Cryptography

> *Created: 30 October 2025 | Last Updated: 30 October 2025*

Basic Crypto Primitives Public Key Encryption

Public Key Encryption and its practical applications in real-world systems like cloud security.

中英雙語解釋 / Bilingual Explanation

公開金鑰加密的歷史背景 / Historical Background of Public Key Encryption

- **1976年：** Whitfield Diffie和Martin Hellman提出了公開金鑰加密（非對稱加密），其中加密和解密使用不同的金鑰。
- ***1976:*** *Whitfield Diffie and Martin Hellman proposed* ***public key encryption*** *(asymmetric encryption).*
- **1997年：** 披露了非對稱金鑰演算法實際上在1970年代早期已由英國政府通信總部開發。
- ***1997:*** *It was disclosed that asymmetric algorithms had been developed in the early 1970s by British GCHQ.*
- 他們將該技術稱為**非秘密加密**。
- *They referred to the technique as* ***non-secret encryption****.*

公開金鑰加密原理 / Public Key Encryption Principles

基本概念：
Basic Concept:

- 使用**公開揭露的金鑰PK**進行加密
- *Use a **publicly disclosed key PK** to encrypt*
- 使用**秘密金鑰SK**進行解密
- *Use a **secret key SK** to decrypt*
- 必需的關係是： $M = \{\{M\}_{PK}\}_{SK}$
- *The requisite relationship is: $M = \{\{M\}_{PK}\}_{SK}$*

公開金鑰加密的基礎 / Basis of Public Key Encryption

任何公開金鑰加密的基礎是識別一個**單向函式**：
*The basis of any public key encryption is identifying a **one-way function**:*

- **容易計算**
- ***Easily computed***
- **沒有額外資訊難以逆推**
- ***Difficult to invert without additional information***

RSA使用的例子：
Example used by RSA:

- 容易將兩個大質數p_1和p_2相乘。
- *It is easy to **multiply** two large primes p_1 and p_2.*
- 但是，將$N = p_1 \times p_2$**分解**以恢復p_1和p_2非常困難。
- *However, it is very difficult to **factor** $N = p_1 \times p_2$ to recover p_1 and p_2.*
- 但是，給定$N = p_1 \times p_2$和p_1或p_2中的任何一個，透過簡單的除法就可以直接恢復另一個。
- *But given $N = p_1 \times p_2$ and either p_1 or p_2, it is straightforward to recover the other by dividing.*

加密效率 / Encryption Efficiency

- 公開金鑰系統在很大程度上解決了金鑰分發問題。

- *Public key systems **largely solve the key distribution problem**.*
- 但是公開金鑰加密可能比對稱加密**慢10,000倍**。
- *But public key encryption may take **10,000 times as long** to perform as symmetric encryption.*
- 計算依賴於更複雜的操作，而不是簡單的位元操作。
- *The computation depends on **more complex operations**, not on simple bit-wise operations.*
- **對稱加密**仍然是商業密碼學的**主力**。
- ***Symmetric encryption** remains the **work horse** of commercial cryptography.*
- 公開金鑰加密扮演一些重要的特殊功能。
- *With public key encryption playing some **important special functions**.*

公開金鑰加密的常用功能 / Common Functions of Public Key Encryption

1. **金鑰分發**
   - ***Key distribution***
2. **數位簽章** - 用於真實性和不可否認性：
   - ***Digital signature** - for authenticity and non-repudiation:*
   - $Sig \leftarrow Sign_K(M)$ ，其中 $Verify_{PK}(M, Sig) = 1$
   - $Sig \leftarrow Sign_K(M)$, *where* $Verify_{PK}(M, Sig) = 1$
   - 且對於所有 $M' \neq M$ ， $Verify_{PK}(M', Sig) = 0$
   - *and* $Verify_{PK}(M', Sig) = 0$ *for all* $M' \neq M$
3. **從交換的公開金鑰派生金鑰**
   - ***Key derivation** from exchanged public keys*
4. **Diffie-Hellman協議** - 在公共通道上建立雙方之間的共享秘密
   - ***Diffie-Hellman protocol** - to establish a mutual secret between two parties over public channel*

實踐中的必需： 公開金鑰基礎建設 - 可信地建立和管理數位憑證，用於驗證特定PK屬於某個實體。
*In practice, a **Public Key Infrastructure (PKI)** is also a must - trusted to create and manage digital certificates to verify that a particular PK belongs to a certain entity.*

加密的三種狀態 / Three States of Encryption

加密可用於保護三種狀態的資料：
Encryption can be used to protect data in three states:

1. 靜態加密 / Encryption at Rest

- 在儲存時加密資料，保護資料免受系統洩露或資料竊取。
- *Protects data from system compromise or data exfiltration by encrypting data while **stored**.*
- **常用技術：** AES通常用於靜態資料加密。
- ***Common technique:** AES is often used to encrypt data at rest.*

2. 傳輸中加密 / Encryption in Transit

- 當資料在兩個通訊實體之間移動時，如果通訊被攔截，保護資料。
- *Protects data if communications are intercepted while data **moves between** two communicating entities.*
- **常用技術：** TLS（傳輸安全）、S/MIME（電子郵件安全）
- ***Common techniques:** TLS for transport security, S/MIME for email*

3. 使用中加密 / Encryption in Use

- 在被伺服器用於執行計算時保護資料。
- *Protects data when being **used by servers** to run computations.*
- **常用技術：** 同態加密
- ***Common technique:** Homomorphic encryption*

Google雲加密範例 / Google Cloud Encryption Example

靜態加密的目的 / Purpose of Encryption at Rest

- **增加防禦深度**以保護資料
- ***Adds a layer of defense in depth** for protecting data*
- 攻擊者沒有加密金鑰也無法存取資料
- *Attackers cannot access data without also having access to **encryption keys**.*
- **減少攻擊面**，有效"切斷"硬體和軟體堆疊的較低層
- ***Reduces attack surface** by effectively "cutting out" lower layers of hardware/software stack*

- 即使較低層被破壞，充分加密的資料也不會被破壞
- *Even if lower layers are compromised, **adequately encrypted data is not compromised**.*
- **透過金鑰管理**強制執行資料存取並可以稽核
- ***Through key management**, access to data is enforced and can be audited.*
- **確保客戶資料的隱私**
- ***Ensures privacy of customer data***

客戶資料類型 / Customer Data Types

客戶內容： 靜態加密
***Customer content:** Encrypted at rest*

- 客戶自己產生或提供給Google的資料
- *Data that customers generate themselves or provide to Google*
- **例子：** 儲存在雲端儲存的資料、計算引擎使用的磁碟快照、身份和存取管理策略
- *Examples: Data in Cloud Storage, disk snapshots in Compute Engine, IAM policies*

客戶元資料： 在合理的程度上保護以進行持續效能/操作
***Customer metadata:** Protected to a degree reasonable for ongoing performance/operations*

- 指所有不能分類為客戶內容的資料
- *Refers to all data that cannot be classified as customer content*
- **例子：** 自動產生的專案編號、時間戳記、IP位址、雲端儲存中物件的位元組大小、計算引擎中的機器類型
- *Examples: Auto-generated project numbers, timestamps, IP addresses, object sizes in Cloud Storage, machine types in Compute Engine*

多層加密 / Multiple Layers of Encryption

Google雲中使用**多層加密**來保護儲存的資料：
*Several **layers of encryption** are used to protect data stored in Google Cloud:*

- 為應用程式需求新增冗餘的資料保護與靈活選項
- *Adds redundant data protection with flexible options for application requirements*

實際應用架構 / Practical Application Architecture

典型的雲安全架構：

Typical cloud security architecture:

```
應用層加密 → 資料庫加密 → 儲存加密 → 硬體加密
    ↓           ↓          ↓          ↓
應用程式      資料庫系統    儲存系統    實體磁碟
  使用應用金鑰  使用資料庫金鑰 使用儲存金鑰 使用硬體金鑰
```

優勢： 即使一個層被破壞，其他層仍然提供保護。
Advantage: *Even if one layer is compromised, other layers still provide protection.*

小測驗 / Test

1. 為什麼公開金鑰加密比對稱加密慢得多？這種效能差異在實踐中有何影響？
   - *Why is public key encryption much slower than symmetric encryption? How does this performance difference impact practical applications?*
2. 在Google雲的例子中，為什麼客戶內容和客戶元資料受到不同級別的保護？
   - *In the Google Cloud example, why are customer content and customer metadata protected at different levels?*
3. 靜態加密如何"減少攻擊面"並"切斷較低層"？
   - *How does encryption at rest "reduce the attack surface" and "cut out lower layers"?*
4. 為什麼在實踐中需要公開金鑰基礎建設？它解決了公開金鑰加密中的什麼問題？
   - *Why is a Public Key Infrastructure needed in practice? What problem does it solve in public key encryption?*

答案與解析 / Answers and Explanations

1. **公開金鑰加密的效能問題**

   - **答案：** 公開金鑰加密較慢是因為它基於**複雜的數學運算**（如大數模冪運算、橢圓曲線運算），而對稱加密使用**簡單的位元操作**（XOR、查表、置換）。這種效能差異導致：
     - **混合加密系統**：在實際系統中，公開金鑰加密僅用於交換對稱金鑰，然後使用對稱加密處理大量資料。
     - **功能分工**：公開金鑰加密專門用於金鑰交換和數位簽章，對稱加密用於批量資料加

密。
    - **效能最佳化**：需要高效能的應用（如視訊串流、大資料處理）主要依賴對稱加密。
  - *Answer: Public key encryption is slower because it's based on **complex mathematical operations** (like modular exponentiation, elliptic curve operations), while symmetric encryption uses **simple bit-wise operations** (XOR, table lookups, permutations). This performance difference leads to:*
    - ***Hybrid encryption systems:** In practice, public key is used only to exchange symmetric keys, then symmetric encryption handles bulk data.*
    - ***Functional specialization:** Public key specializes in key exchange and digital signatures, symmetric in bulk data encryption.*
    - ***Performance optimization:** High-performance applications (video streaming, big data) primarily rely on symmetric encryption.*

2. **內容與元資料的保護差異**
  - **答案：** 客戶內容和元資料受到不同保護級別，因為：
    - **敏感性不同**：客戶內容包含實際的使用者資料（文件、圖片、資料庫記錄），具有高敏感性和隱私要求。
    - **營運需求**：元資料（時間戳記、大小、類型）需要被系統元件存取以進行正常執行、監控和計費。
    - **實用性平衡**：對每個元資料操作都進行完全加密會嚴重影響系統效能和可操作性。
    - **風險分析**：元資料洩露的風險通常低於實際內容洩露的風險。
  - *Answer: Customer content and metadata are protected differently because:*
    - ***Different sensitivity:** Customer content contains actual user data (documents, images, database records) with high sensitivity and privacy requirements.*
    - ***Operational needs:** Metadata (timestamps, sizes, types) needs to be accessible by system components for normal operation, monitoring, and billing.*
    - ***Practical balance:** Fully encrypting every metadata operation would severely impact system performance and operability.*
    - ***Risk analysis:** The risk of metadata exposure is generally lower than actual content exposure.*

3. **靜態加密的攻擊面減少**
  - **答案：** 靜態加密透過以下方式減少攻擊面：
    - **抽象保護**：即使攻擊者獲得實體存取、破壞作業系統或儲存系統，他們仍然需要加密金鑰才能存取資料。

        - **分層防禦**：在應用程式層加密意味著較低層（作業系統、虛擬機器監控程式、儲存系統）的妥協不會自動導致資料洩露。
        - **最小權限**：系統管理員、雲端供應商工程師和備份系統可以在不接觸明文資料的情況下執行職責。
        - **遏制洩露**：如果發生資料洩露，加密的資料對攻擊者無用。
    - *Answer: Encryption at rest reduces attack surface by:*
        - ***Abstract protection:*** *Even if attackers gain physical access, compromise the OS, or storage systems, they still need encryption keys to access data.*
        - ***Layered defense:*** *Encrypting at application layer means compromises at lower layers (OS, hypervisor, storage) don't automatically lead to data exposure.*
        - ***Least privilege:*** *System administrators, cloud engineers, and backup systems can perform duties without accessing plaintext data.*
        - ***Contain breaches:*** *If data is exfiltrated, encrypted data is useless to attackers.*

4. **公開金鑰基礎建設的必要性**
    - **答案：** PKI解決了**公開金鑰認證問題**：
        - **信任建立**：如何確保你獲得的公開金鑰確實屬於聲稱擁有它的人？
        - **防止中間人攻擊**：沒有PKI，攻擊者可以提供他們自己的公開金鑰冒充合法方。
        - **身份綁定**：PKI透過數位憑證將公開金鑰與身份資訊綁定。
        - **撤銷機制**：PKI提供憑證撤銷清單等機制來處理洩露或過期的金鑰。
    - *Answer: PKI solves the **public key authentication problem**:*
        - ***Trust establishment:*** *How to ensure the public key you obtain genuinely belongs to who it claims?*
        - ***Prevent MITM attacks:*** *Without PKI, attackers can provide their own public keys impersonating legitimate parties.*
        - ***Identity binding:*** *PKI binds public keys to identity information through digital certificates.*
        - ***Revocation mechanism:*** *PKI provides mechanisms like certificate revocation lists to handle compromised or expired keys.*

Encryption at the Storage System Layer

encryption at the storage system layer shows how theoretical cryptographic concepts are applied in large-scale cloud infrastructure.

中英雙語解釋 / Bilingual Explanation

儲存系統層加密概述 / Encryption at the Storage System Layer

在儲存系統層，資料被分解為**子檔案區塊**進行儲存：
*Data is broken into **subfile chunks** for storage:*

- 每個區塊可以達到幾GB的大小
- *Each chunk can be up to **several GB** in size*
- 每個區塊在儲存層級使用**單獨的加密金鑰**進行加密
- *Each chunk is encrypted at the storage level with an **individual encryption key***
- 兩個區塊不會具有相同的加密金鑰
- *Two chunks will **not have the same encryption key***

金鑰管理策略 / Key Management Strategy

金鑰隔離原則：
Key Isolation Principle:

- 潛在的資料加密金鑰洩露**僅限於該區塊**
- *A potential data encryption key compromise is **limited to only that chunk***
- 如果資料區塊被更新，它使用**新金鑰**進行加密
- *If a chunk is updated, it is encrypted with a **new key***
- **不重用**現有金鑰
- ***No reuse** of the existing key*

加密時機： 在資料寫入磁碟之前進行加密。
Encryption is inherent in the storage system, rather than added on afterward.

存取控制與分發 / Access Control and Distribution

唯一識別符： 每個資料區塊都有唯一識別符。
*Each data chunk has a **unique identifier**.*

存取控制清單： 確保每個區塊只能由在授權角色下操作的Google服務在特定時間點解密。
*Access control lists ensure each chunk can be decrypted only by services operating under **authorized roles** at that point in time.*

- 防止未經授權存取資料，增強資料安全和隱私
- *Prevents unauthorized access, bolstering both data security and privacy*

分散式儲存： 每個區塊分佈在儲存系統中，並以加密形式複製用於備份和災難恢復。
*Each chunk is **distributed across storage systems** and **replicated in encrypted form** for backup and disaster recovery.*

攻擊者挑戰 / Attacker Challenges

惡意個體想要存取客戶資料需要：
A malicious individual would need:

- 與他們想要的資料對應的**所有儲存區塊**
- ***All storage chunks** corresponding to the data they want*
- 與這些區塊對應的**加密金鑰**
- ***The encryption keys** corresponding to the chunks*

加密演算法和模式 / Encryption Algorithms and Modes

預設使用AES-256：
AES-256 by default:

- 由美國國家標準與技術研究院推薦用於長期儲存使用
- *Recommended by NIST for **long-term storage** use*
- 客戶合規要求的一部分
- *Part of **customer compliance requirements***

主要採用的操作模式：
Majorly adopted modes of operations:

1. **AES-GCM** - 在幾乎所有情況下用於跨雲端儲存儲存的資料
   - ***AES-GCM** - in almost all cases for data stored across Cloud Storage*
2. **AES-CBC + HMAC** - 在選定情況下用於認證
   - ***AES-CBC + HMAC** - in selected cases for authentication*
3. **AES-CTR + HMAC** - 用於某些複製檔案
   - ***AES-CTR + HMAC** - for some replicated files*

詳細架構分析 / Detailed Architecture Analysis

資料區塊加密流程 / Data Chunk Encryption Process

```
原始檔案 → 分割為區塊 → 為每個區塊產生唯一金鑰 → 加密每個區塊 → 分散式儲存
    ↓          ↓              ↓              ↓          ↓
大型檔案    多個GB大小的區塊   每個區塊獨立金鑰   AES-256加密  跨多個儲存節點
```

金鑰層次結構 / Key Hierarchy

典型的雲端儲存金鑰管理：
Typical cloud storage key management:

- **主金鑰 → 金鑰加密金鑰 → 資料加密金鑰**
- ***Master Key** → Key Encryption Keys → **Data Encryption Keys***
- **每個資料區塊**有自己的資料加密金鑰
- ***Each data chunk** has its own data encryption key*
- 金鑰加密金鑰用於保護資料加密金鑰
- *Key encryption keys protect data encryption keys*
- 主金鑰儲存在硬體安全模組中
- *Master keys stored in Hardware Security Modules*

安全優勢分析 / Security Advantages Analysis

1. 金鑰隔離 / Key Isolation

- **限制洩露範圍：** 如果一個金鑰被洩露，隻影響一個資料區塊（幾GB），而不是整個檔案（可能TB級別）
- ***Limited exposure:** One compromised key affects only one chunk (GBs), not entire files (potentially TBs)*
- **細粒度存取控制：** 可以對單個資料區塊實施不同的存取策略
- ***Fine-grained access control:** Different access policies per chunk*

2. 金鑰輪換 / Key Rotation

- **更新時重新加密：** 當資料修改時自動使用新金鑰
- ***Re-encryption on update:** Automatic new key when data is modified*

- **前向安全：** 即使舊金鑰被洩露，新資料仍然安全
- ***Forward secrecy:*** *New data remains secure even if old keys are compromised*

3. 分散式保護 / Distributed Protection

- **資料分散：** 檔案被分成多個區塊分佈在不同實體位置
- ***Data dispersion:*** *Files split into chunks across different physical locations*
- **防禦深度：** 攻擊者需要同時取得多個元件
- ***Defense in depth:*** *Attackers need multiple components simultaneously*

操作模式選擇原理 / Operation Mode Selection Rationale

為什麼主要使用AES-GCM？ / Why Primarily Use AES-GCM?

- **認證加密：** 同時提供保密性和完整性
- ***Authenticated encryption:*** *Both confidentiality and integrity*
- **高效能：** 可以平行化處理
- ***High performance:*** *Can be parallelized*
- **標準化：** 經過充分研究和廣泛採用
- ***Standardized:*** *Well-studied and widely adopted*
- **NIST推薦：** 符合政府和企業標準
- ***NIST recommended:*** *Meets government and enterprise standards*

為什麼在某些情況下使用AES-CBC + HMAC？ / Why AES-CBC + HMAC in Selected Cases?

- **遺留系統相容性**
- ***Legacy system compatibility***
- **特定效能要求**
- ***Specific performance requirements***
- **特殊認證需求**
- ***Special authentication needs***

為什麼對複製檔案使用AES-CTR + HMAC？ / Why AES-CTR + HMAC for Replicated Files?

- **平行處理能力**
- ***Parallel processing capability***

- **串流加密友好**
- ***Streaming encryption friendly***
- **高效複製操作**
- ***Efficient replication operations***

合規性與標準 / Compliance and Standards

NIST推薦 / NIST Recommendations

- **AES-256：** 用於保護敏感資料的長期儲存
- ***AES-256:*** *For protecting sensitive data in long-term storage*
- **FIPS 140-2/3：** 加密模組的聯邦資訊處理標準
- ***FIPS 140-2/3:*** *Federal standards for cryptographic modules*
- **NIST特別出版物800-57：** 金鑰管理建議
- ***NIST SP 800-57:*** *Key management recommendations*

行業合規 / Industry Compliance

- **HIPAA：** 醫療保健資料保護
- ***HIPAA:*** *Healthcare data protection*
- **PCI DSS：** 支付卡行業資料安全標準
- ***PCI DSS:*** *Payment card industry standards*
- **GDPR：** 通用資料保護條例
- ***GDPR:*** *General Data Protection Regulation*
- **SOC 2：** 服務組織控制
- ***SOC 2:*** *Service organization controls*

小測驗 / Test

1. 為什麼儲存系統要將大檔案分解成多個區塊並為每個區塊使用不同的加密金鑰？
    - *Why does the storage system break large files into multiple chunks and use different encryption keys for each chunk?*
2. 當資料區塊被更新時使用新金鑰有什麼安全優勢？
    - *What security advantages are gained by using a new key when a data chunk is*

updated?

3. AES-GCM、AES-CBC+HMAC和AES-CTR+HMAC分別適用於什麼場景？為什麼？
   - *What scenarios are AES-GCM, AES-CBC+HMAC, and AES-CTR+HMAC respectively suitable for? Why?*
4. 攻擊者想要竊取加密的客戶資料面臨哪些具體挑戰？
   - *What specific challenges does an attacker face when trying to steal encrypted customer data?*

答案與解析 / Answers and Explanations

1. **分塊加密的優勢**
   - **答案：** 分塊加密提供了**金鑰隔離**和**洩露控制**：
     - **限制洩露範圍**：如果一個金鑰被洩露，隻影響一個資料區塊（幾GB），而不是整個檔案（可能TB級別）。
     - **細粒度存取控制**：可以對不同的資料區塊實施不同的存取策略和權限。
     - **平行處理**：可以同時加密/解密多個區塊，提高效能。
     - **分散式儲存最佳化**：適合在分散式系統中儲存和管理。
     - **增量更新效率**：只需要重新加密修改的區塊，而不是整個檔案。
   - *Answer: Chunk-based encryption provides **key isolation** and **exposure control**:*
     - ***Limited exposure:** One compromised key affects only one chunk (GBs), not entire files (potentially TBs).*
     - ***Fine-grained access control:** Different access policies and permissions can be applied to different chunks.*
     - ***Parallel processing:** Multiple chunks can be encrypted/decrypted simultaneously for better performance.*
     - ***Distributed storage optimization:** Fits well with distributed system storage and management.*
     - ***Incremental update efficiency:** Only modified chunks need re-encryption, not the entire file.*

2. **金鑰輪換的安全優勢**
   - **答案：** 更新時使用新金鑰提供：
     - **前向保密性**：即使舊金鑰被洩露，新加密的資料仍然安全。
     - **限制金鑰暴露時間**：每個金鑰隻在特定時間段內有效。

- **破壞攻擊者累積**：防止攻擊者透過觀察多個加密版本累積金鑰資訊。
- **符合加密最佳實踐**：遵循"每個加密操作使用新鮮金鑰"的原則。
- **簡化金鑰撤銷**：可以獨立地撤銷特定版本的金鑰而不影響其他資料。

- *Answer: Using new keys on update provides:*
  - ***Forward secrecy:*** *Newly encrypted data remains secure even if old keys are compromised.*
  - ***Limited key exposure time:*** *Each key is valid only for a specific time period.*
  - ***Break attacker accumulation:*** *Prevents attackers from accumulating key information by observing multiple encrypted versions.*
  - ***Cryptographic best practices:*** *Follows the principle of "use fresh keys for each encryption operation."*
  - ***Simplified key revocation:*** *Specific version keys can be revoked independently without affecting other data.*

3. **加密模式的適用場景**

- **答案：**
  - **AES-GCM**：**主要選擇**，適用於需要同時保證保密性和完整性的大多數場景。它高效、標準化且經過充分研究。
  - **AES-CBC + HMAC**：用於**遺留系統相容**或需要**獨立控制加密和認證**的場景。HMAC提供強認證，但效能開銷較大。
  - **AES-CTR + HMAC**：適用於需要**平行處理**和**串流加密**的複製檔案。CTR模式支援隨機存取，適合分散式複製操作。
- *Answer:*
  - ***AES-GCM: Primary choice*** *for most scenarios needing both confidentiality and integrity. It's efficient, standardized, and well-studied.*
  - ***AES-CBC + HMAC:*** *For* ***legacy system compatibility*** *or scenarios needing* ***independent control*** *of encryption and authentication. HMAC provides strong authentication but with higher performance overhead.*
  - ***AES-CTR + HMAC:*** *For* ***parallel processing*** *and* ***streaming encryption*** *in replicated files. CTR mode supports random access, suitable for distributed replication operations.*

4. **攻擊者面臨的挑戰**

- **答案：** 攻擊者需要克服多個層次的防禦：
  - **資料分散**：檔案被分成多個區塊分佈在不同實體位置。

        - **金鑰隔離**：每個區塊使用不同的加密金鑰。
        - **存取控制**：需要繞過基於角色的存取控制系統。
        - **加密強度**：需要破解AES-256加密。
        - **金鑰管理**：需要取得金鑰層次結構中的多個金鑰。
        - **偵測風險**：大規模資料存取嘗試可能觸發安全監控和警報。
    - *Answer: Attackers face multiple layers of defense:*
        - ***Data dispersion:*** *Files are split into chunks across different physical locations.*
        - ***Key isolation:*** *Each chunk uses different encryption keys.*
        - ***Access control:*** *Need to bypass role-based access control systems.*
        - ***Encryption strength:*** *Need to break AES-256 encryption.*
        - ***Key management:*** *Need to obtain multiple keys in the key hierarchy.*
        - ***Detection risk:*** *Large-scale data access attempts may trigger security monitoring and alerts.*

Encryption at the Storage Device Layer and Backups

中英雙語解釋 / Bilingual Explanation

儲存裝置層加密 / Encryption at the Storage Device Layer

- **定義：** 除了在儲存系統層級進行加密外，在大多數情況下，資料也會在儲存裝置（如硬碟、固態硬碟）本身這一級別進行加密。
- **Definition:** In addition to the storage system level encryption described above, in most cases data is also encrypted at the storage device level (e.g., HDD, SSD).
- **目的：** 提供額外的安全防護層。即使實體硬碟被移出或遺失，沒有金鑰也無法讀取其中的資料。這與儲存系統層的加密相輔相成，構成了深度防禦策略。
- **Purpose:** To provide an additional layer of security. Even if a physical drive is removed or lost, the data cannot be read without the key. This complements storage system level encryption, forming a defense-in-depth strategy.

技術細節 / Technical Details

- **加密演算法：** 硬碟與固態硬碟使用 **AES256** 加密演算法。

    - *Uses the **AES256** encryption algorithm for hard disks (HDD) and solid state drives (SSD).*
    - **註：** 少量舊版硬碟使用 AES128。
    - *Note: A small number of legacy HDDs use AES128.*
- **金鑰管理：** 使用獨立的**裝置層級金鑰**。
    - *Uses a separate **device-level key**.*
    - **重要：** 此金鑰與用於在「儲存系統層級」加密資料的金鑰**不同**。這種分離增強了安全性。
    - *Important: This key is **different from the key** used to encrypt the data at the storage system level. This separation enhances security.*
- **Google Cloud 實作：** Google Cloud 所使用的固態硬碟，其 AES256 加密**專門用於使用者資料**。
    - *The SSDs used by Google Cloud implement AES256 for user data **exclusively**.*

備份過程中的加密 / Encryption During Backups

- **核心原則：** 資料在**整個備份過程**中持續保持加密狀態。
    - *Data remains encrypted **throughout the backup** process.*
- **目的：** 避免在備份時不必要地暴露明文資料，確保資料從來源到備份目的地的安全性。
    - *Avoids unnecessarily exposing plaintext data during backup, ensuring data security from source to backup destination.*

備份系統的加密機制 / Backup System Encryption Mechanism

備份系統本身會實施額外的加密層：
The backup system itself implements an additional layer of encryption:

1. **備份檔案加密：**
    - **解釋：** 備份系統會使用獨立的**資料加密金鑰**，對每個備份檔案進行額外的獨立加密。
    - **Explanation:** The backup system further encrypts each backup file independently with its own **Data Encryption Key (DEK)**.

- **金鑰衍生方式：** 此 DEK 衍生自兩個來源：
  - *This DEK is derived from two sources:*
  - 儲存在 **Google 金鑰管理服務** 中的一個主金鑰。
    - *A key stored in **Google's Key Management Service (KMS)**.*
  - 在備份時**隨機生成**的「每個檔案專屬」的種子值。
    - *A **randomly generated per-file seed** at backup time.*

2. **備份詮釋資料加密：**

   - **解釋：** 備份中的所有詮釋資料（例如檔案名稱、路徑、大小等）會使用**另一個獨立的 DEK** 進行加密。
   - **Explanation:** Another, separate **DEK is used for all metadata** in backups (e.g., file names, paths, sizes).
   - **金鑰儲存：** 此詮釋資料的 DEK 也儲存在 Google 的 KMS 中。
   - **Key Storage:** This metadata DEK is also stored in Google's KMS.

總結 / Summary

這種多層次的加密方法（儲存裝置層 + 儲存系統層 + 備份層）確保了資料：
This multi-layered encryption approach (Device Layer + Storage System Layer + Backup Layer) ensures that data is protected:

- **靜態保護：** 無論是主要儲存還是備份儲存中的資料，都處於加密狀態。
  - *At rest: Both in primary storage and in backup storage.*
- **過程保護：** 即使在備份傳輸和處理過程中，資料也不會以明文形式暴露。
  - *In process: Even during backup transfer and processing, data is not exposed in plaintext.*
- **深度防禦：** 即使某一層的金鑰被破解，其他層的加密仍然能提供保護。
  - *With defense-in-depth: If a key from one layer is compromised, the encryption from other layers still provides protection.*

小測試 / Quick Test

問題： 一家公司考慮將其資料備份到雲端。他們擔心即使雲端供應商的實體硬碟被退役或轉售，他們的資料是否仍可能被下一個持有者讀取。根據上述加密模型，這個風險是如何被緩解的？

Question: A company is considering backing up its data to the cloud. They are concerned about whether their data could be read by the next owner if the cloud provider's physical drives are decommissioned or resold. How is this risk mitigated according to the encryption model described above?

1. **識別防護層：**
   - 實體硬碟（HDD/SSD）本身已使用 **AES256** 加密。
   - *The physical drives (HDD/SSD) themselves are encrypted using **AES256**.*
2. **識別關鍵機制：**
   - 加密使用的是獨立的**裝置層級金鑰**，且此金鑰與儲存系統的金鑰不同。
   - *The encryption uses a separate **device-level key**, which is different from the storage system key.*
3. **推論：**
   - 當硬碟離開 Google 的基礎設施時，與其關聯的裝置層級金鑰會被安全地銷毀。沒有此金鑰，硬碟上的資料只是無法讀取的加密資料。
   - *When a drive leaves Google's infrastructure, the device-level key associated with it is securely destroyed. Without this key, the data on the drive is unreadable encrypted data.*

答案： 透過在儲存裝置層實施硬體加密，並由雲端供應商嚴格管理金鑰，即使實體硬碟被轉移，其中的資料也因為沒有對應的金鑰而無法被解讀，從而保護了資料的隱私。
Answer: By implementing hardware encryption at the storage device layer and with the cloud provider's strict key management, even if the physical drive is transferred, the data on it remains encrypted and unreadable without the corresponding key, thus protecting data privacy.

Key Management at Google Cloud

中英雙語解釋 / Bilingual Explanation

資料加密金鑰 vs 金鑰加密金鑰 / Data Encryption Keys vs Key Encryption Keys

- **定義：** 用於加密資料區塊的金鑰稱為**資料加密金鑰**。
  - **Definition:** The key used to encrypt the data chunk is called a **Data Encryption Key (DEK)**.
- **特性：**
  - 這些 DEK 儲存在它們所加密的資料附近。

- *These DEKs are stored near the data that they encrypt.*
- 這些 DEK 本身會被一把**金鑰加密金鑰** 所加密（或稱為「包裹」）。
- *These DEKs are encrypted with (or "wrapped" by) a **Key Encryption Key (KEK)**.*

金鑰加密金鑰 / Key Encryption Keys (KEK)

- **規模管理：**
  - 每個 Google Cloud 服務都擁有一個或多個 KEK。
  - *One or more KEKs exist for each Google Cloud service.*
  - KEK 的數量遠少於 DEK 的數量，這種設計使得大規模儲存和加密資料變得易於管理。
  - *There are a smaller number of KEKs than DEKs. The design makes storing and encrypting data at scale manageable.*
- **集中儲存：**
  - 這些 KEK 集中儲存在 **Google 金鑰管理服務** 中，這是一個專門為儲存金鑰而構建的儲存庫。
  - *These KEKs are stored centrally in **Google's KMS**, a repository built specifically for storing keys.*
- **存取控管與追蹤：**
  - 此設計允許追蹤和控制資料存取。
  - *This design allows tracking and controlling data access.*

金鑰生成 / Key Generation

- **方法：** 所有金鑰均使用 Google 的通用密碼庫生成。
  - *All keys are generated using Google's common cryptographic library.*
- **隨機性來源：**
  - 使用由 Linux 核心隨機數生成器播種的偽隨機數生成器，該生成器又從多個獨立的熵源獲取種子：
  - *Using a PRNG seeded from the Linux kernel's RNG, which in turn is seeded from multiple independent entropy sources:*
    - 來自資料中心環境的熵事件，例如磁碟尋道時間和數據包到達間隔時間的細粒度測量。

    - *Entropic events from the data center environment, such as fine-grained measurements of disk seeks and inter-packet arrival times.*
    - 在可用的情況下（Ivy Bridge 及更新的 CPU）使用 Intel 的 RDRAND 指令。
    - *And Intel's RDRAND instruction where it is available (on Ivy Bridge and newer CPUs).*

KEK 的安全設計與生命週期管理 / KEK Security Design & Lifecycle Management

- **不可匯出性：**
    - 所有 KEK **依設計**無法從 Google 的 KMS 中匯出。
    - *All KEKs are **not exportable** from Google's KMS by design.*
    - 使用這些金鑰進行的所有加密和解密操作都必須在 KMS 內部完成。
    - *All encryption and decryption with these keys must be done within KMS.*
    - **目的：** 防止金鑰洩露和濫用，並使 KMS 能夠在金鑰使用時發出審計紀錄。
    - *Purpose: To prevent leaks and misuse, and enables KMS to emit an audit trail when keys are used.*
- **自動輪替：**
    - KMS 可以按固定的時間間隔自動輪替 KEK。
    - *KMS can automatically rotate KEKs at regular time intervals.*
    - **實作方式：** 資料使用一個金鑰集合進行保護：一個用於加密的現用金鑰，以及一組用於解密的历史金鑰，具體由金鑰輪替排程決定。
    - *Implementation: Data is protected using a key set: one key active for encryption and a set of historical keys for decryption, determined by the key rotation schedule.*
    - **輪替頻率：** 輪替頻率因服務而異。
        - *The rotation frequency varies by services.*
        - **Cloud Storage** 具體會每 90 天輪替其 KEK，最多可儲存 20 個版本，這要求資料至少每五年重新加密一次。
        - *Cloud Storage specifically rotates its KEKs every 90 days, and can store up to 20 versions, requiring re-encryption of data at least once every five years.*
    - **災難復原：** 為實現災難復原目的，KMS 持有的金鑰會進行備份。
        - *KMS-held keys are backed up for disaster recovery purposes.*

KEK 與存取控制 / KEKs and Access Control Lists (ACL)

- **控管方式：** KEK 的使用由 KMS 中每個金鑰的**存取控制清單** 進行管理，每個金鑰都有專屬的政策。
  - *The use of KEKs is managed by **Access Control Lists (ACLs)** in KMS for each key, with a per-key policy.*
- **授權與記錄：**
  - 只有經過授權的 Google 服務和使用者才被允許存取金鑰。
  - *Only authorized Google services and users are allowed access to a key.*
  - 每個金鑰的使用都會在需要該金鑰的個別操作層級上被追蹤。
  - *The use of each key is tracked at the level of individual operations that requires that key.*
  - 每次個人使用者使用金鑰時，都會經過身份驗證並被記錄下來。
  - *Every time an individual uses a key, it is authenticated and logged.*
- **審計性：** 所有資料存取都是可審計的，這作為 Google 整體安全與隱私政策的一部分。
  - *All data accesses are auditable as part of Google's overall security and privacy policies.*

資料解密流程範例 / Data Decryption Flow Example

當一個 Google Cloud 服務需要存取一個加密的資料區塊時：
When a Google Cloud service accesses an encrypted chunk:

1. **服務請求：** 該服務向儲存系統呼叫它所需的資料；儲存系統識別出儲存該資料的區塊（區塊 ID）。
   - *The service makes a call to the storage system for the data it needs; the storage system identifies the chunks in which that data is stored (the chunk IDs).*
2. **取得被包裹的 DEK：** 對於每個區塊，儲存系統會讀取與該區塊儲存在一起的、已被包裹的 DEK。
   - *For each chunk, the storage system pulls the wrapped DEK stored with that chunk.*
3. **請求 KMS 解開包裹：** 儲存系統將被包裹的 DEK 發送給 KMS 以請求解開包裹。
   - *The storage system sends it to KMS for unwrapping.*
4. **雙重驗證：**
   - 儲存系統基於作業標識符和區塊 ID，驗證被識別的作業是否被允許存取該資料區塊。
   - *The storage system verifies that the identified job is allowed to access that data chunk based on a job identifier, and using the chunk ID.*

    - KMS 驗證該儲存系統不僅有權使用與該服務關聯的 KEK，並且有權解開該特定 DEK 的包裹。
    - *KMS verifies that the storage system is authorized both to use the KEK associated with the service, and to unwrap that specific DEK.*
5. **解密資料：** KMS 將解開包裹後的 DEK 傳回給儲存系統，儲存系統使用該 DEK 解密資料區塊，然後將資料傳遞給服務。
    - *KMS passes the unwrapped DEK back to the storage system, which decrypts the data chunk and passes it to the service.*

總結： 為了解密一個資料區塊，儲存服務必須呼叫 Google 的 KMS 來檢索該資料區塊解開包裹後的資料加密金鑰。
Summary: To decrypt a data chunk, the storage service calls Google's KMS to retrieve the unwrapped Data Encryption Key (DEK) for that data chunk.

小測試 / Quick Test

問題： 一個開發團隊在 Google Cloud 上使用 Cloud Storage。他們想知道，當 KMS 自動輪替 KEK 後，他們是否需要手動下載新的金鑰或重新加密他們所有的資料？為什麼？
Question: A development team using Cloud Storage on Google Cloud wants to know if they need to manually download new keys or re-encrypt all their data when KMS automatically rotates the KEK. Why or why not?

1. **回憶機制：** KMS 的金鑰輪替機制使用一個金鑰集合，包含一個現用金鑰和多个历史金鑰。
    - *Recall the mechanism: KMS's key rotation uses a key set with one active key and a set of historical keys.*
2. **推論對加密的影響：**
    - **新資料：** 輪替後，*新* 的資料會使用新的現用 KEK 來包裹其 DEK。
    - *New Data: After rotation, *new* data will have its DEK wrapped by the new active KEK.*
    - **舊資料：** *現有* 的資料其 DEK 仍然使用輪替前的 KEK 版本進行包裹。KMS 會保留這些旧 KEK 版本用於解密。
    - *Existing Data: *Existing* data has its DEK wrapped by previous versions of the KEK. KMS retains these old KEK versions for decryption.*
3. **結論：**
    - 開發團隊**不需要**進行任何手動操作。
    - *The development team does **NOT** need to take any manual action.*

    - 整個金鑰輪替、新資料加密以及舊資料解密的過程都由 KMS 和 Cloud Storage 服務在後台自動無縫處理。
    - *The entire process of key rotation, encrypting new data, and decrypting old data is handled automatically and seamlessly in the background by KMS and the Cloud Storage service.*

答案： 不需要手動介入。因為 KMS 的金鑰輪替設計會同時維護現用與历史的 KEK，系統能自動使用正確的金鑰來加密新資料或解密舊資料，對使用者是完全透明的。
Answer: No manual intervention is required. Because KMS's key rotation design maintains both active and historical KEKs, the system automatically uses the correct key to encrypt new data or decrypt old data, which is completely transparent to the user.

Encryption key hierarchy and root of trust

中英雙語解釋 / Bilingual Explanation

加密金鑰階層 / Encryption Key Hierarchy

這是一個層層保護的結構，確保金鑰本身的安全，並最終保護您的資料。
This is a layered protection structure that ensures the security of the keys themselves, and ultimately, your data.

第一層：KMS 主金鑰 / Layer 1: KMS Master Key

- **功能：** Google 的 KMS 服務本身受到一個根金鑰的保護，該金鑰稱為 **KMS 主金鑰**，它負責包裹（加密）KMS 中的所有 KEK。
    - **Function:** Google's KMS service itself is protected by a root key called the **KMS master key**, which wraps (encrypts) all the KEKs in KMS.
- **規格與儲存：**
    - 此 KMS 主金鑰為 **AES256**。
    - *This KMS master key is **AES256**.*
    - 此 KMS 主金鑰儲存在另一個稱為 **根 KMS** 的金鑰管理服務中。
    - *This KMS master key is stored in another key management service, called **Root KMS**.*

第二層：根 KMS / Layer 2: Root KMS

- **特性：**

- 根 KMS 儲存的鍵數量要少得多——大約只有十幾個。
- *Root KMS stores a much smaller number of keys — approximately a dozen.*
- 它**不在一般的生產機器上運行**，而是在每個 Google 資料中心的**專用機器**上運行，以減少攻擊面。
- *It is **not run on general production machines**, but instead is run only on **dedicated machines** in each Google data centre.*

第三層：根 KMS 主金鑰 / Layer 3: Root KMS Master Key

- **功能：** 根 KMS 同樣擁有自己的根金鑰，稱為**根 KMS 主金鑰**。
  - **Function:** Root KMS in turn has its own root key, called the **root KMS master key**.
- **規格與儲存：**
  - 根 KMS 主金鑰也是 **AES256**。
  - *The root KMS master key is also **AES256**.*
  - 它儲存在一個稱為**根 KMS 主金鑰分散器**的點對點基礎設施中，該設施會在全球範圍內複寫這些金鑰。
  - *It is stored in a peer-to-peer infrastructure, called the **root KMS master key distributor**, which replicates these keys globally.*

第四層：根 KMS 主金鑰分散器 / Layer 4: Root KMS Master Key Distributor

- **安全設計：**
  - 根 KMS 主金鑰分散器**僅將金鑰保存在 RAM 中**，且與根 KMS 運行在同一批專用機器上。
  - *The root KMS master key distributor only holds the keys **in RAM** on the same dedicated machines as Root KMS.*
  - 它使用日誌記錄來驗證金鑰的正確使用。
  - *It uses logging to verify proper use.*
- **點對點運作模式：**
  - 當一個新的分散器實例啟動時，它會被配置一個正在運行的分散器實例的主機名清單。
  - *When a new instance is started, it is configured with a list of host names already running distributor instances.*
  - 然後，分散器實例可以從其他正在運行的實例獲取根 KMS 主金鑰。
  - *Distributor instances can then obtain the root KMS master key from other running instances.*

- 根 KMS 主金鑰僅存在於**數量有限且經過特殊加固的機器 RAM 中**。
- *The root KMS master key exists only in RAM on a limited number of specially secured machines.*

災難復原與實體備份 / Disaster Recovery & Physical Backup

- **備份方式：** 為了災難復原，根 KMS 主金鑰也會被備份。
  - *For disaster-recovery, the root KMS master key is also backed up.*
  - 備份儲存在**安全硬體設備**中，這些設備放置在兩個物理隔離的全球 Google 地點的**高度安全區域的實體保險庫**內。
  - *The backup is on **secure hardware devices** stored in **physical safes** in highly secured areas in two physically separated, global Google locations.*
- **啟動條件：**
  - 只有當**所有**分散器實例同時故障時（例如，在全球重啟的情況下），才需要此備份。
  - *This backup would be needed only if **all** distributor instances were to go down at once (e.g., in a global restart).*
- **存取控管：** 全球只有**少於 20 名** Google 員工能夠存取這些保險庫。
  - *Fewer than **20** Google employees are able to access these safes.*

全局可用性與複寫 / Global Availability and Replication

高可用性和低延遲的全局金鑰存取至關重要。
High availability and low latency, global access to keys, are critical at every level.

- **KMS：** 具有高度可擴展性，在 Google 的全球資料中心中複寫了數千次。
  - *KMS is highly scalable, and it is replicated **thousands of times** in Google's data centers globally.*
- **根 KMS 主金鑰分散器：** 使用**通訊協定**來同步金鑰。
  - *The root KMS master key distributor uses a **gossiping protocol**.*
  - **運作方式：** 在固定的時間間隔，每個分散器實例會隨機選擇另一個實例來比較其金鑰，並協調任何金鑰版本的差異。
  - *At a fixed time interval, each instance of the distributor picks a random other instance to compare its keys with and reconciles any differences.*
- **去中心化設計：** **不存在**一個所有 Google 基礎設施都依賴的中央節點；這使得 Google 能夠以高可用性維護和保護金鑰材料。

- *There is **no central node** that all of Google's infrastructure depends on; this allows Google to maintain and protect key material with high availability.*

經驗總結 / Lessons Learned

- **靜態加密**有助於保護客戶資料。
  - *Encryption at rest helps secure customer data.*
- 透過金鑰管理在**最細粒度層級**控制資料存取，確保資料安全與隱私。
  - *Control data access at the **finest level of granularity** through key management to ensure both data security and privacy.*
- **加密金鑰階層**確保了安全性、可用性和可擴展性。
  - *The **encryption key hierarchy** ensures security, availability, and scalability.*
- 所有金鑰使用都經過**記錄和審計**，提供了超越僅使用 ACL 的保護。
  - *All key use is **logged and auditable**, providing protection beyond the use of ACLs.*
- **去中心化設計**避免了單點故障。
  - *The **decentralized design (no central node)** avoids a single point of failure.*

小測試 / Quick Test

問題： 為什麼 Google 要設計一個如此複雜的多層次金鑰階層（DEK -> KEK -> KMS Master Key -> Root KMS Master Key），而不是將所有金鑰都集中儲存在一個超級安全的 KMS 中？這種階層式設計的主要優點是什麼？
Question: Why did Google design such a complex multi-layer key hierarchy (DEK -> KEK -> KMS Master Key -> Root KMS Master Key), instead of storing all keys in one super-secure KMS? What are the main advantages of this hierarchical design?

1. **安全性：**
   - 將最敏感的金鑰（根主金鑰）與大量日常使用的金鑰（KEK, DEK）**隔離**開來。
   - ***Isolates** the most sensitive keys (root master keys) from the high-volume operational keys (KEKs, DEKs).*
   - 最敏感的金鑰可以受到更嚴格、更實體的保護（如專用機器、RAM 僅存、實體保險庫）。
   - *The most sensitive keys can be protected with more stringent, physical measures (dedicated machines, RAM-only, physical safes).*
2. **可擴展性：**

    - 數十億的 DEK 可以由數千個 KEK 管理，而這些 KEK 又由少數幾個 KMS 主金鑰保護，最終由極少數的根主金鑰保護。這使得管理海量金鑰成為可能。
    - *Billions of DEKs can be managed by thousands of KEKs, which are protected by a few KMS master keys, ultimately secured by a handful of root master keys. This makes managing keys at a massive scale possible.*
3. **可用性與效能：**
    - 日常的資料加密/解密操作只需要呼叫儲存大量 KEK 的 KMS，而無需觸及更深層、吞吐量較低的根 KMS 系統，這保證了低延遲和高可用性。
    - *Daily data encryption/decryption operations only need to call the high-throughput KMS, without touching the deeper, lower-throughput Root KMS systems, ensuring low latency and high availability.*
4. **職責分離：**
    - 不同的團隊和系統可以負責金鑰階層中的不同層級，實作職責分離的安全原則。
    - *Different teams and systems can manage different layers of the hierarchy, implementing the security principle of **separation of duties**.*

答案： 這種階層設計的主要優點在於**平衡了安全性、可擴展性與可用性**。它透過隔離最關鍵的金鑰來最大化安全性，同時允許大量金鑰的管理與高效能的日常操作，並透過去中心化設計確保了系統的韌性。
*Answer: The main advantage of this hierarchical design is that it **balances security, scalability, and availability**. It maximizes security by isolating the most critical keys, while allowing for the management of a vast number of keys and high-performance daily operations. The decentralized design also ensures system resilience.*

Certificates and Web of Trust

中英雙語解釋 / Bilingual Explanation

信任網 / Web of Trust

- **概念：** 網路上許多活動，特別是電子商務，都依賴於在各方之間建立一個**信任網**。
    - **Concept:** Much of what happens on-line, particularly e-commerce, depends on establishing a **web of trust** relationships among the parties.
- **核心問題：** 為什麼 A 應該信任一個他從未打過交道的 B？
    - **Core Question:** Why should A trust B with whom he's never previously dealt?
- **可能的答案：** A 可以依賴一個已知的第三方來「擔保」B。
    - **Possible Answer:** A might rely on a known third party to "vouch for" B.

- **現實世界的例子：** 商會、商業改善局、信用報告機構、朋友等，在某種程度上都充當了某些商業交易的認證機構。
  - **Real-world Examples:** The Chamber of Commerce, Better Business Bureau, credit reporting agencies, friends, etc. all function in part as certification authorities for some commercial transactions.

對信任的需求 / Need for Trust

- **公開金鑰基礎建設的情境：** 在公開金鑰基礎建設中，如果 A 知道 B 的公開金鑰，那麼 A 可以：
  - **With a PKI:** If A knows B's public key, then A can:
  - 安全地發送訊息給 B。
    - *Send a message securely to B;*
  - 確信來自 B 的訊息確實源自 B。
    - *Be assured that a message from B really originated with B.*
- **關鍵問題：** 但是，A 如何知道 B 提供的公開金鑰**確實是 B 的**，而不是別人的？
  - **Key Problem:** But, how does A know that the public key B presents is **really B's public key** and not someone else's?
- **信任的綁定：** 在分散式線上環境中最需要信任的情況，就是**可靠地將公開金鑰與身分綁定在一起**。
  - **The Binding of Trust:** The most common circumstance in which trust is needed is **reliably binding a public key to an identity**.

憑證 / Certificates

- **定義：** 憑證是「介紹信」的電子等效物。
  - **Definition:** A certificate is the electronic equivalent of a "letter of introduction."
- **建構方式：** 憑證是使用**數位簽章**和**雜湊函式**構建的。
  - **Construction:** A certificate is constructed with **digital signatures** and **hash functions**.
- **核心功能：** 一個公開金鑰和使用者的身分被**綁定**在一個憑證中，並由**認證機構**簽署，以擔保此綁定的準確性。
  - **Core Function:** A public key and a user's identity are **bound together** within a certificate, **signed by a certification authority**, vouching for the accuracy of the binding.

運作方式範例 / How It Might Work

假設：
Suppose:

- *X* 是公司總裁；*Y* 是她的下屬。每人擁有一對 RSA 公開金鑰。
  - *X is the president of a company; Y is her subordinate. Each has an RSA public key pair.*

建立憑證的步驟：
Steps to create a certificate:

1. *Y* 安全地將訊息 `{Y, PK_Y}` 傳遞給 *X*。
   - *Y securely passes message `{Y, PK_Y}` to X.*
2. *X* 產生該訊息的密碼學雜湊值：`H({Y, PK_Y})`。
   - *X produces a cryptographic hash of the message: `H({Y, PK_Y})`.*
3. *X* 使用她的私密金鑰對該雜湊值進行簽署，產生 `Sign_SK_X ( H({Y, PK_Y}) )`。
   - *X signs this hash with her private key, producing `Sign_SK_X ( H({Y, PK_Y}) )`.*
4. *X* 將所有資訊組合起來，產生 `{Y, PK_Y, Sign_SK_X ( H({Y, PK_Y}) )}`。
   - *X produces `{Y, PK_Y, Sign_SK_X ( H({Y, PK_Y}) )`.*
5. 這最後的結果就成為由 *X* 簽署的 *Y* 的憑證。
   - *This last then becomes Y's certificate, signed by X.*

驗證憑證 / Validating the Certificate

假設 *Y* 向 *Z* 出示該憑證：
Suppose Y presents to Z the certificate:
`{Y, PK_Y, Sign_SK_X ( H({Y, PK_Y}) )}`

Z 會如何處理它？*Z* 能從中得知什麼？
What does Z do with this? What does Z learn?

1. **取得憑證與公開金鑰：** *Z* 首先需要取得他信任的 *X* 的公開金鑰。
   - *Z first needs to obtain X's public key (PK_X), which he trusts.*
2. **驗證簽章：** *Z* 使用 *X* 的公開金鑰來解密憑證中的數位簽章，從而得到 *X* 當初計算出的雜湊值 `H1`。
   - *Z uses PK_X to decrypt the digital signature in the certificate, obtaining the hash value `H1` that X originally computed.*

3. **計算雜湊值：** *Z* 自己對憑證中的 `{Y, PK_Y}` 部分進行雜湊計算，得到 `H2` 。
    - *Z independently computes the hash of the `{Y, PK_Y}` part from the certificate, obtaining `H2`.*
4. **比對與信任推導：** *Z* 比對 `H1` 和 `H2` 。
    - *Z compares `H1` and `H2`.*
    - 如果 `H1 == H2` ，則 *Z* 可以確信：
        - *If `H1 == H2`, then Z can be assured that:*
        - 此憑證擔保了 *Y* 和 `PK_Y` 的綁定關係。
            - *The message certifies the binding of Y and PK_Y.*
        - *X* 是擔保此綁定的認證機構。
            - *X is the certifying authority.*
        - 資料項 `Y` 和 `PK_Y` 在簽署後未被篡改或損壞。
            - *Data items Y and PK_Y were not altered or corrupted.*

此機制的前提是： ***Z* 必須擁有一個可信的 *X* 的公開金鑰**，才能驗證 *X* 的簽章。這就形成了一條信任鏈。
*This scheme assumes that **Z has a trustworthy public key for X**, to verify X's signature. This forms a chain of trust.*

經驗總結 / Lessons Learned

- **憑證的必要性：** 在分散式環境中建立信任網需要憑證。
    - *Certificates are needed to establish a web of trust in a distributed environment.*
- **核心問題：** 相互猜疑的實體如何建立信任關係？
    - *How do mutually suspicious entities establish a relationship of trust?*
- **解決方案：** 一個受信任的個人或機構可以透過認證「身分與公開金鑰的綁定」來為另一方擔保。
    - *A trusted individual can vouch for another party by certifying the binding of identity to public key.*
- **驗證：** 第三方可以檢查此綁定的有效性。
    - *A third party can check the validity of the binding.*

小測試 / Quick Test

問題： 在現代的網路瀏覽器中，當我們造訪一個 HTTPS 網站時（例如

`https://www.example.com` ），瀏覽器會自動驗證網站伺服器傳送的憑證。請根據上述概念，描述這個過程背後的信任建立流程。
Question: In modern web browsers, when we visit an HTTPS website (e.g., `https://www.example.com`), the browser automatically verifies the certificate sent by the web server. Describe the trust establishment process behind this based on the concepts above.

1. **收到憑證：** 瀏覽器從 `www.example.com` 收到一個憑證，其中綁定了網站的身分和其公開金鑰。
   - *The browser receives a certificate from `www.example.com` which binds the site's identity to its public key.*
2. **尋找簽署者：** 該憑證由一個認證機構簽署，例如 "Let's Encrypt" 或 "DigiCert"。瀏覽器**預先內建**了一組它信任的頂級根 CA 的公開金鑰。
   - *This certificate is signed by a Certification Authority (CA), like "Let's Encrypt" or "DigiCert". The browser has a **pre-installed list** of trusted root CA public keys.*
3. **驗證鏈：** 瀏覽器使用它信任的根 CA 的公開金鑰，來驗證中繼 CA 的憑證（如果需要），最終驗證網站伺服器憑證上的簽章。
   - *The browser uses the trusted root CA's public key to verify the signature on the intermediate CA's certificate (if any), and ultimately the signature on the website's certificate.*
4. **建立信任：** 如果簽章驗證成功，瀏覽器就相信「`www.example.com`」這個身分確實與它收到的公開金鑰綁定在一起。後續的加密通訊（例如建立 TLS 連線）都基於這個信任。
   - *If the signature is valid, the browser trusts that the identity `www.example.com` is correctly bound to the public key it received. All subsequent secure communication is based on this trust.*

答案： 信任是透過一個**預先設定的信任根**來建立的。瀏覽器製造商將主要認證機構的公開金鑰內建於軟體中。當網站出示由這些 CA 簽署的憑證時，瀏覽器可以透過驗證數位簽章來確認網站身分的真實性，從而建立起對網站的信任。
*Answer: Trust is established through a **pre-configured trust anchor**. Browser vendors embed the public keys of major CAs into the software. When a website presents a certificate signed by one of these CAs, the browser can verify the website's identity authenticity by validating the digital signature, thereby establishing trust in the website.*

Chains of Trust and Certification Authorities

中英雙語解釋 / Bilingual Explanation

信任網 / Web of Trust

- **概念：** 網路上許多活動，特別是電子商務，都依賴於在各方之間建立一個**信任網**。
  - **Concept:** Much of what happens on-line, particularly e-commerce, depends on establishing a **web of trust** relationships among the parties.
- **核心問題：** 為什麼 A 應該信任一個他從未打過交道的 B？
  - **Core Question:** Why should A trust B with whom he's never previously dealt?
- **可能的答案：** A 可以依賴一個已知的第三方來「擔保」B。
  - **Possible Answer:** A might rely on a known third party to "vouch for" B.
- **現實世界的例子：** 商會、商業改善局、信用報告機構、朋友等，在某種程度上都充當了某些商業交易的認證機構。
  - **Real-world Examples:** The Chamber of Commerce, Better Business Bureau, credit reporting agencies, friends, etc. all function in part as certification authorities for some commercial transactions.

對信任的需求 / Need for Trust

- **公開金鑰基礎建設的情境：** 在公開金鑰基礎建設中，如果 A 知道 B 的公開金鑰，那麼 A 可以：
  - **With a PKI:** If A knows B's public key, then A can:
  - 安全地發送訊息給 B。
    - *Send a message securely to B;*
  - 確信來自 B 的訊息確實源自 B。
    - *Be assured that a message from B really originated with B.*
- **關鍵問題：** 但是，A 如何知道 B 提供的公開金鑰**確實是 B 的**，而不是別人的？
  - **Key Problem:** But, how does A know that the public key B presents is **really B's public key** and not someone else's?
- **信任的綁定：** 在分散式線上環境中最需要信任的情況，就是**可靠地將公開金鑰與身分綁定在一起**。
  - **The Binding of Trust:** The most common circumstance in which trust is needed is **reliably binding a public key to an identity**.

憑證 / Certificates

- **定義：** 憑證是「介紹信」的電子等效物。
  - **Definition:** A certificate is the electronic equivalent of a "letter of introduction."

- **建構方式：** 憑證是使用**數位簽章**和**雜湊函式**構建的。
  - **Construction:** A certificate is constructed with **digital signatures** and **hash functions**.
- **核心功能：** 一個公開金鑰和使用者的身分被**綁定**在一個憑證中，並由**認證機構**簽署，以擔保此綁定的準確性。
  - **Core Function:** A public key and a user's identity are **bound together** within a certificate, **signed by a certification authority**, vouching for the accuracy of the binding.

運作方式範例 / How It Might Work

假設：
Suppose:

- *X* 是公司總裁；*Y* 是她的下屬。每人擁有一對 RSA 公開金鑰。
  - *X is the president of a company; Y is her subordinate. Each has an RSA public key pair.*

建立憑證的步驟：
Steps to create a certificate:

1. *Y* 安全地將訊息 `{Y, PK_Y}` 傳遞給 *X*。
   - *Y securely passes message* `{Y, PK_Y}` *to X.*
2. *X* 產生該訊息的密碼學雜湊值：`H({Y, PK_Y})`。
   - *X produces a cryptographic hash of the message:* `H({Y, PK_Y})`.
3. *X* 使用她的私密金鑰對該雜湊值進行簽署，產生 `Sign_SK_X ( H({Y, PK_Y}) )`。
   - *X signs this hash with her private key, producing* `Sign_SK_X ( H({Y, PK_Y}) )`.
4. *X* 將所有資訊組合起來，產生 `{Y, PK_Y, Sign_SK_X ( H({Y, PK_Y}) )}`。
   - *X produces* `{Y, PK_Y, Sign_SK_X ( H({Y, PK_Y}) )`.
5. 這最後的結果就成為由 *X* 簽署的 *Y* 的憑證。
   - *This last then becomes Y's certificate, signed by X.*

驗證憑證 / Validating the Certificate

假設 *Y* 向 *Z* 出示該憑證：
Suppose Y presents to Z the certificate:
`{Y, PK_Y, Sign_SK_X ( H({Y, PK_Y}) )}`

Z 會如何處理它？*Z* 能從中得知什麼？
What does Z do with this? What does Z learn?

1. **取得憑證與公開金鑰：** *Z* 首先需要取得他信任的 *X* 的公開金鑰。
   - *Z first needs to obtain X's public key (PK_X), which he trusts.*
2. **驗證簽章：** *Z* 使用 *X* 的公開金鑰來解密憑證中的數位簽章，從而得到 *X* 當初計算出的雜湊值 `H1` 。
   - *Z uses PK_X to decrypt the digital signature in the certificate, obtaining the hash value `H1` that X originally computed.*
3. **計算雜湊值：** *Z* 自己對憑證中的 `{Y, PK_Y}` 部分進行雜湊計算，得到 `H2` 。
   - *Z independently computes the hash of the `{Y, PK_Y}` part from the certificate, obtaining `H2` .*
4. **比對與信任推導：** *Z* 比對 `H1` 和 `H2` 。
   - *Z compares `H1` and `H2` .*
   - 如果 `H1 == H2` ，則 *Z* 可以確信：
     - *If `H1 == H2` , then Z can be assured that:*
     - 此憑證擔保了 *Y* 和 `PK_Y` 的綁定關係。
       - *The message certifies the binding of Y and PK_Y.*
     - *X* 是擔保此綁定的認證機構。
       - *X is the certifying authority.*
     - 資料項 `Y` 和 `PK_Y` 在簽署後未被篡改或損壞。
       - *Data items Y and PK_Y were not altered or corrupted.*

此機制的前提是：*Z* 必須擁有一個可信的 *X* 的公開金鑰，才能驗證 *X* 的簽章。這就形成了一條信任鏈。
*This scheme assumes that **Z has a trustworthy public key for X**, to verify X's signature. This forms a chain of trust.*

經驗總結 / Lessons Learned

- **憑證的必要性：** 在分散式環境中建立信任網需要憑證。
  - *Certificates are needed to establish a web of trust in a distributed environment.*
- **核心問題：** 相互猜疑的實體如何建立信任關係？
  - *How do mutually suspicious entities establish a relationship of trust?*
- **解決方案：** 一個受信任的個人或機構可以透過認證「身分與公開金鑰的綁定」來為另一方擔保。

    - *A trusted individual can vouch for another party by certifying the binding of identity to public key.*
- **驗證：** 第三方可以檢查此綁定的有效性。
    - *A third party can check the validity of the binding.*

小測試 / Quick Test

問題： 在現代的網路瀏覽器中，當我們造訪一個 HTTPS 網站時（例如 `https://www.example.com`），瀏覽器會自動驗證網站伺服器傳送的憑證。請根據上述概念，描述這個過程背後的信任建立流程。
Question: In modern web browsers, when we visit an HTTPS website (e.g., `https://www.example.com`), the browser automatically verifies the certificate sent by the web server. Describe the trust establishment process behind this based on the concepts above.

1. **收到憑證：** 瀏覽器從 `www.example.com` 收到一個憑證，其中綁定了網站的身分和其公開金鑰。
    - *The browser receives a certificate from `www.example.com` which binds the site's identity to its public key.*
2. **尋找簽署者：** 該憑證由一個認證機構簽署，例如 "Let's Encrypt" 或 "DigiCert"。瀏覽器**預先內建**了一組它信任的頂級根 CA 的公開金鑰。
    - *This certificate is signed by a Certification Authority (CA), like "Let's Encrypt" or "DigiCert". The browser has a **pre-installed list** of trusted root CA public keys.*
3. **驗證鏈：** 瀏覽器使用它信任的根 CA 的公開金鑰，來驗證中繼 CA 的憑證（如果需要），最終驗證網站伺服器憑證上的簽章。
    - *The browser uses the trusted root CA's public key to verify the signature on the intermediate CA's certificate (if any), and ultimately the signature on the website's certificate.*
4. **建立信任：** 如果簽章驗證成功，瀏覽器就相信「`www.example.com`」這個身分確實與它收到的公開金鑰綁定在一起。後續的加密通訊（例如建立 TLS 連線）都基於這個信任。
    - *If the signature is valid, the browser trusts that the identity `www.example.com` is correctly bound to the public key it received. All subsequent secure communication is based on this trust.*

答案： 信任是透過一個**預先設定的信任根**來建立的。瀏覽器製造商將主要認證機構的公開金鑰內建於軟體中。當網站出示由這些 CA 簽署的憑證時，瀏覽器可以透過驗證數位簽章來確認網站身分的真實性，從而建立起對網站的信任。

*Answer: Trust is established through a **pre-configured trust anchor**. Browser vendors embed the public keys of major CAs into the software. When a website presents a certificate signed by one of these CAs, the browser can verify the website's identity authenticity by validating the digital signature, thereby establishing trust in the website.*

Entity Authentication

中英雙語解釋 / Bilingual Explanation

識別與實體驗證 / Identification and Entity Authentication

這兩個概念密切相關但有所不同：
These two concepts are closely related but distinct:

- **識別：** 指的是一個主體向另一個實體**表明其身分**。
  - **Identification:** Refers to a subject **revealing its identity** to another entity.
- **實體驗證：** 指的是**驗證**所聲稱的身分是否正確的過程。它確認「你就是你所聲稱的那個人」。
  - **Entity Authentication:** Refers to a **verification** that the alleged identity is correct. It confirms "you are who you claim to be."

為什麼需要識別？ / Why is Identification Needed?

識別是許多安全機制的基礎，主要用於：
Identification is the foundation for many security mechanisms, primarily for:

- **存取控制：** 決定哪個主體可以存取哪個物件（例如，使用者A能否讀取檔案B）。
  - **Access Control:** e.g., which subject can access which object?
- **審計：** 分析日誌記錄，以清晰、易理解的方式呈現系統的活動資訊。沒有準確的識別，日誌就無法追溯到具體負責的使用者。
  - **Auditing:** Analysis of log records to present information about the system in a clear, understandable manner.

人類的識別方法 / How Humans Can Be Identified

人類可以透過多種方式進行識別，常見的分類如下：
Humans can be identified by several means, commonly categorized as follows:

1. 所知之物 / Something They Know

- **解釋：** 依靠使用者記憶中的秘密。
  - **Explanation:** Relies on secrets memorized by the user.
- **例子：** 密碼、複雜密語、個人識別碼、個人資料問題的答案等。
  - **Examples:** Password, passphrase, Personal Identification Number (PIN), answers to questions on personal data, etc.

2. 所具之形 / Something They Are

- **解釋：** 依靠使用者獨一無二的生理特徵。
  - **Explanation:** Relies on the user's unique physiological characteristics.
- **例子：** **生物特徵識別**，如虹膜紋理、視網膜圖案、人臉或指紋識別、手部幾何形狀、掌紋或靜脈圖案、體味分析等。
  - **Examples: Biometric identification**: iris texture, retina pattern, face or fingerprint recognition, finger or hand geometry, palm or vein patterns, body odor analysis, etc.

3. 所行之為 / Something They Do

- **解釋：** 依靠使用者獨特的行為模式。
  - **Explanation:** Relies on the user's unique behavioral patterns.
- **例子：** 手寫簽名動態、鍵盤敲擊節奏、聲音特徵、唇部運動模式等。
  - **Examples:** Handwritten signature dynamics, keystroke dynamics, voice, lip motion, etc.

---

4. 所持之物 / Something They Have

- **解釋：** 依靠使用者擁有的實體裝置或物件。
  - **Explanation:** Relies on a physical device or object possessed by the user.
- **例子：** **存取權杖**，如實體鑰匙、身份證、智慧卡、手機、PDA等。
  - **Examples: Access tokens**: physical key, ID card, smartcard, mobile phone, PDA, etc.

5. 所在之處 / Where They Are

- **解釋：** 依靠使用者的地理位置或網路位置。
  - **Explanation:** Relies on the user's geographic or network location.
- **例子：** 終端線路、電話來電顯示、IP位址、手機或無線區域網路定位資料、GPS位置等。
  - **Examples:** Location information: terminal line, telephone caller ID, Internet (IP) address, mobile phone or wireless LAN location data, GPS.

多重要素驗證 / Multi-Factor Authentication (MFA)

- **核心概念：** 為了實現高安全性，需要**結合多種**識別技術。
  - **Core Concept:** For high security, **several identification techniques need to be combined**.
- **目的：** 這樣可以降低單一方法所帶來的風險，例如：
  - **Purpose:** This reduces the risks associated with any single method, such as:
  - 誤接受率/誤拒絕率（生物特徵的準確性問題）
    - *False-accept/false-reject rates (biometric accuracy issues)*
  - 權杖被盜
    - *Token theft*
  - 人為疏失（如密碼洩露）
    - *Carelessness (e.g., password leakage)*
  - 中繼攻擊或冒用身分
    - *Relaying and impersonation*
- **常見實踐：** 使用「所知之物」（密碼）加上「所持之物」（手機驗證App）進行登入，就是典型的**雙重因素驗證**。
  - **Common Practice:** Using a password (something you know) with a mobile authenticator app (something you have) is a classic example of **Two-Factor Authentication (2FA)**.

經驗總結 / Lessons Learned

- **識別**指的是主體表明其身分，而**實體驗證**指的是驗證該身分的正確性。
  - *Identification refers to a subject revealing its identity, while entity authentication refers to a verification that the alleged identity is correct.*
- 識別與驗證是**存取控制**和**審計**的基礎。
  - *Needed for access control and auditing.*
- 在實務中，可以結合多種識別技術來**提升安全強度**。
  - *Several identification techniques can be combined to raise the security strength in practice.*

小測試 / Quick Test

問題： 一位使用者嘗試從一台新的筆記型電腦登入公司的雲端電子郵件服務。系統要求她輸入密碼後，還需要從她手機上的驗證器應用程式輸入一個6位數的動態碼。請根據上述分類，分析這個登入過程使用了哪幾種識別方法？這屬於哪種安全實踐？
Question: A user tries to log into her company's cloud email service from a new laptop. After entering her password, the system requires her to enter a 6-digit dynamic code from an authenticator app on her smartphone. Based on the categories above, which identification methods are used in this login process? What security practice does this represent?

1. **分析步驟1：輸入密碼**
    - 這屬於「**所知之物**」，因為密碼是使用者記憶中的秘密。
    - *This is "**Something You Know**" because the password is a memorized secret.*
2. **分析步驟2：從手機App輸入動態碼**
    - 手機是使用者擁有的實體裝置，屬於「**所持之物**」。
    - *The smartphone is a physical device possessed by the user, falling under "**Something You Have**".*
    - 動態碼與該裝置綁定，確保登入者同時持有該手機。

答案： 這個登入過程結合了「所知之物」（密碼）和「所持之物」（綁定動態碼的手機）兩種不同的識別方法。這種實踐稱為**雙重因素驗證**，它透過要求提供來自兩個不同類別的憑證，顯著提高了帳戶的安全性。即使攻擊者竊取了密碼，由於他們沒有使用者的實體手機，也無法完成登入。
*Answer: This login process combines two different identification methods: "Something You Know" (password) and "Something You Have" (smartphone bound to the dynamic code). This practice is called **Two-Factor Authentication (2FA)**. It significantly enhances account security by requiring credentials from two distinct categories. Even if an attacker steals the password, they cannot complete the login without physical possession of the user's phone.*

Password and PINs

中英雙語解釋 / Bilingual Explanation

密碼與PIN碼基礎 / Basics of Passwords and PINs

- **定義：** 一種字元序列。
    - **Definition:** A sequence of characters.
- **例子：** 10位數字、一串字母等。
    - **Examples:** 10 digits, a string of letters, etc.
- **理想生成方式：** 應由使用者或電腦隨機生成（可包含使用者輸入）。

    - **Ideal Generation:** Supposed to be generated randomly, by user or computer with user input.
- **另一種形式：** 單字序列，例如**複雜密語**。
    - **Another Form:** Sequence of words, e.g., pass-phrases.

人為選擇密碼的挑戰 / The Challenge of Human-Generated Passwords

- **普遍問題：** 人們傾向於選擇容易猜測的密碼。
    - **General Problem:** People tend to pick easy to guess passwords.
- **安全性不足：** 隨機挑選的單字熵值很低，且字典中的單字數量通常少於 2^18 個，易受字典攻擊。
    - **Low Security:** Randomly picked single words have low entropy, and dictionaries have less than 2^18 entries.

研究發現 / Research Findings:

一項關於網頁瀏覽的研究顯示：
A study of web browsing shows:

- 平均每位使用者擁有 **6-7 個密碼**，每個密碼大約在 **4 個網站**上重複使用。
    - *Average user has **6-7 passwords**, sharing each among about **4 sites**.*
- 使用者傾向於**直到得知密碼已外洩**才會更改密碼。
    - *Users tend **NOT to change password until they know it has been compromised**.*
- 而當他們更改時，新密碼的長度傾向於**僅達到規定允許的最短長度**。
    - *And when they do, the new passwords tend to be **as short as allowed**.*
- **可記憶性**是環境如何影響安全性的一個很好例子——因為需要記憶多組密碼，導致行為改變，從而降低了安全性。
    - *In the context of remembering passwords/passphrases, **memorability** is a good example of how environment affects security.*

密碼與PIN碼的常見改善措施 / Common Improvements to Passwords and PINs

為了應對上述挑戰，系統管理員和服務提供商通常會實施以下改善措施：
To address these challenges, system administrators and service providers often implement the following improvements:

1. **限制嘗試速率與監控失敗登入：**
    - 限制密碼的嘗試速率，以強制執行「拒絕延遲」。
    - *Restrict rate at which passwords can be tried to enforce the "reject delay".*
    - 監控失敗的登入嘗試。
    - *Monitor failed logins.*
2. **制定複雜度政策：**
    - 要求最低長度，並必須包含數字、標點符號和大小寫字母。
    - *Require minimum length and inclusion of digits, punctuation, and mixed case letters.*
3. **提供密碼創建建議：**
    - 建議難以猜測的選擇方法，例如：
    - *Suggest recipes for difficult to guess choices, e.g.:*
    - 使用整個句子、與個人歷史相關句子的首字母等。
    - *Entire phrase, initials of a phrase related to personal history, etc.*
    - **例子：** "Dartmouth College Hanover NH, Dartmouth Medical Center/Mary Hitchcock memorial hospital" -> **DCHNH,DMC/MHmh**
        - *Example: "Dartmouth College Hanover NH..." -> **DCHNH,DMC/MHmh***
4. **比對弱密碼清單：**
    - 將設定的密碼與目錄和已公開的熱門/已外洩密碼清單進行比較。
    - *Compare passwords with directories and published lists of popular/compromised passwords.*
    - 這些清單通常包含：個人姓名、寵物名、品牌名、名人名、各種排列組合的姓名首字母和生日等。
    - *These lists include: the person's names, pet names, brand names, celebrity names, patterns of initials and birthdays in various arrangements, etc.*
5. **推廣密碼管理員與獨立密碼：**
    - 鼓勵使用密碼管理員、機器生成的權杖，並在每個信任域使用獨立的密碼。
    - *Encourage use of a password manager, machine-generated tokens, and use of independent passwords for each trust domain.*
    - **注意：** 使用密碼管理員後，使用者現在必須記住用於解鎖其他所有密碼的**主密碼**。
    - *Note: for password manager, now must remember the **master password** to unlock*

the other passwords.

6. **協助發放隨機密碼：**
    - 協助發放隨機生成的PIN碼或密碼，最好是**可發音的**，以便記憶。
    - *Help issue randomly generated PINs or passwords, preferably **pronounceable** ones.*
7. **強制定期更換密碼與防止重複使用：**
    - 強制使用者在特定時間後更改密碼。
    - *Force users to change passwords after some time has expired.*
    - **如何防止重複使用？**
        - *How do you force users not to re-use passwords?*
    - 記錄先前的密碼（與歷史密碼比對）。
        - *Record previous passwords.*
    - 在一定時間內禁止再次更改（防止使用者快速循環改回舊密碼）。
        - *Block changes for a period of time.*
    - 給予使用者時間思考好的密碼：不要強制他們在登入前就必須更改，並提前數天警告他們密碼即將到期。
        - *Give users time to think of good passwords: don't force them to change before they can log in, warn them of expiration days in advance.*

經驗總結 / Lessons Learned

- 密碼目前仍是身份驗證的主要基礎，並且會持續存在。
    - *Passwords are here to stay and they provide a basis for most forms of authentication.*
- 選擇和管理密碼與PIN碼並不容易，且容易出錯。
    - *Selecting and managing passwords and PINs is not easy, and can be error-prone.*
- **可記憶性**是環境如何影響安全性的一個典型例子。
    - *Memorability is a good example of how environment affects security.*
- 許多常見的改善措施已在實踐中應用，它們是必要的。
    - *Many common improvements are already in practice, and they are necessary.*

小測試 / Quick Test

問題： 一家公司計劃更新其密碼政策。他們考慮強制要求所有員工每60天更改一次密碼，且新

密碼必須包含大寫字母、小寫字母、數字和符號。然而，有員工反映這導致他們經常忘記新密碼，並傾向於將密碼寫在便利貼上貼在螢幕旁。根據上述內容，這種政策可能忽略了哪些改善措施中的要點？有什麼更好的實踐方法？
Question: A company plans to update its password policy. They consider forcing all employees to change passwords every 60 days, requiring uppercase, lowercase, digits, and symbols. However, employees complain this leads to frequent forgotten passwords and they tend to write passwords on sticky notes attached to their monitors. Based on the content above, what key points from the improvements might this policy be ignoring? What are some better practices?

1. **分析問題：**
    - 頻繁的強制更換（尤其若未給予足夠思考時間）會加劇使用者的記憶負擔，導致**「可記憶性」**問題，從而引發不安全的行為（寫下密碼）。
    - *Frequent forced changes exacerbate the **memorability** issue, leading to insecure behaviors (writing passwords down).*
    - 單純的複雜度要求，若未與其他措施（如比對弱密碼清單）結合，使用者仍可能創建符合規則但易猜測的密碼（如 `Password1!`）。
2. **建議更好的實踐：**
    - **推廣密碼管理員：** 這是解決多組複雜密碼記憶問題的根本方法之一。公司可以推薦或提供企業版的密碼管理員。
        - *Promote Password Managers: This is a key solution to managing multiple complex passwords.*
    - **放寬更換頻率或採用風險導向驗證：** 根據NIST等現代指南，不建議頻繁強制更換密碼，除非有洩漏疑慮。可結合多因素驗證來提升帳戶整體安全性。
        - *Relax Forced Rotation or Use Risk-Based Authentication: Modern guidelines (like NIST) don't recommend frequent forced changes unless a breach is suspected. Combine with MFA for overall account security.*
    - **強調密碼長度與可記憶性：** 鼓勵使用**複雜密語**（一長串隨機但可記憶的單字），其長度提供的安全性往往高於短而複雜的密碼，且更容易記憶。
        - *Emphasize Length & Memorability: Encourage the use of **passphrases** (a long sequence of random but memorable words), which can be more secure and easier to remember than short, complex passwords.*
    - **實施帳戶鎖定與監控：** 與其僅依賴密碼複雜度和更換，不如實施登入失敗鎖定和異常登入活動監控。
        - *Implement Account Lockout & Monitoring: Instead of relying solely on password complexity and rotation, implement login failure lockouts and monitor for*

anomalous login activity.

答案： 該政策可能忽略了「可記憶性」對使用者行為的影響，以及推廣密碼管理員和複雜密語等更現代的實踐。更好的方法是**減輕使用者的記憶負擔**（透過密碼管理員）和**提升使用者體驗**（透過鼓勵使用複雜密語），並結合**多因素驗證和異常監控**來構建更強大且使用者友好的安全體系。
*Answer: The policy likely ignores the impact of "memorability" on user behavior and more modern practices like promoting password managers and passphrases. A better approach is to **reduce the user's memory burden** (via password managers) and **improve user experience** (by encouraging passphrases), combined with **Multi-Factor Authentication and anomaly monitoring** to build a stronger and more user-friendly security posture.*

Authentication Protocols

中英雙語解釋 / Bilingual Explanation

身份驗證協定概述 / Authentication Protocols Overview

- **目的：** 允許一個**聲稱者** 向**驗證者** 證明其身分。
  - **Purpose:** Allow a **claimant** to prove their identity to a **verifier**.
- **證明方式：** 通常是通過展示其對某個秘密材料（如密碼、私密金鑰）的知識、持有或綁定關係。
  - **Method:** By showing, e.g., their knowledge of a secret key materials, like password, etc.
- **安全目標：** 旨在防止涉及**冒用身分** 的攻擊，例如 C 假裝是 A。
  - **Security Goal:** They aim to prevent **masquerade** (C pretending to be A).
- **需防範的攻擊類型：**
  - **重送攻擊：** 攻擊者重複使用先前竊聽到的合法驗證訊息。
    - *Replay attacks - using a previously transmitted message.*
  - **反射攻擊：** 攻擊者將驗證者發送的挑戰原封不動地發回給驗證者，或在不同連線間利用對方的回應。
    - *Reflection attacks - returning a previous message to its originator.*
  - **交錯攻擊：** 攻擊者從一個或多個正在進行或先前的驗證交換中提取資訊，並組合起來用於攻擊。
    - *Interleaving attacks - using information from one or more ongoing or previous authentication exchanges.*

挑戰-回應機制與其漏洞 / Challenge-Response Mechanism and Its Vulnerability

一個常見的身份驗證方式是**挑戰-回應** 機制。驗證者發送一個隨機的、一次性的挑戰值給聲稱者，聲稱者必須使用共享秘密（如密碼或金鑰）對該挑戰進行某種運算後，將結果（回應）傳回，驗證者再進行驗證。
*A common authentication method is the **challenge-response** mechanism. The verifier sends a random, one-time challenge to the claimant. The claimant must perform a computation on this challenge using a shared secret (like a password or key) and return the result (response). The verifier then verifies this response.*

對雙向相同協定的反射攻擊 / Reflection Attack on Bidirectional Identical Protocols

當一個系統在**兩個方向**（例如客戶端與伺服器互相驗證）使用**完全相同**的挑戰-回應協定時，可能會遭受反射攻擊。
*When a system uses the **exact same** challenge-response protocol in **both directions** (e.g., client and server authenticating each other), it can be vulnerable to a reflection attack.*

攻擊步驟 / Attack Steps:

1. **攻擊者連線目標：** 攻擊者（C）發起一個連線到目標驗證者（V）。
   - *The attacker (C) initiates a connection to a target verifier (V).*
2. **目標發出挑戰：** 目標 V 嘗試驗證 C，於是向 C 發送一個挑戰 `Challenge_V1`。
   - *The target V attempts to authenticate C by sending it a challenge `Challenge_V1`.*
3. **攻擊者開啟第二連線並反射挑戰：** 攻擊者 C 開啟**另一個**連線到 V。在這個新連線中，C 將剛剛收到的 `Challenge_V1` 作為自己的挑戰發送給 V。
   - *The attacker C opens **another** connection to V, and sends V this `Challenge_V1` as its own challenge.*
4. **目標計算並傳回回應：** 目標 V 收到 `Challenge_V1` 後，誠實地計算出正確的回應 `Response_V1`，並將其發送給攻擊者 C（在第二連線中）。
   - *The target V responds to the challenge with the correct `Response_V1`.*
5. **攻擊者完成第一連線的驗證：** 攻擊者 C 現在獲得了 V 對 `Challenge_V1` 的正確回應 `Response_V1`。他將 `Response_V1` 發送回 V，作為對**第一個連線**中挑戰的回應。
   - *The attacker sends that `Response_V1` back to V on the **original** connection.*
6. **攻擊成功：** 目標 V 在第一個連線中收到 `Response_V1`，驗證通過，從而誤以為攻擊者 C 是合法實體。
   - *V verifies the response on the original connection and mistakenly authenticates C.*

核心問題： 協定在兩個方向上對稱，使得攻擊者能將一個連線中收到的挑戰，在另一個連線中

「反射」給驗證者，並利用驗證者自己的回應來通過驗證。
The Core Problem: The protocol is symmetric in both directions, allowing the attacker to "reflect" a challenge from one connection back to the verifier in another connection, and use the verifier's own response to authenticate.

安全的驗證協定組成要素 / Components of Secure Authentication Protocols

為了抵禦上述攻擊，安全的身份驗證協定在進行挑戰-回應交換時，會包含以下難以偽造的元件：
To resist such attacks, secure authentication protocols perform challenge-response exchanges that include unforgeable components:

1. **密碼學檢查值：**
    - **訊息驗證碼：** 使用共享密鑰生成，保證訊息的完整性和真實性。
        - *Message Authentication Codes (MACs)*
    - **數位簽章：** 使用私密金鑰生成，提供不可否認性和真實性。
        - *Digital Signatures*
2. **時變參數：** 確保每次交換都是獨一無二的，防止重送攻擊。
    - *Time variant parameters (nonces)*
    - **時間戳記：** 依賴同步的時鐘來確保訊息的新鮮度。
        - *Timestamps*
    - **序列號：** 每次交換使用遞增的號碼。
        - *Sequence numbers*
    - **隨機數：** 每次隨機生成的一個數字，在其關聯密鑰的生命週期內極不可能重複。這是目前最常用的方式。
        - *Random numbers (nonces) - generated such that they are extremely unlikely to repeat during the lifetime of an associated cryptographic key.*

此外，安全的協定設計還包括：
Furthermore, secure protocol design includes:

- **明確標識參與方：** 在計算 MAC 或簽章時，明確將參與雙方的身份標識包含在內，防止交錯攻擊中角色混淆。
- **使用不同的金鑰：** 雙向驗證時使用不同的金鑰，防止反射攻擊中直接利用對方的回應。
- **協定方向非對稱：** 設計非對稱的協定流程，使客戶端和伺服器的驗證步驟有所不同。

經驗總結 / Lessons Learned

- 身份驗證協定是建立信任的關鍵，但設計不當會引入嚴重漏洞。
  - *Authentication protocols are crucial for establishing trust, but flawed design can introduce critical vulnerabilities.*
- **挑戰-回應** 是核心機制，但其實現必須謹慎。
  - *Challenge-response is a core mechanism, but its implementation must be careful.*
- 在雙向驗證中，使用**完全相同且對稱**的協定是危險的，容易導致**反射攻擊**。
  - *Using the **identical and symmetric** protocol for bidirectional authentication is dangerous and prone to **reflection attacks**.*
- 安全的協定需要結合**密碼學檢查值**和**時變參數**，並在設計上考慮到各種交錯攻擊的可能性。
  - *Secure protocols need to combine **cryptographic check values** and **time-variant parameters**, and be designed to consider various interleaving attacks.*

小測試 / Quick Test

問題： 一個簡單的協定設計如下：客戶端（C）和伺服器（S）共享一個對稱密鑰 K。S 向 C 發送一個隨機數 Nc。C 必須回傳 `E(K, Nc)`（即用 K 加密 Nc）。然後 S 也向 C 發送另一個隨機數 Ns，C 需要驗證 S 回傳的是 `E(K, Ns)`。這個協定是否能有效防範反射攻擊？為什麼？
Question: A simple protocol is designed as follows: Client (C) and Server (S) share a symmetric key K. S sends a nonce Nc to C. C must return `E(K, Nc)`. Then S sends another nonce Ns to C, and C needs to verify that S returns `E(K, Ns)`. Is this protocol effective against reflection attacks? Why?

1. **分析協定對稱性：** 這個協定在兩個方向上使用了**完全相同的操作**：一方發送一個隨機數，另一方回傳該隨機數的加密結果。
   - *The protocol uses the **exact same operation** in both directions: one party sends a nonce, the other returns the encryption of that nonce.*
2. **構思反射攻擊：**
   - 攻擊者 A 可以冒充 C 連接到 S。
   - S 向 A 發送挑戰 Nc。
   - 攻擊者 A 同時（或隨後）冒充 S 連接到真正的 S（或利用另一個服務執行相同協定）。
   - 在這個新連線中，A 將 Nc 作為挑戰發送給 S。
   - S 誠實地計算出 `E(K, Nc)` 並回傳給 A。
   - 攻擊者 A 將 `E(K, Nc)` 作為對第一個連線的回應發送給 S。

    - S 驗證 `E(K, Nc)` 正確，從而錯誤地驗證了攻擊者 A。
3. **結論：**
    - 這個協定**無法**有效防範反射攻擊。

答案： 不能。因為該協定在客戶端和伺服器端使用了對稱的驗證邏輯。攻擊者可以將從伺服器收到的挑戰，反射給另一個伺服器實例（或同一伺服器的另一個連線），並將伺服器計算出的正確加密結果用作自己的回應，從而通過驗證。改進方法是讓雙方的驗證流程**不對稱**，例如在加密物件中綁定雙方身份標識和對話的具體方向。
*Answer: No, it is not effective. Because the protocol uses symmetric authentication logic for both client and server. An attacker can reflect a challenge received from the server to another instance of the server (or another connection to the same server) and use the correct encrypted result computed by the server as their own response, thereby passing authentication. An improvement is to make the verification flows **asymmetric**, for example, by binding the identities of both parties and the direction of the conversation within the encrypted object.*

NeedhamSchroeder protocol Kerberos

好的，我們來用中英雙語解釋「Needham-Schroeder 協定」與「Kerberos」這兩個重要的身份驗證系統。

中英雙語解釋 / Bilingual Explanation

Needham-Schroeder 協定概述 / Needham–Schroeder Protocol Overview

- **歷史地位：** 許多現有的身份驗證協定都衍生自 Needham 和 Schroeder 於 1978 年提出的協定，其中包括廣泛使用的 **Kerberos** 身份驗證協定套件。
    - **Historical Significance:** Many existing protocols are derived from one proposed by Needham and Schroeder (1978), including the widely used **Kerberos** authentication protocol suite.
- **核心設計：** 它是一個**共享金鑰** 的身份驗證協定，旨在為後續的對稱加密通訊**生成並傳播一個工作階段金鑰**。
    - **Core Design:** It is a **shared-key authentication protocol** designed to generate and propagate a **session key** for subsequent symmetrically encrypted communication.
- **前提假設：** 請注意，此協定設計於公開金鑰基礎建設普及之前，因此不依賴 PKI。
    - **Assumption:** Note that there is no public key infrastructure in place.

協定核心概念：信任的第三方 / Core Concept: Trusted Third Party

Needham-Schroeder 是一個基於**對稱密碼學**並依賴**信任第三方**的身份驗證協定：
It is a trusted third party based authentication with symmetric cryptography:

- **初始狀態：** 使用者 A 和伺服器 B **最初並不共享**一個秘密金鑰。
  - *User A and server B do not share a secret key initially.*
- **信任基石：** 但是，身份驗證伺服器 S 與網路中的**每一個實體**都共享一個獨特的秘密金鑰。
  - *But authentication server S shares secret keys with everyone.*
- **協定目標：** A 向 S 請求與 B 建立一個安全工作階段。
  - *A requests a session with B from S.*
- **S 的角色：** S 生成一個工作階段金鑰 K_AB，並分別為 A 和 B 將其加密。
  - *S generates session key K_AB and encrypts it separately for both A and B.*
- **防止重送攻擊：** 這些「票證」中包含一個**時間戳記 T_S** 和**有效期限 L**，以限制它們的使用時間，確保訊息的新鮮度。
  - *These tickets contain a timestamp T_S and lifetime L to limit their usage time.*
- **結果：** A 可以確信 B 的最終訊息是新鮮的，而不是來自先前交換的重送。
  - *A knows B's last message is fresh, not a replay from an earlier exchange.*

Kerberos：現代的實踐 / Kerberos: The Modern Implementation

- **與 Needham-Schroeder 的關係：** Kerberos 是 Needham-Schroeder 協定的一個擴展，現在**廣泛用於企業網路**中，在桌面電腦與伺服器之間進行身份驗證。
  - **Relationship to N-S:** An extension of the Needham–Schroeder protocol is now widely used in corporate computer networks between desktop computers and servers.
- **實作範例：** 它以 MIT 的 Kerberos 協定形式存在，同時也實作在 **Microsoft 的 Active Directory 服務**中。
  - **Implementations:** in the form of MIT's Kerberos protocol, which is also implemented in **Microsoft's Active Directory service**.

Kerberos 核心概念 / Kerberos Core Concepts

- **主體名稱：** Kerberos 透過**主體名稱**來識別使用者、電腦和服務。其格式為 `user@REALM` 或 `service/computer@REALM`。
  - **Principal Name:** Kerberos identifies users, computers and services via a principal name of the form `user@REALM` or `service/computer@REALM`.
  - **例子：** 一個使用者可能是 `alice@COMPANY.COM`，一個網頁伺服器可能是

`HTTP/www.server.com@COMPANY.COM` 。
    - *Example: A user might be `alice@COMPANY.COM`, a web service might be `HTTP/www.server.com@COMPANY.COM`.*
- **領域：** 一個 **Kerberos 領域** 標識了（一組）可以對該主體進行身份驗證的**金鑰分發中心**。
    - **Realm:** A **Kerberos realm** identifies (a set of) **Key Distribution Centers (KDCs)** that can authenticate that principal.
    - **微軟術語：** 在 Microsoft 的文件中，使用 **「網域」** 一詞來代替「領域」。
        - *Microsoft documentation uses the term **domain** instead.*

簡單比喻：
Simple Analogy:

- **KDC / 驗證伺服器 (S)：** 就像一個大型活動的**總票務中心**。你（A）必須先到這裡證明你的身分（例如用員工證），才能取得進入不同區域（服務 B、C、D）的門票。
- **票證：** 總票務中心給你的是一張加密的、有時效性的門票，上面寫著「允許 A 在接下來 8 小時內與 B 通訊」。
- **領域：** 整個活動會場就是一個「領域」。如果你的公司有美國和亞洲分公司，可能就會有 `US.COMPANY.COM` 和 `ASIA.COMPANY.COM` 兩個不同的領域/KDC。

經驗總結 / Lessons Learned

- Needham-Schroeder 是一個共享金鑰身份驗證協定，具有非常重要的**歷史意義**。
    - *Needham-Schroeder is a shared-key authentication protocol that has been very important historically.*
- 它清晰地闡明了：
    - *It illustrates:*
    - 協定的**整體結構**。
        - *the overall structure of protocols;*
    - 某些參與者（如驗證伺服器）可能扮演**特殊角色**。
        - *that some principals may have special roles to play;*
    - **隨機數** 在確保訊息新鮮度方面的實用性。
        - *the usefulness of nonces.*
- Needham-Schroeder 協定的一個擴展，即 **MIT 的 Kerberos 協定**，現已廣泛用於企業電腦網路中。
    - *An extension of the Needham–Schroeder protocol, in the form of MIT's Kerberos protocol, is now widely used in corporate computer networks.*

小測試 / Quick Test

問題： 在一個使用 Kerberos 進行身份驗證的公司環境中，當一位員工（`alice@COMPANY.COM`）嘗試存取內部網頁伺服器（`HTTP/intranet.company.com@COMPANY.COM`）上的一個受保護頁面時，請描述 Kerberos 如何基於 Needham-Schroeder 的「信任第三方」模型來促成這次安全存取。
Question: In a corporate environment using Kerberos for authentication, when an employee (`alice@COMPANY.COM`) tries to access a protected page on an internal web server (`HTTP/intranet.company.com@COMPANY.COM`), describe how Kerberos, based on the "trusted third party" model of Needham-Schroeder, facilitates this secure access.

1. **向 KDC 驗證：** Alice 首先向公司的 KDC（信任第三方）證明自己的身分（通常透過密碼）。KDC 驗證後，發給她一個**票證授予票證**，這是一個特殊的票證，允許她後續向 KDC 請求其他服務的票證。
    - *Alice first authenticates to the company KDC (trusted third party), typically with her password. Upon verification, the KDC gives her a **Ticket-Granting Ticket (TGT)**, which is a special ticket to request service tickets later.*
2. **請求服務票證：** Alice 想要存取網頁伺服器。她向 KDC 出示她的 TGT，並請求一個用於 `HTTP/intranet.company.com` 服務的票證。
    - *Alice wants to access the web server. She presents her TGT to the KDC and requests a ticket for the `HTTP/intranet.company.com` service.*
3. **取得服務票證：** KDC 生成一個新的**工作階段金鑰**，供 Alice 和網頁伺服器使用。它建立一個服務票證，其中包含這個工作階段金鑰和 Alice 的身分資訊，並用**網頁伺服器的秘密金鑰**加密此票證（這樣只有伺服器能讀取）。然後，KDC 將這個服務票證和用 **Alice 的密鑰**加密的工作階段金鑰一起發送給 Alice。
    - *The KDC generates a new **session key** for Alice and the web server. It creates a service ticket containing this session key and Alice's identity, encrypted with the **web server's secret key**. It then sends this service ticket and the session key (encrypted for Alice) back to her.*
4. **向伺服器驗證：** Alice 將收到的服務票證發送給網頁伺服器。由於票證是用伺服器的密鑰加密的，伺服器可以解密它，從而獲取工作階段金鑰並確認 Alice 的身分（因為 KDC 是雙方都信任的）。
    - *Alice sends the service ticket to the web server. The server decrypts it (using its own key), obtains the session key, and verifies Alice's identity (because the KDC is trusted by both).*
5. **安全通訊：** 現在 Alice 和網頁伺服器共享一個由 KDC 安全分發的工作階段金鑰，她們可以使用這個金鑰來加密後續的所有通訊。

答案： Kerberos 嚴格遵循了 Needham-Schroeder 的信任第三方模型。公司的 KDC 充當了所有使用者和服務都信任的中央機構。它負責驗證使用者身分，並安全地分發工作階段金鑰（透過雙重加密的票證），從而使使用者和服務能夠在從未直接共享秘密的情況下，相互驗證並建立安全的通訊通道。

Answer: Kerberos strictly follows the Needham-Schroeder trusted third party model. The company's KDC acts as the central authority trusted by all users and services. It is responsible for authenticating user identities and securely distributing session keys (via doubly-encrypted tickets), enabling users and services to mutually authenticate and establish a secure communication channel without having ever shared a secret directly beforehand.