信息安全基本目标

一、 核心基础：CIA 三元组 (数据属性)

这三个要素定义了数据在理想状态下应该是什么样子的。

1. 机密性 (Confidentiality)

- **定义**：只有被授权的人或系统才能访问信息。
- **核心问题**：**“不该看的人看不到。”**
- **攻防示例**：
  - 攻击：网络窃听、数据泄露。
  - 防御：加密技术（AES, RSA）、物理隔离。

2. 完整性 (Integrity)

- **定义**：数据在存储或传输过程中保持准确、一致，未被篡改。
- **核心问题**：**“数据没有被偷偷改过。”**
- **攻防示例**：
  - 攻击：中间人篡改、SQL注入修改数据。
  - 防御：哈希校验（MD5/SHA）、数字签名。

3. 可用性 (Availability)

- **定义**：授权用户在需要时能及时、可靠地访问服务或数据。
- **核心问题**：**“想用的时候能用。”**
- **攻防示例**：
  - 攻击：DDoS 攻击、勒索软件锁死文件。
  - 防御：负载均衡、异地容灾备份、RAID 磁盘阵列。

二、 关键扩展：交互与信任机制 (流程属性)

这三个要素定义了用户如何安全地使用系统，以及如何建立法律层面的信任。

4. 认证 (Authentication)

- **定义**：验证系统实体（用户或进程）所声称的身份的过程。
- **核心问题**：**“你是谁？”** (Verifying Identity)
- **PPT 重点**：它是访问系统的第一道门槛。
- **典型技术**：
  - **Kerberos 协议**（如 PPT 所述，一种基于票据的认证协议）。
  - 密码、生物识别（指纹/人脸）、双因素认证 (2FA)。

5. 访问控制 / 授权 (Access Control / Authorisation)

- **定义**：保护系统资源，防止未经授权的访问，限制用户只能执行其权限内的操作。
- **核心问题**：**“你能干什么？”** (Permissions)
- **PPT 重点**：通过认证后，系统决定你能读、写还是删除。
- **典型技术**：
  - **ACL (访问控制列表)**：明确规定谁对哪个文件有什么权限。
  - RBAC (基于角色的访问控制)：比如“经理”角色可以审批，“员工”角色只能提交。

6. 不可否认性 (Non-repudiation)

- **定义**：防止通信的一方在事后否认其参与了该通信或行为。
- **核心问题**：**“这事是你干的，别赖账。”** (Proof of Involvement)
- **PPT 重点**：主要用于保护交易和法律责任。
- **典型技术**：
  - **数字签名**：结合私钥加密，确保除了你没人能签出这个名。
  - 审计日志（Audit Trails）：不可篡改的操作记录。

三、 融会贯通：六大要素如何协作？

为了帮你彻底记住，我们用**“网上银行转账”**这个场景，把这 6 个概念串起来：

1. **认证 (Authentication)**：
   你打开银行 App，输入密码和人脸识别。系统确认**你是你**。
2. **访问控制 (Access Control)**：
   系统检测你的权限。虽然你是用户，但你**只能操作**自己的账户，**无权访问**银行的后台数据库（那是管理员的权限）。

3. **机密性 (Confidentiality)**：
   你查看余额。余额数字通过 HTTPS **加密传输**，此时坐在你旁边蹭 Wi-Fi 的黑客截获了数据包，但也**看不懂**内容。
4. **完整性 (Integrity)**：
   你输入“转账 100 元”。数据包传到银行服务器时，系统通过哈希校验确认数据包**依然是** 100 元，而不是被篡改成了 10000 元。
5. **不可否认性 (Non-repudiation)**：
   你点击“确认转账”，App 使用你的私钥对交易指令进行了**数字签名**。第二天如果你打电话给银行说“我没转过这笔钱”，银行拿出你的签名记录，让你**无法抵赖**。
6. **可用性 (Availability)**：
   即使双 11 转账的人很多，银行使用了**负载均衡**和服务器集群，保证 App **没有崩溃**，让你顺利完成了操作。

四、 总结图表

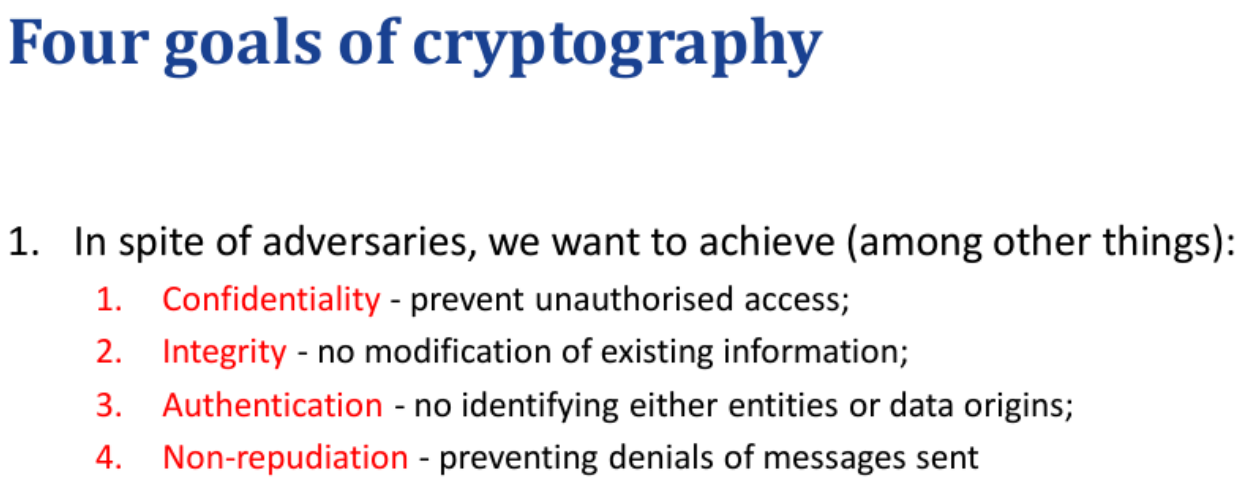
类别	概念	英文名称	核心作用	典型技术/协议
基础属性 (CIA)	**机密性**	Confidentiality	**保密** (防泄露)	加密 (AES, TLS)
	完整性	Integrity	**防篡改** (保真)	哈希 (SHA), 校验和
	可用性	Availability	**抗压** (防宕机)	备份, 冗余, DDoS清洗
流程机制 (扩展)	**认证**	Authentication	**验明正身** (你是谁)	**Kerberos**, 密码, 生物识别
	访问控制	Access Control	**权限管理** (能干嘛)	**ACL**, RBAC
	不可否认性	Non-repudiation	**铁证如山** (防赖账)	数字签名, 审计日志

这样整理之后，你就拥有了一个既包含**底层原理 (CIA)** 又包含**实现机制 (PPT内容)** 的完整网络安全知识框架了。

密码学核心目标与应用

第一步：密码学的四大核心目标

这是密码学存在的意义，也是它要解决的四个终极问题：



1. **机密性 (Confidentiality)**
   - **目标**：确保信息不被窃听。
   - **手段**：**加密（Encryption）**。把明文变成乱码。
2. **完整性 (Integrity)**
   - **目标**：确保信息未被篡改。
   - **手段**：**哈希函数（Hashing）**。提取数据的“指纹”。
3. **认证 (Authentication)**
   - **目标**：确认发信人的身份。
   - **手段**：验证对方是否持有正确的**密钥**。
4. **不可否认性 (Non-repudiation)**
   - **目标**：防止事后抵赖。
   - **手段**：**数字签名**（这是非对称加密独有的能力，对称加密做不到这一点）。

第二步：对称加密 (Symmetric Encryption)

这是密码学的“老前辈”，历史悠久，简单粗暴。

1. 核心特点

“一把钥匙开一把锁。”
通信双方（Alice 和 Bob）使用**完全相同**的一把密钥（Secret Key）。

- **加密：** Alice 用密钥 K 将明文锁住。
- **解密：** Bob 用**同一个**密钥 K 将密文打开。

2. 现实比喻

保险箱模式。
Alice 买了一个保险箱，设定密码是 `123456`，把文件放进去锁好。快递给 Bob。Bob 收到后，必须输入同样的 `123456` 才能打开取出文件。

3. 经典算法

- **DES / 3DES**（老旧，已不安全）
- **AES**（目前的国际标准，极其安全，WPA2 WiFi密码用的就是这个）

4. 致命弱点：密钥分发难题 (Key Distribution)

既然双方都要用同一把钥匙，Alice 第一次如何把钥匙安全地交给 Bob？

- 如果在网上直接发给 Bob，被黑客截获了怎么办？
- 如果派人送过去，成本又太高。
- *这就好比：为了安全传输数据，我们需要一把密钥；但为了安全传输密钥，我们需要另一把密钥……陷入死循环。*

第三步：非对称加密 (Asymmetric Encryption)

为了解决“密钥怎么给对方”的问题，天才数学家们发明了非对称加密（公开密钥加密）。

1. 核心特点

“两把钥匙，成对出现。”
每个人都拥有一对钥匙：

- **公钥 (Public Key)**：公开给全世界，谁都能拿。
- **私钥 (Private Key)**：严格保密，只有自己有，打死不能给别人。

神其规则：

- 用**公钥**加密的数据，**只有私钥**能解开。
- 用**私钥**加密的数据，**只有公钥**能解开。

2. 两种用法（解决了不同问题）

- **用法 A：为了机密性（加密通信）**
  - *场景：* 我想给马云发一条机密信息。
  - *操作：* 我拿到马云的**公钥**（满大街都是），把信锁上。
  - *结果：* 全世界只有马云有**私钥**，所以只有他能解开。
  - *比喻：* **信箱投递**。谁都能往邮筒（公钥）里塞信，但只有邮递员有钥匙（私钥）能把信取出来。
- **用法 B：为了认证与不可否认性（数字签名）**
  - *场景：* 马云发了一条公告，要证明确实是他发的。
  - *操作：* 马云用他的**私钥**对公告进行加密（签名）。
  - *结果：* 大家用马云的**公钥**去解。如果能解开，说明这东西只能是用马云的私钥锁的。因为私钥只有他有，所以**证明是他发的（认证）**，且他**无法赖账（不可否认性）**。

3. 经典算法

- **RSA**（最经典）
- **ECC**（椭圆曲线加密，比特币就在用）

第四步：巅峰对决 —— 对称 vs. 非对称

理解了原理后，我们需要对比它们的优劣，因为现代网络安全（如 HTTPS）是把这两者结合起来使用的。

特性	对称加密 (Symmetric)	非对称加密 (Asymmetric)
密钥数量	单一密钥 (Shared Secret)	密钥对 (Public + Private)
加密速度	**极快** (适合处理大数据文件)	**极慢** (比对称慢 100-1000 倍，计算复杂)
安全性	依赖于密钥不泄露	数学难题 (大数分解等) 保障安全
密钥管理	**极难** (密钥分发困难，N个人互通需要 N*(N-1)/2 把钥匙)	**容易** (公钥随意分发，私钥自己留好)
功能支持	仅支持机密性、完整性(需结合哈希)、认证	支持所有目标，**独占不可否认性 (数字签名)**
典型应用	硬盘加密、视频流加密、ZIP压缩包密码	身份验证、数字证书、密钥交换

总结与最佳实践

既然**对称加密快但不好传密钥**，而**非对称加密安全但速度慢**，聪明的工程师想出了**混合加密（Hybrid Encryption）**：

HTTPS 的工作流程（简化版）：

1. **握手阶段（非对称）：** 浏览器和服务器先用 **RSA（非对称加密）** 建立连接。虽然慢，但只传输很少的数据——即**“会话密钥”**。
2. **传输阶段（对称）：** 一旦双方安全地交换了“会话密钥”，后续看视频、刷网页的大量数据传输，全部切换回 **AES（对称加密）**。

身份认证机制

根据您提供的课件（特别是**第8页**），身份认证的三大核心要素（Authentication Factors）的定义和含义如下：

1. 所知 (Something you know)

- **定义**：**知识的展示 (Demonstration of knowledge)**。即验证用户脑海中是否记得或掌握特

定的信息。
- **含义**：这是最传统的认证方式，系统假设只有真正的用户才知道这个秘密信息。
- **课件中的例子**：密码 (password)、母亲的婚前姓氏 (mother's maiden name) 等安全问题答案。

2. 所有 (Something you have)

- **定义**：**物品的持有 (Possession of an item)**。即验证用户是否拥有某个特定的物理对象。
- **含义**：系统假设只有真正的用户才持有这个特定的物品。
- **课件中的例子**：物理钥匙 (physical key)、大学身份证 (UoA ID card)、智能卡 (smart card)。

3. 所是 (Something you are)

- **定义**：**内在特征 (Intrinsic/inherent characteristic of you)**。即验证用户本身固有的、与生俱来的生物特征。
- **含义**：这是基于生物识别技术的认证，依靠人体独一无二的生理特征来确认身份。
- **课件中的例子**：指纹 (fingerprint)、虹膜扫描 (iris scan) 或其他生物识别特征 (biometric)。

补充说明 (来自第10页)：
课件还提到，为了更强的安全性，现代系统通常使用 **双因素认证 (Two-factor authentication, 2FA)**，即要求用户必须提供上述三种要素中的**两种不同要素**（例如：密码+指纹，或者 密码+智能卡）。注意：提供两个同类要素（如密码+生日）不算双因素认证。

4. 密码安全存储

根据提供的课件（特别是第 **7, 8, 30, 46, 53** 页），以下是对密码安全存储的详细解答和对比分析。

---

1. 理解为何使用密码学哈希函数是存储密码的最佳实践

核心依据：单向性（One-way Property）与 原像抗性（Pre-image Resistance）

根据课件第 **7页** 和 **8页** 的定义，密码学哈希函数具有**单向性（One-way）**，具体表现为**原像抗性（Pre-image Resistance）**：

- **定义**：给定一个哈希值 y，要在计算上找到一个输入 x 使得 $h(x) = y$ 是极其困难的。
- **通俗理解**：你可以很容易地把“密码”变成“乱码（哈希值）”，但几乎不可能把“乱码”还原成“密码”。

为什么这是最佳实践？（结合课件第 53 页）

1. **无需存储秘密**：课件第 **53页** 指出，通过哈希，系统**“避免存储完整的（秘密）认证数据”**。系统只需要存储“补充数据”（即哈希值）。
2. **即使泄露也安全**：如果黑客攻破了数据库，拿到了密码文件，他们看到的只是一堆杂乱的哈希值。由于哈希函数的单向性（第7页），黑客无法像解密文件那样直接逆向还原出用户的真实密码。
3. **验证流程安全**：当用户登录时，系统只需将用户输入的密码进行一次哈希运算，然后对比计算出的哈希值与数据库存储的哈希值是否一致（课件第54页 Example 1 逻辑）。系统本身都不需要知道原始密码是什么。

---

2. 分析并对比：加密整个文件 vs. 单独加密 vs. 哈希函数

虽然课件主要集中在哈希函数上，但我们可以根据课件中关于密钥（Key）的描述（第 43, 47 页）推导出加密方案的劣势，并与哈希进行对比。

方案 A：加密整个密码文件 (Encrypting the whole file)

- **机制**：使用对称加密算法（如 AES）将包含所有用户密码的文件加密。
- **劣势**：
  - **密钥单点故障**：系统为了验证用户登录，必须在内存中持有解密密钥来读取文件。如果黑客攻入服务器（通常这也是获取密码文件的前提），他们很可能也能在内存或配置文件中找到这个**密钥**。
  - **全盘瓦解**：一旦密钥被盗，黑客可以瞬间解密**所有**用户的密码。
  - **性能瓶颈**：每次验证可能需要解密整个文件或大块数据，效率极低。

方案 B：单独加密每个密码 (Encrypting each password)

- **机制**：对数据库中的每个密码字段单独进行加密存储。

- **劣势**:
  - **可逆性风险**：加密（Encryption）本质上是**双向**的（可逆的）。只要有密钥，就能还原出明文。这违反了课件第 **7页** 强调的“单向性”安全原则。
  - **内部威胁**：拥有密钥的管理员或恶意员工可以直接还原出用户的明文密码，侵犯用户隐私（用户可能在其他网站使用相同密码）。
  - **密钥管理困难**：同样面临密钥存储的问题。如果密钥与数据在同一台服务器上，安全性提升有限。

方案 C：使用哈希函数 (Using Hash Functions) —— 推荐方案

- **机制**：存储 $Hash(Password)$。
- **优势**（结合课件）：
  - **不可逆（第7页）**：没有密钥可以窃取，因为哈希本身就不需要密钥来还原（Unkeyed Hash, 第40页）。即便黑客拿到数据，也必须通过暴力破解或字典攻击（Dictionary Attack, 第30页）来猜测密码，成本极高。
  - **隔离性**：攻破数据库并不等于直接获得密码。

---

3. 细化讲解与进阶安全（基于课件第 30 和 46 页）

仅仅使用普通的哈希函数（如 MD5 或 SHA-1）存储密码在现代已经不够安全了，课件中指出了潜在的攻击方式和改进方案。

风险：字典攻击与速度（第 30 页）

- **问题**：普通的哈希函数设计得**太快了**（Efficiency, 第29页）。
- **攻击**：课件第 **30页** 提到“**字典攻击（Dictionary Attack）**”。黑客可以预先计算出常用密码（如 "123456", "password"）的哈希值，列成一张表。如果你的密码哈希值在表中，密码瞬间就被破解了。
- **现状**：课件指出，如果哈希计算速度太快，黑客每秒可以尝试数十亿次比对。

解决方案 1：加盐（Salting）（第 30 页提及）

- 课件第 **30页** 提到了 **"(insufficiently salted) password hashes"**（加盐不足的密码哈希）。
- **原理**：在密码后面拼接一个随机字符串（Salt）再进行哈希。即 $Hash(Password + Salt)$。

- **作用**：即使两个用户用同样的密码，因为盐不同，存储的哈希值也不同。这让黑客无法使用预先计算好的“彩虹表”或字典表批量破解。

解决方案 2：密钥派生函数 / 慢哈希（第 46 页）

- **核心观点**：课件第 **46页** 明确指出，**"Now considered poor practice to use... algorithms like SHA directly... as they are too fast"**（现在直接使用 SHA 等算法被认为是糟糕的实践，因为它们太快了）。
- **推荐做法**：使用 **Key Derivation Functions (KDF)**，例如 **PBKDF2** 或 **scrypt**。
- **原理**：
  - **迭代（Iterate）**：把哈希过程重复很多次（例如 100,000 次）。这让验证一次密码的时间从 0.0001秒 变成 0.1秒。对用户登录没感觉，但对黑客的暴力破解来说，成本增加了数万倍。
  - **内存困难（Memory Hard）**：如 **scrypt**（第46页），强制要求大量内存，使得黑客难以用专用硬件（ASIC/GPU）进行大规模并行破解。

总结

根据课件内容，**使用哈希函数（特别是配合加盐和慢速算法如 scrypt）是存储密码的唯一正确方式。**它利用数学上的**单向性**，确保了即使数据被盗，黑客也无法直接还原出用户的原始密码，从而保护了系统的机密性和完整性。

5. 系统安全与基础防护

根据您提供的两份课件（`Firewalls...pdf` 和 `Web clients.pdf`），以下是对系统风险、防火墙及特洛伊木马的详细解答与深化讲解。

第一部分：基于课件的核心解答

1. 系统风险与威胁环境

核心解答：
根据 **PDF 1 (Page 3)** 的描述，现代网络面临的系统风险主要源于将**受保护的内部网络（Protected Network, P）**连接到**不可信的外部互联网（Internet, I）**。

- **攻击目标**：针对信息资源、机器和网络基础设施。
- **风险形式**：远程入侵、嗅探（Sniffing）、扫描、恶意软件（Malware）以及协议攻击。
- **双向风险**：不仅要防止外部攻击者进入内部（控制访问），还要防止内部已被攻陷的账户向外泄露敏感数据（Exfiltration），或者内部人员不当使用外部资源。

2. 防火墙的核心功能

核心解答：
根据 **PDF 1 (Page 5)** 的定义，防火墙是一种**网络安全设备**，其核心功能是**控制网络两部分之间的流量**（通常是内部受信任网络与外部互联网之间）。
它主要通过两种过滤方式实现控制：

- **入站过滤 (Ingress filtering)**：控制哪些流量可以**进入**网络。
- **出站过滤 (Egress filtering)**：控制哪些流量可以**离开**网络。

3. 特洛伊木马 (Trojan Horse)

核心解答：
根据 **PDF 2 (Page 11)** 关于“客户端漏洞来源”的描述，特洛伊木马属于**恶意软件（Malware）** 的一种。

- **定义**：它通常隐藏在看似合法的**下载文件**中。
- **机制**：攻击者利用用户对某个站点或文件的信任，诱导用户下载并执行该文件，从而将恶意代码植入客户端系统。

第二部分：细化讲解与深入分析

为了帮助您更透彻地理解，结合课件的其他章节，我们对上述三个概念进行细化。

1. 深入理解：系统风险的各个层面

- **Web 安全与传统安全的区别 (PDF 2, Page 5-6)：**
  - **通信网络安全**：关注的是防止攻击者拦截或控制网络传输通道。
  - **操作系统 (OS) 安全**：关注的是防止恶意软件控制客户端机器。
  - **Web 安全风险**：更加复杂，因为攻击者可能无法控制网络，但可以诱骗受害者访问恶意网站，或者利用浏览器对合法网站的信任（如 XSS, CSRF）。

- **零信任架构 (Zero Trust) (PDF 1, Page 13)：**
  - 传统的风险模型假设“内网是安全的，外网是危险的”。
  - **现代风险观**：**“永不信任，始终验证”**。即便主机在内网中，也不能默认它是可信的，所有访问都必须经过认证和授权。这是应对现代系统风险的高级策略。

2. 深入理解：防火墙的策略与类型

- **防火墙的两种策略 (PDF 1, Page 8)：**
  - **默认允许 (Permissive / Default Allow)**：“法无禁止即可为”。只拦截黑名单里的流量。缺点是如果忘了禁止某个新威胁，系统就会受攻击。
  - **默认拒绝 (Restrictive / Default Deny)**：“法无授权即禁止”。只放行白名单里的流量（如 HTTP, SSH）。这是更安全但在易用性上需要权衡的策略。
- **防火墙的技术演进 (PDF 1, Page 11-12, 24-28)：**
  - **包过滤 (Packet Filters)**：像“电话号码拦截”。只看 IP 地址和端口号，不看内容。
  - **应用层代理 (Application-level proxies)**：像“电话监听”。能理解数据内容（如 HTTP 请求），安全性高但速度慢。
  - **有状态包过滤 (Stateful Packet Filters)**：
    - 不仅看单个包，还能记住**连接状态**（如 TCP 三次握手）。
    - *优势*：如果一个包不是新连接的开始（没有 SYN 标志），也不是已有连接的一部分，直接丢弃。它比普通包过滤更智能、更高效（Page 30）。

3. 深入理解：特洛伊木马与客户端漏洞

- **信任的滥用 (PDF 2, Page 11)：**
  特洛伊木马的核心在于**社会工程学**。用户访问了一个“看起来值得信任”的站点，或者是合法的站点被攻击者篡改了。
- **后果**：
  一旦木马进入系统，它就绕过了网络边界防御（防火墙），直接在内部运行。这也是为什么仅仅依靠边界防火墙是不够的，必须结合操作系统安全补丁和反病毒软件（OS Security）。
- **隐蔽性**：
  类似于 **PDF 2 (Page 12)** 提到的“隐蔽通道”，恶意软件可能会利用看似正常的出站流量（如图片加载请求）将窃取的数据传回给攻击者服务器，这使得检测变得更加困难。

总结

这三者构成了一个防御闭环：

1. **系统风险分析** 让我们知道要保护什么（CIA 三元组）。
2. **防火墙** 是第一道防线，控制网络边界的流量进出。
3. **特洛伊木马** 的存在提醒我们，边界防御不是万能的，必须警惕应用层和客户端的内部威胁，尤其是在用户下载和执行外部内容时。
4.

6、非对称加密方案对比与应用

这是一个非常好的切入点！你提到的第 6 点总结非常精准地概括了 Diffie-Hellman (DH) 和 RSA 在现代密码学中的不同定位。

为了让你更深入地理解这两个方案在实际架构（如端到端加密）中的取舍，我们结合课件内容，从**功能原理**、**安全性特性**以及**实际应用场景**三个维度来进行深度剖析。

1. 核心对比：Diffie-Hellman (DH) vs. RSA

虽然两者都属于非对称加密（Asymmetric Cryptography），但它们的设计初衷和数学“陷阱（Trapdoor）”是不同的。

(1) 功能定位的本质区别

- **Diffie-Hellman (DH): 协商而非加密**
  - [cite_start]正如课件所述，DH 是最早的非对称协议之一，但它并不是传统意义上的“加密” [cite: 126]。
  - [cite_start]**核心功能：** 它的目的是在不安全的信道上建立一个**共享的秘密（Shared Secret）** [cite: 123]。
  - [cite_start]**过程特点：** 双方各自贡献一部分信息（私密的 a 和 b），通过混合公共参数（g, p），最终生成相同的密钥，而不需要直接传输密钥本身 [cite: 124, 130][cite_start]。课件中用“颜料混合”的例子非常形象地说明了这一点：你可以看到混合后的颜色，但无法分离出原来的私密颜色 [cite: 128, 129]。
  - [cite_start]**局限性：** 原始的 DH 协议不具备身份验证功能，如果不配合签名机制，容易遭受“中间人攻击”（Man-in-the-Middle Attack）[cite: 156]。
- **RSA: 全能型选手**
  - [cite_start]**核心功能：** RSA 既可以用于**加密/解密**，也可以用于**数字签名** [cite: 81, 82, 210]。

- [cite_start]**过程特点：** 它依赖于公钥（Public Key）和私钥（Private Key）的配对。公钥加密的信息只有私钥能解（保密性），私钥加密的信息（即签名）公钥能解（认证性/不可抵赖性） [cite: 77, 81]。

(2) 数学难题的基础 (Trapdoor)

特性	Diffie-Hellman (DH)	RSA
数学基础	[cite_start]**离散对数问题 (DLP):** 给定 g 和 $g^x \mod p$，很难求出 x [cite: 116, 151]。	[cite_start]**大整数分解问题:** 给定 p 和 q [cite: 201, 237]。
计算开销	计算幂模运算，相对较快，但对长报文加密依然太慢。	[cite_start]涉及大数幂运算，计算 [cite: 61]。

2. 深度剖析：端到端加密 (E2EE) 中的选择策略

在构建像 WhatsApp 这样的端到端加密应用时，为什么我们倾向于使用 **Diffie-Hellman (特别是 Ephemeral DH)** 来协商密钥，而不是直接用 RSA 传输密钥？这里有两个关键因素：**性能**和**前向保密性**。

(1) 性能与混合加密模式

[cite_start]课件中提到，非对称加密通常太慢，不适合处理大量数据 [cite: 61]。

- [cite_start]**策略：** 我们不直接用 RSA 加密聊天内容。相反，我们使用非对称算法（如 DH）来协商出一个**对称会话密钥（Symmetric Session Key）** [cite: 63]。
- **结果：** 之后的大量数据传输（视频、文本）都使用这个对称密钥进行加密（如 AES），兼顾了安全性和速度。

(2) 决定性因素：完美前向保密 (Perfect Forward Secrecy)

这是现代安全通信（如 SSL/TLS 和 WhatsApp）选择 DH 的核心原因。

- **RSA 的风险（静态密钥）：** 如果你使用固定的 RSA 密钥对来传输会话密钥。一旦攻击者在未来某天偷走了你的**长期私钥**，他就可以解密过去截获的所有加密流量。这意味着“一朝被破，全盘皆输”。
- [cite_start]**DH 的优势（临时密钥）：** 课件特别提到了 **Ephemeral Diffie-Hellman (DHE)** [cite: 163]。
  - [cite_start]在 DHE 中，每次会话都生成新的、临时的公钥/私钥对 [cite: 164]。

- 会话结束后，这些临时密钥就被销毁。
- [cite_start]**结果：** 即使攻击者获得了你的长期身份私钥，他也无法解密之前的历史消息，因为解密需要的是当时那个已经被销毁的临时密钥。这就实现了**完美前向保密 (Perfect Forward Secrecy)** [cite: 158, 167]。

3. 总结与应用建议

回到你的问题，我们可以这样决策：

1. [cite_start]**如果你的目标是“密钥协商”且看重“前向保密”：** 首选 **Ephemeral Diffie-Hellman (DHE 或 ECDHE)**。这是现代 HTTPS 和即时通讯软件（如 WhatsApp [cite: 169]）的标准做法。
2. [cite_start]**如果你的目标是“身份验证”或“数字签名”：** 必须使用 **RSA**（或 DSA/ECDSA）。因为 DH 本身无法证明“你是谁”，必须配合 RSA 签名来防止中间人攻击 [cite: 165]。
3. **混合使用：** 最强大的方案通常是结合两者——用 RSA（或类似算法）来**签名** DH 的参数以验证身份，然后用 DH 来**协商**会话密钥以保证通信内容的机密性和前向安全性。

让我们来验证一下你的理解：

[cite_start]课件的第 36 页提到，基础的 Diffie-Hellman 协议容易受到**中间人攻击 (Man-in-the-Middle Attack)** [cite: 156]。

基于我们刚才的分析，你能推测出为什么单纯的 DH 无法防御这种攻击，以及我们需要引入什么机制（可能涉及 RSA）来解决这个问题吗？

7. 密码学计算步骤

这份这是为您整理的完整版《密码学算法计算学习笔记》。

这份笔记整合了 **RSA 密钥生成、扩展欧几里得算法（求私钥的核心）** 以及 **Diffie-Hellman 密钥交换** 的完整流程、公式推导、手算示例和代码逻辑。

密码学算法计算学习笔记

第一部分：RSA 算法 (RSA Algorithm)

1. RSA 密钥生成步骤

RSA 的安全性依赖于大整数分解的困难性。

1. **选择素数**：
   选择两个不相等的大素数 p 和 q。
2. **计算模数 (Modulus)**：

$$n = p \times q$$

   - n 的长度即为密钥长度（如 2048 位）。
   - n 是公钥和私钥的一部分。
3. **计算欧拉函数 (Euler's Totient)**：

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

4. **选择公钥指数 (Public Exponent)**：
   选择整数 e，满足以下两个条件：
   - $1 < e < \phi(n)$
   - $\gcd(e, \phi(n)) = 1$ （e 与 $\phi(n)$ 互质）
5. **计算私钥指数 (Private Exponent)**：
   计算 d，使得 d 是 e 模 $\phi(n)$ 的乘法逆元：

$$d \times e \equiv 1 \pmod{\phi(n)}$$

   - **计算方法**：使用**扩展欧几里得算法**（见下文详解）。

最终密钥结构：

- **公钥 (Public Key)**: $PU = \{e, n\}$
- **私钥 (Private Key)**: $PR = \{d, n\}$

2. RSA 加密与解密公式

- **加密**: $C = M^e \pmod{n}$（M 为明文，C 为密文）
- **解密**: $M = C^d \pmod{n}$

---

3. 核心工具：扩展欧几里得算法 (求 RSA 私钥)

场景：已知 e 和 $\phi(n)$，求解 d。
目标：找到 d，满足 $d \cdot e + k \cdot \phi(n) = 1$（即 $d \cdot e \pmod{\phi(n)} = 1$）。

方法 A：手算步骤 (回代法) —— *考试推荐*

示例数据：$e = 7, \phi(n) = 160$。

第1步：辗转相除 (求 GCD)

建立等式：$A = q \times B + r$

1. $160 = 22 \times 7 + 6 \Rightarrow$ 移项备用：$6 = 160 - 22 \times 7$
2. $7 = 1 \times 6 + 1 \Rightarrow$ 移项备用：$1 = 7 - 1 \times 6$
3. $6 = 6 \times 1 + 0$ (结束)

第2步：逆向回代 (Back Substitution)

从余数为 1 的等式开始往回代入：

1. 写出末尾等式：

$$1 = 7 - 1 \times \mathbf{6}$$

2. 将第1步中 6 的表达式代入：

$$1 = 7 - 1 \times (\mathbf{160} - \mathbf{22} \times \mathbf{7})$$

3. 整理合并（把 160 和 7 当作变量，不要算出具体数值）：

$$1 = 7 - 1 \times 160 + 22 \times 7$$

$$1 = (1 + 22) \times 7 - 1 \times 160$$

$$1 = \mathbf{23} \times 7 - 1 \times 160$$

4. **结论**：
   系数 23 即为 d。

$$d = 23$$

> **重要提示**：如果算出的系数是负数（例如 -23），则 $d = -23 + \phi(n)$。

方法 B：程序化逻辑 (迭代法) —— *理解代码用*

```
# 扩展欧几里得算法求私钥 d
# 输入: e, phi_n
def get_private_key(e, phi_n):
    # 初始化
    r_old, r_new = phi_n, e   # 余数序列，对应公式中的 a 和 b
    t_old, t_new = 0, 1       # 系数序列，对应求出的逆元

    # 循环直到余数为 0
    while r_new > 0:
        quotient = r_old // r_new  # 计算商

        # 更新余数 (r_old - quotient * r_new)
        r_old, r_new = r_new, r_old - quotient * r_new

        # 更新系数 (t_old - quotient * t_new)
        # 这是核心：系数的变化规律与余数完全一致
        t_old, t_new = t_new, t_old - quotient * t_new

    # 循环结束，r_old 为 GCD (必须为1)
    if r_old > 1: return "Error: Not Coprime"

    # t_old 即为逆元，处理负数情况
    d = t_old
    if d < 0: d = d + phi_n

    return d
```

4. RSA 完整计算示例与伪代码

假设：$p = 17, q = 11$。

```python
# --- RSA 完整流程 ---

# 1. 初始参数
p = 17
q = 11

# 2. 计算 n
n = p * q
# n = 187

# 3. 计算 phi(n)
phi_n = (p - 1) * (q - 1)
# phi_n = 16 * 10 = 160

# 4. 选择公钥指数 e
# 条件: 1 < e < 160 且 gcd(e, 160) = 1
e = 7

# 5. 计算私钥指数 d
# 使用上面的扩展欧几里得算法求解:
# 7*d = 1 mod 160
# ... (经过计算) ...
d = 23

# --- 最终结果 ---
# 公钥 PU = {7, 187}
# 私钥 PR = {23, 187}

# --- 加密测试 ---
# 假设明文 M = 88
# C = M^e mod n
C = (88 ** 7) % 187  # C = 11

# --- 解密测试 ---
# M = C^d mod n
M_decrypted = (11 ** 23) % 187 # M = 88 (恢复原文明文)
```

第二部分：Diffie-Hellman 密钥交换 (DH Key Exchange)

1. 算法原理

DH 用于在不安全的通道上让双方协商出一个**共享密钥 (Shared Secret)**，其安全性基于**离散对数问题**。

2. 计算步骤

1. **公开参数**：
   - q：大素数。
   - α：q 的本原根。
2. **Alice 生成**：
   - 私钥 $X_A < q$。
   - 公钥 $Y_A = \alpha^{X_A} \pmod q$。
3. **Bob 生成**：
   - 私钥 $X_B < q$。
   - 公钥 $Y_B = \alpha^{X_B} \pmod q$。
4. **计算共享密钥 K**：
   - Alice 计算: $K = (Y_B)^{X_A} \pmod q$
   - Bob 计算: $K = (Y_A)^{X_B} \pmod q$

3. 公式推导 (证明一致性)

我们需要证明 Alice 和 Bob 算出的是同一个数。

- Alice 端: $K = (Y_B)^{X_A} \pmod q = (\alpha^{X_B})^{X_A} \pmod q = \alpha^{X_B \cdot X_A} \pmod q$
- Bob 端: $K = (Y_A)^{X_B} \pmod q = (\alpha^{X_A})^{X_B} \pmod q = \alpha^{X_A \cdot X_B} \pmod q$
- 因为乘法交换律 $X_A \cdot X_B = X_B \cdot X_A$，得证。

4. DH 计算示例与伪代码

假设：$q = 353, \alpha = 3$。

```
# --- Diffie-Hellman 流程 ---

# 1. 公开参数
q = 353
alpha = 3

# 2. Alice 操作
X_a = 97                  # Alice 私钥 (随机选择)
Y_a = (alpha ** X_a) % q # Alice 公钥: 3^97 mod 353 = 40

# 3. Bob 操作
X_b = 233                 # Bob 私钥 (随机选择)
Y_b = (alpha ** X_b) % q # Bob 公钥: 3^233 mod 353 = 248

# --- 交换公钥后计算共享密钥 ---

# Alice 计算 K
# 使用 Bob 的公钥 (248) 和自己的私钥 (97)
K_alice = (Y_b ** X_a) % q
# K_alice = 248^97 mod 353 = 160

# Bob 计算 K
# 使用 Alice 的公钥 (40) 和自己的私钥 (233)
K_bob = (Y_a ** X_b) % q
# K_bob = 40^233 mod 353 = 160

# 验证: K_alice == K_bob == 160
```

8、对称加密算法深入分析 (DES)

根据您提供的课件（*Network Security Technology - Symmetric cryptography*），以下是关于DES（数据加密标准）的深入分析和具体计算步骤详解。

第一部分：DES 的局限性与核心组件分析

根据课件（Page 18-19, 47），DES 虽然曾是世界标准，但存在明显的安全隐患：

1. DES 的局限性 (Limitations & Weaknesses)

- **密钥长度过短（最主要弱点）**：
  - DES 的密钥虽然表面是 64 位，但实际有效密钥只有 **56 位**（每 8 位有一个校验位被丢弃）。
  - **暴力破解风险**：密钥空间为 2^{56}（约 7.2 亿亿）。虽然数字很大，但在 1998 年，EFF（电子前哨基金会）制造的 "Deep Crack" 机器仅用 **4 天** 就暴力破解了 DES 密钥。现代计算机甚至更快。
- **算法公开透明**：
  - S 盒、置换表（P-table）等都是公开的，安全性完全依赖于密钥的保密（柯克霍夫原则）。
- **置换的无用性**：
  - 课件提到初始置换（IP）和最终置换（Final Permutation）在安全性上几乎“无用”（Useless），它们主要是为了适应早期电子设备的硬件实现，不增加密码学强度。
- **对称性**：
  - 加密和解密使用相同的算法和密钥（只是子密钥顺序相反），一旦密钥泄露，通信完全崩塌。

2. DES 子密钥生成 (Subkeys Generation)

DES 需要进行 16 轮加密，因此需要从原始密钥中生成 16 个 48 位的子密钥 (k_1 到 k_{16})。

- **输入**：64 位原始密钥。
- **步骤 1：PC-1 置换（去除校验位）**
  - 根据 PC-1 表，丢弃第 8, 16, 24, 32, 40, 48, 56, 64 位。
  - 剩余 56 位分为两部分：前 28 位为 C_0（课件记为 A_0），后 28 位为 D_0（课件记为 B_0）。
- **步骤 2：循环左移 (Left Shifts)**
  - 对每一轮 i（1 到 16），对两部分分别进行循环左移。
  - **移位规则**（Page 31）：第 1, 2, 9, 16 轮左移 **1 位**；其余轮次左移 **2 位**。
- **步骤 3：PC-2 置换（压缩）**
  - 将移位后的 56 位输入 PC-2 表，选择其中的 48 位作为该轮的子密钥 k_i。
- **输出**：16 个 48 位的子密钥。

3. DES 扩展函数 (Expansion Function)

这是 f 函数的第一步，目的是让数据的右半部分（32 位）能与子密钥（48 位）进行异或运算，并产生“雪崩效应”。

- **输入**：32 位数据（来自上一轮的右半部分 R_{i-1}）。
- **过程**：使用扩展置换表（E-table），将 32 位扩展为 48 位。
- **原理**（Page 38）：表中有一些位被重复使用。例如，第 32 位可能同时出现在输出的第 1 位和第 47 位。
- **作用**（Page 37）：
  - **匹配长度**：32 位 $\rightarrow$ 48 位，以便与 48 位子密钥进行 XOR。
  - **雪崩效应 (Avalanche Effect)**：输入的一位变化会影响输出的两位，进而影响后续 S 盒的多个输出，使得输出剧烈变化，增加破解难度。

第二部分：DES 算法具体步骤详解

DES 是一个 **Feistel 结构** 的分组密码，处理 **64 位明文块**，使用 **56 位密钥**。

步骤 1：初始置换 (Initial Permutation, IP)

- **操作**：将输入的 64 位明文块按照固定的 IP 表打乱顺序。
- **例子**（Page 22）：
  - 输入块的第 58 位 $\rightarrow$ 变成输出的第 1 位。
  - 输入块的第 50 位 $\rightarrow$ 变成输出的第 2 位。
- **结果**：乱序后的 64 位数据，被分为左右两半：
  - L_0（左 32 位）
  - R_0（右 32 位）

步骤 2：16 轮迭代 (16 Rounds)

这是 DES 的核心。每一轮（第 i 轮，从 1 到 16）处理方式相同。

Feistel 结构公式（Page 14）：

- $L_i = R_{i-1}$ （下一轮的左半部分 = 上一轮的右半部分）
- $R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$ （下一轮的右半部分 = 上一轮左半部分 XOR f 函数的结果）

f 函数（轮函数）的具体步骤（Page 35-44）：
f 函数接收 32 位的 R_{i-1} 和 48 位的子密钥 k_i。

1. **扩展置换 (E)**：
   - 将 32 位的 R_{i-1} 扩展为 48 位（见上文扩展函数分析）。
2. **密钥加 (Key Mixing)**：
   - 将扩展后的 48 位数据与 48 位的子密钥 k_i 进行 **XOR (⊕)** 运算。
3. **代换 (Substitution) —— S-Box (核心步骤)**：
   - **输入**：XOR 后的 48 位数据，被分成 **8 组**，每组 **6 位**。
   - **处理**：每 6 位通过一个对应的 S-Box（S1 到 S8），输出 **4 位**。
   - **S-Box 查找规则（举例）**（Page 41-42）：
     - 假设进入 S1 盒的 6 位是 `101010`。
     - **行号**：取首尾 2 位 `1` 和 `0` $\rightarrow$ 二进制 `10` = **第 2 行**。
     - **列号**：取中间 4 位 `0101` $\rightarrow$ 二进制 `0101` = **第 5 列**。
     - **查表**：在 S1 盒的第 2 行第 5 列找到数值。假设课件中该值为 6。
     - **输出**：将 6 转为 4 位二进制 `0110`。
   - **结果**：8 个 S 盒的输出拼在一起，共 $8 \times 4 = 32$ 位。
4. **置换 (Permutation, P)**：
   - 将 S 盒输出的 32 位数据，通过 P-Box 进行最后一次打乱。
   - **输出**：f 函数的最终 32 位结果。

本轮结束：
将 f 函数的输出与 L_{i-1} 进行 XOR，得到 R_i。

步骤 3：最终置换 (Final Permutation, IP⁻¹)

- **操作**：16 轮结束后，得到 L_{16} 和 R_{16}。注意：标准 DES 在输出前会交换左右两半，即合并为 $R_{16}L_{16}$。
- **处理**：使用 IP 表的逆表 (IP^{-1}) 进行置换。
- **结果**：64 位密文。

总结 DES 加密流程图解

```
明文 (64 bit)
    ↓
初始置换 (IP)
    ↓
分割 -> L0 (32 bit), R0 (32 bit)
    ↓
----------- 第 1 轮 -----------
    L1 = R0
    R1 = L0 XOR f(R0, K1)
        [f函数细节: E扩展 -> XOR K1 -> S盒压缩 -> P置换]
    ↓
... (重复 15 次) ...
    ↓
----------- 第 16 轮 ----------
    L16 = R15
    R16 = L15 XOR f(R15, K16)
    ↓
合并 (注意通常是 R16 L16)
    ↓
最终置换 (IP逆)
    ↓
密文 (64 bit)
```

为了让你直观地理解 DES 的计算过程，我们很难手动演示完整的 16 轮（那需要写几千个 0 和 1）。

最好的方法是： 我们设定一个场景，重点演示 **“第 1 轮加密”** 中的核心步骤，特别是 f **函数** 和 **S-盒代换** 的细节。

场景设定 (Scenario)

- **明文 (Plaintext, 64-bit)**: `0123456789ABCDEF` (十六进制)
  - 二进制: `0000 0001 ... 1110 1111`
- **子密钥 (Subkey K_1, 48-bit)**: 假设这是通过密钥生成算法生成的第 1 轮子密钥。

- 假设 K_1 (十六进制): `1B02EFF34567`
- 二进制: `0001 1011 0000 0010 1110 1111 1111 0011 0100 0101 0110 0111`

第一步：初始置换 (IP)

64 位明文经过 IP 表打乱，并被切分为左右两部分（各 32 位）。

- **原始明文**: `0123456789ABCDEF`
- **经过 IP 置换后 (假设结果)**:
  - L_0 **(左 32 位)**: `CC00CC00` (二进制 `1100 1100 ...`)
  - R_0 **(右 32 位)**: `F0AAF0AA` (二进制 `1111 0000 1010 1010 1111 0000 1010 1010`)

第二步：核心迭代 (第 1 轮)

目标：计算下一轮的 L_1 和 R_1。
公式:

1. $L_1 = R_0$ （直接赋值）
2. $R_1 = L_0 \oplus f(R_0, K_1)$ （这是难点）

我们重点计算 $f(R_0, K_1)$:

1. 扩展置换 (Expansion, E)

将 32 位的 R_0 扩展为 48 位。

- **输入** R_0: `F0AAF0AA`
- **二进制**: `111100 001010 101011 110000 101010 ...` (为了便于观察，E 扩展通常是将某些位复制)
- **扩展后** $E(R_0)$ **(48 位)**:
  - 假设结果为: `011110 100001 010101 010101 ...` (此处为模拟逻辑)
  - *注：扩展是把上一组的最后一位和下一组的第一位放到当前组，比如中间的 4 位变成 6 位。*

2. 与子密钥异或 (Key Mixing)

将扩展后的 $E(R_0)$ 与子密钥 K_1 进行异或 ($\oplus$)。

```
  E(R0): 011110 100001 010101 ... (48位)
  K1   : 000110 110000 001011 ... (48位，取自我们设定的假设值)
----------------------------------------
  XOR结果: 011000 010001 011110 ... (48位)
```

我们将这 48 位结果分成 **8 组**，每组 **6 位**。我们拿**第一组**来演示最重要的 S-盒代换。

3. S-盒代换 (Substitution) —— 最关键步骤

假设异或后的**第一组 6 位**数据是：`011011`。
我们将使用 **S1 盒** (S-Box 1) 来处理这组数据。

S1 盒内容 (参考课件 Page 42):

行\列	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	1
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	1

计算过程：

- **输入**: `011011`
- **计算行号 (Row)**: 取第 1 位和第 6 位 → `0` 和 `1` → 二进制 `01` = **第 1 行。**
- **计算列号 (Col)**: 取中间 4 位 → `1101` → 十进制 **13** = **第 13 列。**

查表:

- 在 S1 表中，找到 **第 1 行、第 13 列** 的数字。
- 查表得数值：**5。**

输出:

- 将 5 转换为 **4 位二进制**：`0101`。

这就是 S1 盒的输出。对剩下的 7 组 6 位数据做同样的操作，分别查 S2 到 S8 盒。
最终得到 $4 \times 8 = 32$ 位的 S 盒输出。

4. P-盒置换 (Permutation)

将 S 盒输出的 32 位数据再次打乱顺序（只是改变位置）。

- 假设 S 盒总输出: `0101 ...`
- 经过 P 置换后: `1010 ...` (这就是 f 函数的最终结果)。

第三步：生成第 1 轮结果

现在我们有了 $f(R_0, K_1)$ 的结果（32 位），回到 Feistel 公式：

1. **计算 R_1**:
   - L_0: `CC00CC00` (32 位)
   - $f(R_0, K_1)$: `1010...` (32 位, 上一步算出来的)
   - $R_1 = L_0 \oplus f(R_0, K_1)$
2. **计算 L_1**:
   - $L_1 = R_0$ = `F0AAF0AA`

第 1 轮结束时的数据:

- 左半部分: `F0AAF0AA`
- 右半部分: `L0 XOR f结果`

接下来，将这两个部分作为输入，重复上述过程进行第 2 轮，直到第 16 轮。

总结：在这个例子中你需要记住什么?

如果考试让你举例或计算，通常只会考查 **S-盒** 的部分，因为其他部分只是机械的移位或异或。

必记的 S-Box 计算口诀:

> **“首尾定行，中间定列”**
>
> - Input (6 bit): **B1** B2 B3 B4 B5 **B6**
> - Row = **B1 B6** (转十进制)
> - Col = B2 B3 B4 B5 (转十进制)
> - Output = 表中对应的值 (转 4 bit 二进制)

9、古典密码与模式问题：凯撒密码原理、块密码的原理及其模式问题、模式问题的解决方案，如采用如密码块链接、计数器等更安全的模式。

根据提供的课件（Introduction to Cryptography），针对“古典密码与模式问题”的解答如下：

1. 凯撒密码原理 (Caesar Cipher Principle)

- **基本概念**：凯撒密码是一种**单表代换密码 (Monoalphabetic Substitution Cipher)**（课件第16页）。
- **工作原理**：
  - 将字母表中的每个字母按照固定的数量（k）进行左移或右移，从而得到密文。
  - 通常忽略空格，将字母组成固定长度的块。
- **数学表达**（课件第16页）：
  - **加密**：$\mathcal{E}_k : i \rightarrow i + k \mod 26$
  - **解密**：$\mathcal{D}_k : i \rightarrow i - k \mod 26$
- **安全性**：安全性较低，仅需通过**字母频率分析 (Letter frequency analysis)** 即可破解（课件第17页）。

2. 块密码原理及其模式问题 (Block Cipher Principle & Mode Problems)

块密码原理

- **定义**：属于对称密钥加密。它将明文分割成**固定长度的块 (Fixed-length block)**（通常

$n = 64$位），并使用用户提供的密钥将每个明文块转换为相同长度的密文块（课件第25页）。
- **解密**：使用相同的密钥对密文块应用逆变换。

模式问题 (以 ECB 为例)

- **ECB (电子密码本模式)**：这是最基本的模式。消息被分成块，每个块**独立**加密：$c_j = E_k(m_j)$（课件第37页）。
- **存在的问题**：
  - 由于每个块是独立加密的，且使用的是相同的密钥，**相同的明文块会产生完全相同的密文块。**
  - 这意味着如果明文中存在重复的模式，密文中也会显现出来，这无法掩盖数据中的统计规律，容易受到重放攻击或模式分析攻击。

3. 模式问题的解决方案 (Solutions / Safer Modes)

为了解决ECB模式中“相同明文生成相同密文”的问题，课件介绍了以下几种更安全的模式，它们引入了“反馈”或“链接”机制：

A. 密码块链接 (CBC: Cipher-Block Chaining)

- **原理**：在加密之前，当前的明文块会先与**前一个密文块**进行异或 (XOR) 运算（课件第38页）。
- **第一块的处理**：对于第一个明文块，使用一个**初始化向量 (Initializing Vector, IV)** 进行异或：$c_1 = E_k(m_1 \oplus IV)$。
- **公式**：$c_j = E_k(m_j \oplus c_{j-1})$。
- **效果**：即使明文块相同，由于前一个密文块不同，生成的当前密文块也会不同。

B. 密文反馈 (CFB: Cipher FeedBack)

- **原理**：将块密码转换为流密码使用。它使用一个移位寄存器（初始为IV）。
- **过程**：加密移位寄存器中的内容生成中间密文，取其左侧 r 位与明文进行异或得到密文。然后将密文反馈回移位寄存器（课件第40页）。
- **公式**：$C_i = E_k(C_{i-1}) \oplus P_i$（注：此处简化表达，具体涉及移位操作）。

C. 输出反馈 (OFB: Output FeedBack)

- **原理**：与CFB类似，但反馈机制不同。

- **过程**：加密移位寄存器产生中间输出，将该输出（而不是密文）反馈回移位寄存器，同时也用该输出与明文异或生成密文（课件第42页）。
- **公式**：$C_i = P_i \oplus O_i$；$O_i = E_k(O_{i-1})$。
- **特点**：产生的密钥流与明文无关。
-

10. 哈希函数的安全性

根据提供的课件（*Network Security Technology - Hash functions and message authentication codes*），针对您提出的关于哈希函数安全性的两个问题，解答如下：

1. 碰撞抵抗 (Collision Resistance)

含义：
碰撞抵抗是指：虽然在理论上哈希碰撞（即两个不同的输入产生相同的哈希值）是必然存在的，但在计算上，要**找到**任意两个不同的输入 x 和 x'，使得它们的哈希值 $h(x) = h(x')$ 是**极其困难的（infeasible）**。

基于课件的详细解释：

- **必然存在性 (鸽巢原理)**：课件第11页提到，由于输入空间（任意长度的消息）远大于输出空间（固定长度 L 的哈希值），根据鸽巢原理（Pigeonhole Principle），必然存在不同的输入映射到相同的输出。
- **计算困难性**：课件第13页定义了“抗碰撞性”（Collision-resistant）。它并不意味着碰撞不存在（有时也称为“无碰撞/collision-free”，但这只是术语），而是指通过现有的计算能力和密码学手段，很难在合理的时间内搜索到这样的一对碰撞。
- **与原像抗性的区别**：这与“原像抗性”（Pre-image resistance，给定哈希值找输入）不同。抗碰撞性关注的是找到**任意**一对冲突，而不需要指定特定的哈希值（课件第13页）。

2. 短哈希值的问题 (Problems with Short Hash Values)

核心风险：
使用短哈希值（例如16位）会面临严重的**生日攻击（Birthday Attack）**风险，导致极其容易发生哈希碰撞，从而破坏数据的完整性。

基于课件的详细解释：

- **生日悖论 (Birthday Paradox)**：课件第20-25页介绍了生日悖论。即使哈希值的空间很大（N），找到两个具有相同哈希值的输入所需的尝试次数并非 $N/2$，而是大约 $\sqrt{N}$（即 $N^{1/2}$）。
- **攻击复杂度**：课件第26页指出，对于输出长度为 L 位的哈希函数，生日攻击找到一个碰撞的时间复杂度仅为 $O(2^{L/2})$。
- **16位哈希的具体计算**：
  - 如果哈希长度 $L = 16$ 位。
  - 哈希值的总空间为 $2^{16} = 65,536$。
  - 根据生日攻击公式 $2^{L/2}$，攻击者只需要尝试大约 $2^{16/2} = 2^8 = \mathbf{256}$ 次，就有很大概率找到两个哈希值相同的不同输入。
  - 这意味着对于计算机而言，破解16位哈希并在几毫秒内伪造数据是轻而易举的。
- **课件结论**：课件第28页明确指出，即使是64位的哈希值（2^{32} 复杂度）也被认为太小而无法抵御生日攻击。因此，现代哈希通常要求至少128位（如MD5，虽已被破）或160位（如SHA-1，也不再安全）甚至更高（如SHA-256），以确保 $2^{L/2}$ 足够大，使计算不可行。