains the shop's **public key**, its domain name and organisation details, the CA's name, and the validity dates

- the whole lot is **signed** by the CA using the CA's own private key
- a browser that trusts that CA can verify the signature with the CA's public key, so it knows the public key really belongs to that shop

(d) SSL/TLS provides a **secure, encrypted channel** between the browser and the server

- giving confidentiality, integrity and authentication of the server
- it sits **between** the application layer protocol, such as HTTP, and the transport layer — which is why the secure version is called HTTPS

(e) the browser sends a "hello" listing the TLS versions and cipher suites it supports

- the server replies with its chosen cipher suite and its **digital certificate**
- the browser verifies the certificate against a trusted CA
- the two then agree a **session key**, which the browser sends encrypted with the server's public key

(f) symmetric encryption is far **faster** and less demanding on the processor for large amounts of data

- asymmetric encryption is used only to exchange the session key safely, after which the bulk of the traffic uses that shared key