

《密码学及应用》零基础学习手册

按中南大学计算机学院网络空间安全系 段桂华 主讲课件整理 | 每一步都从"为什么要这样"讲起 | 左侧可搜索, 每节末尾有自测 (点开看答案)

开始之前

这本手册怎么用

先说一件事

如果你打开密码学课本, 第一反应是"这堆符号是什么鬼"——这不是你的问题, 是课本的写法问题。

课本习惯先给定义再给公式, 可人是靠"先遇到问题、再找办法"来理解的。这本手册反过来: 先讲一个具体的难题, 再告诉你密码学家怎么解的。

三条建议

建议	为什么
① 别背公式, 先懂它想解决什么	比如"欧拉定理", 看着吓人, 其实就是一句大白话: 在某个圈里一直转, 转够圈数一定回到起点。懂了这句, 公式自己就记住了。
② 每个概念亲手算一遍	密码学的数学都很小 (模 26、模 17 这种), 拿纸笔算两三个数就通了。光看不算, 等于没学。
③ 每节末尾的自测要真做	那些题都是"看懂了但一动手就卡"的经典卡点。做出来才算会。

怎么读那些小标签

手册里会出现几种彩色方框, 各有分工:

- 生活场景** 一个具体的故事, 告诉你"这个问题从哪来"
- 为什么需要它** 这个概念是为了解决什么难题才被发明出来的
- 大白话** 把抽象的数学用日常语言再说一遍
- 动手做** 一步一步带你算, 建议拿纸笔跟着走
- 小心坑** 最容易出错的地方
- 常见误解** "看着像对、其实错"的说法
- 前后呼应** 这个知识后面会在哪里再用到

学习路线建议

如果时间紧, 第 1 章 (尤其是 1.4 数学基础) 必须吃透——后面每一章都在用它。

第 2 章是"算法大观园", 认得出名字、知道它靠什么数学难题撑着就够了。

第 3~5 章是"协议", 核心是理解为什么要这样设计, 步骤记不清没关系, 能说清设计思路就有分。

第 6、7 章相对独立, 随时可以看。

这份手册和"考前速查"有什么区别

两件事, 两份材料:

材料	什么时候用
本手册（零基础学习版）	平时学——每个概念都从"为什么需要它"讲起，带场景、比喻、手把手演算、自测题
开卷考试宝典（配套的另一份）	考试时查——一页纸公式表、八大解题模板、易错点清单，直接照抄卷面格式

建议的用法：先用这份手册把概念搞懂（特别是 1.4 节），再拿那份速查去练手算和背公式。

第 0 章

密码学到底在干什么

0.1 一封信的烦恼

生活场景

Alice 要给 Bob 寄一封信，内容只有一个字：

明天三点，会议室见

问题是：这封信要经过一条大马路，路上谁都能把信拆开看一眼。

怎么才能让路人看不懂？

唯一的办法：把内容改写掉

Alice 想到了一个主意——把每个字母往后挪几位：

原话：meet me

规则：每个字母往后挪 1 位

结果：nffu nf

路人看到 nffu nf，一脸茫然。Bob 收到后，只要按同样规则往回挪 1 位，就还原了：

nffu nf → 往回挪 1 位 → meet me ✓

大白话

密码学的全部内容，就是这件事：

可读的内容 → 改写成看不懂的样子 → 传输 → 改回来 → 可读的内容

改写叫加密，改回来叫解密，改写后的样子叫密文，原来的样子叫明文。

没有别的了。后面所有的算法、协议、数学，都只是在想办法把这件简单的事做得更结实。

但这里有个致命问题

如果“往后挪 1 位”这个规则是固定的，那路人**多看几封信也能猜出来**。

所以他得往规则里掺一个只有他俩知道的秘密——这个秘密就是密钥。

同样是 meet：

用密钥 1 → nffu

用密钥 3 → phhw

用密钥 11 → xpp e ...

同样的规则，不同的密钥，结果完全不同。路人就算知道“你们是把字母往后挪”，也不知道挪几位。

0.2 密钥才是命根子

为什么需要它

假设你是 Alice，你有两样东西可以保密：

选择一：把“方法”藏起来。

比如自创一套“把每个字换成字典里的页码”，谁也不告诉。

选择二：把“参数”藏起来。

方法公开：每个字母往后挪 N 位。但 N 是几，只有你和 Bob 知道。

哪种更靠得住？ 答案是第二种。原因有三条。

理由一：方法根本藏不住

你要把加密软件装到成千上万台电脑上。每台电脑里都有你的算法。只要有一台被拆开看，方法就泄露了。

既然迟早会泄露，不如**主动公开**——这样还能让全世界的密码学家来帮你找漏洞。

大白话

锁的构造图可以公开（让大家检查它结不结实），但**钥匙必须自己藏好**。

这就是现代密码和古典密码的分水岭。香农（信息论之父）有句话就是这个意思：

数据的安全基于密钥，而不是密码算法。

理由二：秘密越少越好管

方案	要看住几样东西
算法保密	① 整套算法（可能几百行代码）② 密钥 ③ 所有知道算法的人
算法公开	① 密钥（就一串数字）

管理一串数字，比管理一整套方法容易太多。

理由三：公开能换来全世界的免费审查

- 算法公开 → 全世界的密码学家都来尝试攻破 → **有人发现漏洞就立刻改**，算法越来越强
- 算法保密 → 只有你自己知道它有没有漏洞 → 万一有洞，要等到**被人攻破了**才知道，那时已经晚了

常见误解

误解：“算法公开了，那大家不就能破解了吗？”

纠正：不会。算法公开只是告诉攻击者“我用的是哪把锁”，但他**还是没有钥匙**。安全性完全由密钥承担——只要密钥够长够随机，消息就是安全的。

所以有两个推论：① **密钥要足够长**（不然能穷举）；② **密钥不能弱**（不能是生日、名字、123456）。这就是为什么课件要用整整第 7 章讲“密钥管理”。

▼ 自测：如果 Alice 的加密算法被公开了，她和 Bob 的通信还安全吗？

安全。 因为算法的公开**本身**不降低安全性，它只是把安全的全部责任交给了密钥。只要密钥足够长、足够随机、只有他俩知道，那通信就是安全的。

注意措辞：是"安全"，不是"基本安全"。因为这是定义决定的——算法公开是一个**主动的设计选择**，不是妥协。

0.3 攻击者会怎么下手

为什么需要它

要知道怎么设计密码，先得知道攻击者会怎么打。密码学是一门"攻防"学问——你防守的思路，全部来自攻击者的手段。

攻击者手里的"情报"越多，攻击越容易。按情报多少，分四档：

攻击类型	攻击者手里有什么	难度
唯密文攻击	只截到一堆看不懂的密文	最难
已知明文攻击	知道几组"明文 ↔ 密文"的对应关系	中等
选择明文攻击	能 自己挑内容 让对方加密，再看结果	最容易
针对密钥攻击	目标不是读消息，而是 直接拿到密钥	(主要针对公钥密码)

用具体例子体会一下

场景演示

假设还用"每个字母往后挪 N 位"这个规则。

① **唯密文攻击**：路人只看到 `nffu nf`。他只能**猜**——试试 N=1? N=2? 一个个试过去。
(如果字母只有 26 个，最多试 26 次。这就是**穷举攻击**。)

② **已知明文攻击**：路人偷看到一张便签，上面写着" `meet` 加密后是 `nffu` "。
他一看：`m→n, e→f, t→u`。**每个字母都往后挪了 1 位！** 密钥当场暴露。

③ **选择明文攻击**：这是最狠的。路人能让你加密任意内容。他让你加密一个 `a`，你给出 `d`。
他立刻知道：**a 变成了 d，往后挪了 3 位。** 一次就破。

大白话

攻击者的逻辑很朴素：你**只要泄露了"输入"和"输出"的对应关系**，我就能反推出规则。

所以密码设计者的任务不是"让攻击者猜不着"，而是——**让他手里有再多的输入输出对，也推不出密钥。**

还有两种具体的攻击手段

手段	做法	为什么有效
穷举攻击	把所有可能的密钥都试一遍	密钥空间小时非常有效；所以密钥要够长
字典攻击	用常见单词的组合来猜	因为很多人用生日、名字当密码（见 7.1 节）

▼ 自测：下面四种情况分别属于哪种攻击？

- A. 路人只有一堆看不懂的密文。
- B. 路人知道"meet" 加密后是 "nffu"。
- C. 路人能自己挑一个字让你加密，然后看结果。
- D. 路人专门去推算你用的那个"挪几位"。

答：A 唯密文攻击；B 已知明文攻击；C 选择明文攻击；D 针对密钥攻击。

补充：A/B/C 是按"手里有什么情报"分的，D 是按"攻击目标"分的。D 的目标不是读出某条消息，而是**直接拿到密钥**——所以课件说它"主要针对公钥密码系统"。

0.4 整门课三条主线

现在回头看，整门课其实就围绕三个问题转：

问题	对应课件	一句话概括
① 怎么改写才安全？	第 1、2 章（算法）	设计足够复杂的"改写规则"
② 攻击者会怎么打？	各章的安全性分析	每一步都要经得起这四种攻击
③ 密钥怎么给到对方？	第 3~7 章（协议与管理）	这是密码学的"死穴"，也是诞生公钥密码的原因

前后呼应

第三个问题特别重要，值得先剧透一下：

如果 Alice 和 Bob 要通信，他们得先共享一个密钥。可密钥本身怎么安全地送过去呢？寄过去会被截获，加密寄又需要一个密钥——**死循环**。

古典密码的解法是**"事先当面约好"**（所以只能熟人之间用）。

而 1976 年 Diffie 和 Hellman 想出了一个绝妙的办法：让双方在**完全公开**的情况下，协商出一个**只有他俩知道**的密钥。这就是"公钥密码"的开端，也是第 2.4 节和 3.3 节要讲的内容。

第 1 章

从最朴素的想法开始

这一章要解决什么

这一章回答三个问题：

1. 我们到底要保护信息的什么？（不只是"别被看到"这么简单）
2. 一个加密系统由哪几块拼成？
3. 古人是怎么加密的？他们的办法为什么后来不管用了？

1.4 节的数学是全书地基——它看起来最枯燥，但后面每一章都在用它。建议在这一节多花时间。

1.1 信息要保护什么

生活场景

你去银行办事，走一遍流程：

- 进门，保安查身份证——确认你是不是你
- 进大厅后，你只能去普通柜台，进不了金库
- 你跟柜员说话，不希望旁边的人听见
- 你签的合同，事后不能被偷偷改动
- 你签完字，不能翻脸说"我没签"

这五件事，正好对应信息安全的五类安全服务。

服务	用一句话问	课件里的正式说法
认证（鉴别）服务	"你是谁？我怎么信你就是你？"	在网络交互过程中，对收发双方的身份及数据来源进行验证
访问控制服务	"我能干什么？"	防止未授权用户非法访问资源，包括用户身份认证和用户权限确认
数据保密性服务	"别人能否偷听到？"	防止数据在传输中被破解或泄露
数据完整性服务	"传输中被改过没有？"	防止数据在传输过程中被篡改
抗否认性服务	"我想赖账，怎么办？"	防止收发双方在操作后否认自己所做的操作

这五个问题，各自用什么技术解决？

大白话

课件的答案是：不同的服务，配不同的密码技术。

安全服务	用什么技术实现
保密性服务	加密
认证鉴别服务	密钥、数字签名
访问控制服务	密钥、权限
完整性服务	加密、数字摘要
抗否认性服务	数字签名

前后呼应

这张表是整门课的总纲。后面每一章，本质上都在实现其中某一项：

- 第 2 章讲"加密"和"数字摘要"具体怎么实现
- 第 3.5 节讲"认证鉴别"怎么实现
- 第 2.5 节讲"数字签名"（抗否认性 + 完整性）怎么实现

三个"如何实现"（课件用图说明的场景）

场景一：保密性

Alice 给 Bob 发消息，路上有黑客在偷听（被动攻击）。

解决：加密。黑客截到的是密文，看不懂。

场景二：完整性和鉴别

Alice 算出一个"指纹" $z = h_k(m)$ ，跟着消息 m 一起发。

Bob 收到后，自己也对 m 算一遍指纹 $y = h_k(m)$ 。

如果 $y \neq z$ ，说明 m 在路上被改过了。（因为改动一点点，指纹就会大变）

解决：密钥 + 哈希函数。

场景三：抗否认性

Alice 说"我没发过这条消息"，Bob 说"你发了"。谁对？

这就需要请一个仲裁者来判定。可仲裁者凭什么判？他需要一个只有 Alice 能生成、但谁都能验证的东西。

解决：数字签名。（注意这个巧妙的性质——"只有她能做，但谁都能验"，这正是数字签名的精髓。）

▼ 自测：请把右边的技术连到左边的服务上

服务	答案
保密性服务	加密
认证鉴别服务	密钥、数字签名
访问控制服务	密钥、权限
完整性服务	加密、数字摘要
抗否认性服务	数字签名

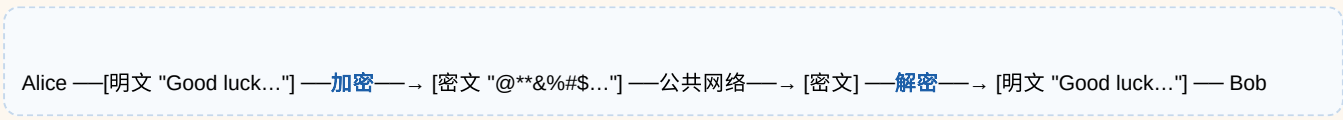
记忆窍门：
"保密靠加密"、"完整靠摘要"、"抗赖靠签名"——三个最容易混的先记住。剩下的"认证"和"访问控制"都需要密钥（因为要先认出身份，才能谈权限）。

1.2 一个密码系统由什么组成

五要素

生活场景

回到开头那个寄信的例子：



拆开看，一共出现了五样东西。

要素	是什么	例子
明文	要保护的原始消息	Good luck
密文	加密后看不懂的样子	@**&%#\$
加解密算法	明文 ↔ 密文的变换规则	"每个字母往后挪 N 位"
密钥	算法里的可调参数，保密的关键	N = 3
密钥空间	密钥所有可能取值的集合	N 可以是 1~25，共 25 个

大白话

算法是"菜谱"，密钥是"放多少盐"。

菜谱可以公开（大家都知道怎么做这道菜），但你放盐放多少，别人才不知道这道菜最后是什么味道——这就是密钥的作用。

密钥空间就是"盐可以放多少种量"。可选量越多，别人猜中的概率越低。

密码学发展的三个阶段

阶段	特征
1949 年以前	手工阶段， 密码学是一门艺术 。主要靠隐写术（隐形墨水、字符格式变化、图像）
1949~1975	机械阶段，现代密码出现。 1949 年香农提出"保密系统信息理论" ，提出：数据的安全基于 密钥 而不是算法
1976 年以后	计算机阶段， 公钥密码出现 。1976 年 Diffie & Hellman 提出不对称密钥密码；1977 年 RSA；90 年代 ECC

前后呼应

为什么 1949 年香农那句话是分水岭？

因为在它之前，密码学家花大量精力在"怎么把算法藏好"；在它之后，大家明白了——**算法该公开，只需要守住密钥**。整个现代密码学就建立在这一点上。

两种"安全"，别搞混

	理论安全（无条件安全）	实际安全（计算安全）
什么意思	攻击者无论截获多少密文，都得不到足够信息唯一决定明文	攻击者如果有 无限资源 ，任何系统都能破；但在 有限资源 内破不了
要什么条件	密钥长度 ≥ 明文长度，而且 密钥只用一次 （一次一密）	破它的代价 > 它能带来的收益
现实能不能做到	能，但 用不起来 （密钥太长、管理太麻烦，见 1.3 节 OTP）	现实中全部用的是这种

大白话

理论安全 = "数学上不可能打开"。

实际安全 = "理论上能打开，但打开它花的钱比里面东西还贵，所以没人去开"。

很反直觉吧？——**世界上没有绝对安全的密码系统，只有"划算不划算"**。

两种分类维度（重点，容易考）

维度一：算法公开不公开？

类型	算法	密钥
古典密码	保密	保密
现代密码	公开	保密

维度二：加密密钥公开不公开？

	对称密码	公钥密码
密钥	加密密钥和解密密钥相同（或能互相推出）	公钥和私钥不同，而且很难从公钥推出私钥
特点	快，但密钥管理复杂	密钥管理简单，但慢
代表	DES、AES、SM4、RC4	RSA、ECC、SM2、DH
用途	加密消息	协商密钥、数字签名
n 人互通的密钥数	$n(n-1)/2$	n 个私钥

大白话：用保险柜和邮箱来记

对称密码 = 保险柜。

你把文件放进保险柜、设密码锁上。取的时候要用**同一个密码**。

→ 所以密码必须保密；两个人用就得商量一个密码（人越多，要商量很多对密码）。

公钥密码 = 邮箱。

谁都能往邮箱里投信（公钥公开），但**只有本人有密码能开邮箱**（私钥保密）。

→ 所以密钥管理简单（每人只需管好自己的邮箱密码）。

为什么对称密码的密钥数那么吓人？

动手算一算

假设 5 个人（A、B、C、D、E）要两两保密通信。

每对人需要一个共享密钥：A-B、A-C、A-D、A-E、B-C、B-D、B-E、C-D、C-E、D-E。

数一数：**10 个**。公式是 $n(n-1)/2 = 5 \times 4 / 2 = 10$ 。

如果是 100 个人呢？ $100 \times 99 / 2 = 4950$ 个密钥。每个人要记 99 个密钥——这根本管不过来。

换成公钥密码：100 个人，每人只要管好自己的**1 个私钥**。答案就是“密钥管理简单”。

▼ 自测：为什么说公钥密码“密钥管理简单”？

因为公钥是公开的，不需要保密传输；只有私钥需要自己保管，一人一把。

对比一下：

- 对称密码：n 个人需要 $n(n-1)/2$ 个密钥，而且每个都要“秘密地”送到对方手里
- 公钥密码：n 个人需要 n 个私钥，公钥随便贴在网上都行

“送密钥”这件事本身就是个大麻烦（见 0.4 节说的死循环），公钥密码直接绕开了它——所以叫“密钥管理简单”。

1.3 古典密码：换内容还是换位置

这一节要解决什么

古人没有电脑，他们的加密办法很朴素。但**朴素不等于没用**——它们包含了密码学最基本的两种思路，而且把它们搞懂，你就理解了“什么叫密码分析”。

另外这一节还埋了一堆数学基础（约简、同余、逆元），是 1.4 节的地基。

第一步：所有古典密码只有两种花样

为什么需要它

你可能觉得古典密码五花八门：凯撒密码、斯巴达密码棒、莫斯密码.....但把它们摊开看，归根到底只有两种做法。

类型	做法	例子
代替密码	把明文里的 字符换成别的字符 （内容变了，位置没变）	凯撒密码、莫斯密码
换位密码	把明文里的 字符顺序打乱 （内容没变，位置变了）	斯巴达密码棒

大白话

拿一副扑克牌打比方：

- **代替** = 把**花色全换掉**（红桃→黑桃，黑桃→梅花...），但牌的顺序还是原来那样
- **换位** = 把牌洗乱，花色不变，但位置全变了

怎么一眼判断是哪种？

动手做：数字母

判断方法只有一句：**数一数密文和明文的字母，是不是同一批？**

- **是同一批**（字母没变，只是顺序乱了）→ **换位**
- **不是同一批**（字母被换过了）→ **代替**

练一练（别看答案，先自己判断）

明文 → 密文	是哪种？	为什么
hello → khood		
hello → loleh		
computer → retupmoc		
secret → vhfuhw		
attackatdawn → dawnattackat		

▼ 看答案

明文 → 密文	是哪种	为什么
hello → kloor	代替	h→k, e→h, l→o... 字母被换过了
hello → loleh	换位	还是 h,e,l,o 那 5 个字母，只是顺序乱了
computer → retupmoc	换位	字母完全相同，其实就是倒过来写（小心！看着陌生但字母没变）
secret → vhfuhw	代替	s→v, e→h, c→f... 每个都往后挪了 3 位
attackatdawn → dawnattackat	换位	同一批字母，把后 4 个挪到了前面

重点提醒：第 3 题最容易错。很多人看 `retupmoc` 觉得"这么陌生，字母肯定被换过了"——但"看着陌生"≠"字母变了"。

永远先数字母，别看观感。这一步只花 5 秒，能防住所有这类陷阱。

接下来：单表 vs 多表（这个区分很关键）

代替密码还能再分。区别在于：是一张表用到底，还是变着表用。

	单表代替	多表代替
做法	一张替换表，从头用到尾	每隔几个字母就换一张表
效果	同一个字母永远变成同一个密文字母	同一个字母在不同位置会变成不同的密文字母
代表	移位密码、仿射密码	Vigenère 密码、OTP

为什么这个区分重要

因为单表代替有个致命弱点：它保留了字母的"出现频率"。

比如英文里 `e` 出现得最多。那么在单表代替的密文里，"出现最多的那个字母"大概率就对应 `e`。攻击者靠这个就能破译——这叫**字母频率攻击**。

多表代替就是来治这个病的：同一个 `e` 在不同位置变成不同的字母，**频率被摊平了**，统计就没用了。

前后呼应

记住"频率"这个词。第 1.3 节的 Vigenère 部分，我们会讲它怎么用"重合指数"（0.038 / 0.067 / 0.047 那三个数）重新把多表代替攻破。

① 移位密码（凯撒密码）：最朴素的想法

生活场景

凯撒大帝要传军令，他想到的办法是：每个字母往后挪几位。

明文： i love china

每个字母挪 2 位 →

密文： k nqxc ejkpc

看着像乱码，但规律极其简单。

先做一件准备工作

要算数，得先把字母变成数字。英文字母 a~z 用 0~25 表示：

字母	a	b	c	d	e	...	z
数字	0	1	2	3	4	...	25

注意是从 0 开始，不是 1。为什么？因为后面要算“% 26”，结果必须落在 0~25 这 26 个数字里，正好对应 26 个字母。如果从 1 开始，算到 26 会变成 0，0 不在 1~26 里，就漏了。

公式

$$\text{加密: } y = (x + k) \% 26 \quad \text{解密: } x = (y - k) \% 26$$

其中 x 是明文字母的数字，y 是密文字母的数字，k 是密钥。

大白话

把字母的数字加上 k，如果超过 25 就“绕回来”。

所谓“绕回来”，是因为字母表是一个环——z 后面又接回 a：

a b c d e ... x y z → a b c ...

(数到 z 就绕回 a)

动手做：跟课件算一遍

课件给的例子是 $(24 + 3) \% 26 = 1$ 。拆开看：

1. 数字 24 对应字母 y
2. $24 + 3 = 27$ ，超过 25 了
3. $27 \% 26 = 1$ （因为 $27 - 26 = 1$ ）
4. 数字 1 对应字母 b

所以 y 加密后变成 b。用“绕回来”的说法验证：y 往后数 3 个 → z → a → b ✓

解密就反过来减：课件写 $(1 - 3) \% 26 = 24$ 。

1. 密文 1 对应 b
2. $1 - 3 = -2$ ，是负数
3. $-2 \% 26 = 24$ （因为 $-2 + 26 = 24$ ）
4. 数字 24 对应字母 y ✓ 还原了

小心坑：算出来是负数怎么办

课件这句 $(1-3) \% 26 = 24$ 里藏着一条全书通用的规则：

算出负数，就加模数变正。

为什么加 26 就行？因为 -2 和 24 相差正好 26，在“模 26”的世界里，这两个数是同一个：

$$-2 \equiv 24 \pmod{26}$$

课件有句原话："在密码学中：任何整数模 m 的约简都是非负数。"这个"约简"就是指这个动作。

这条规则你会在后面反复用到（比如求逆元、RSA 计算、欧拉定理降指数），一定要养成习惯。

移位密码的安全性问题

为什么需要它

移位密码的密钥 k 可以取 1~25，一共只有 25 种可能。

这意味着什么呢？攻击者最多试 25 次就能破译。拿张纸手算，一分钟都用不了。

这个"可选密钥的个数"就是密钥空间。课件点明了：

密钥空间决定了穷举攻击的难度。

所以课件紧接着问了一句：**如何增大密钥空间，提高安全性？**——这就引出了下一个算法。

▼ 自测：用密钥 $k=3$ 加密 cat，结果是什么？

fdw

一步步来：

- $c = 2, 2 + 3 = 5 \rightarrow f$
- $a = 0, 0 + 3 = 3 \rightarrow d$
- $t = 19, 19 + 3 = 22 \rightarrow w$

再解回来（密文 khooor，密钥 3）：

- $k = 10, 10 - 3 = 7 \rightarrow h$
- $h = 7, 7 - 3 = 4 \rightarrow e$
- $o = 14, 14 - 3 = 11 \rightarrow l$
- $o = 14, 14 - 3 = 11 \rightarrow l$
- $r = 17, 17 - 3 = 14 \rightarrow o$

得到 hello ✓

再练一个会出负数的：解 djx，密钥 5。

- $d = 3, 3 - 5 = -2 \rightarrow -2 + 26 = 24 \rightarrow y$
- $j = 9, 9 - 5 = 4 \rightarrow e$
- $x = 23, 23 - 5 = 18 \rightarrow s$

得到 yes ✓ —— 看到负数的处理了吧。

② 仿射密码：给密钥多加一个参数

为什么需要它

移位密码的问题：密钥空间只有 25，太容易被穷举。

仿射密码的思路非常直接：一个参数不够，我上两个参数。

	公式	参数个数
--	----	------

移位密码	$y = (x + k) \% 26$	1 个 (k)
仿射密码	$y = (a \cdot x + b) \% 26$	2 个 (a, b)

就是把 $x + k$ 升级成了 $a \cdot x + b$ ——先乘 a ，再加 b 。

(顺便：如果 $a = 1$ ，仿射密码就退化成了移位密码。)

公式

$$\text{加密: } y = E_{a,b}(x) = (a \cdot x + b) \% 26$$

$$\text{解密: } x = D_{a,b}(y) = (a^{-1} \cdot y - a^{-1} \cdot b) \% 26$$

关键要求： a 必须与 26 互素 (也就是 a^{-1} 必须存在)。

为什么 a 不能随便选? (这是重点)

先看解密公式需要什么

解密要用到 a^{-1} (a 的乘法逆元)。如果 a^{-1} 不存在，就解不回来。

课件里讲过一个规则：乘法逆存在的条件是 $(n, m) = 1$ ，也就是两数互素。

所以 a 必须与 26 互素。

动手做：看看 $a=2$ 会出什么事

假设 $a = 2$ ，加密规则 $y = (2x + b) \% 26$ 。逐个数试一遍：

明文 x	$2x$	$y = 2x \% 26$
0	0	0
13	26	0
1	2	2
14	28	2

看出来了吗? $x=0$ 和 $x=13$ 加密后都是 0!

这意味着：收到密文 0，你分不清原文是 0 还是 13。信息丢了，救不回来。

根子在哪? 因为 $\gcd(2, 26) = 2 \neq 1$ ，2 和 26 有公约数，所以“乘 2”这个操作不是一一对应的——多个输入挤到同一个输出去了。

26 个明文只产生了 13 种密文——一半的密文根本用不上。

大白话

好的加密必须“一对一”：每个明文有唯一的密文，每个密文也能唯一地还原成明文。

a 与 26 互素，才能保证“乘 a ”这个操作是漂亮的一一对应。

课件用的例子 $a = 9$ ， $\gcd(9, 26) = 1$ ✓ 合法。

仿射密码的密钥空间

动手算一算

a 有多少种选法? $26 = 2 \times 13$ ，所以和 26 互素的数就是“不是 2 的倍数、也不是 13 的倍数”的数。

在 1~25 里数一数：

1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25

(就是除了 13 之外的奇数) 共 12 个。

b 有多少种选法? b 可以是 0~25 任意一个, 共 26 个。

所以总共 $12 \times 26 = 312$ 种。但其中 $a=1, b=0$ 这一组是“什么都不做” ($y=x$), 等于没加密, 要去掉。

$$\text{密钥空间} = 12 \times 26 - 1 = 311$$

	密钥空间	穷举要试几次
移位密码	25	25 次
仿射密码	311	311 次

大了 12 倍多。但说实话——311 次也就几分钟的事, 还是不够安全。(后面 Vigenère 会把密钥空间真正拉起来。)

跟着课件算一遍

加密: 明文 meet, 密钥 $(a,b) = (9,3)$

加密式: $y = (9x + 3) \% 26$

逐个算:

- $m = 12, (9 \times 12 + 3) = 111, 111 \% 26 = 7 \rightarrow h$ (因为 $26 \times 4 = 104, 111 - 104 = 7$)

课件给的结果是 meet me at midnight $\rightarrow$ hnns hn ds hxeqxfos, 第一个字母确实是 h ✓

解密: 那两个数 (3 和 17) 哪来的?

课件说这个密钥的解密式是 $x = (3y + 17) \% 26$ 。这两个数不是拍脑袋来的, 而是:

$$\text{解密密钥} = (a^{-1}, -a^{-1} \cdot b)$$

第一步: 求 $a^{-1} = 9^{-1} \% 26$

用 Euclid 算法 (详细方法见 1.4 节, 这里先看结果):

$$26 - 2 \times 9 = 8$$

$$9 - 1 \times 8 = 1 \rightarrow 1 = 9 - 8 = 9 - (26 - 2 \times 9) = 3 \times 9 - 26$$

得到 $3 \times 9 - 1 \times 26 = 1$, 所以 $9^{-1} = 3$ 。验算: $9 \times 3 = 27 = 26 + 1 \equiv 1 \checkmark$

第二步: 算 $-a^{-1}b = -(3 \times 3) = -9$

是负数, 加 26 变正: $-9 + 26 = 17$

所以解密式是 $x = (3y + 17) \% 26$ 。

验算: 密文 $h = 7, (3 \times 7 + 17) = 38, 38 \% 26 = 12 \rightarrow m \checkmark$ 还原了

小心坑: 加密密钥 $\neq$ 解密密钥

用途	密钥
加密	$(a, b) = (9, 3)$
解密	$(a^{-1}, -a^{-1}b) = (3, 17)$

两组数完全不同。考试如果问"解密密钥"，填成加密的那组就错了。

(这个坑在 RSA 里也一样：加密用公钥 (e,n) ，签名用私钥 d 。)

仿射密码怎么被攻击的 (已知明文攻击)

如果攻击者知道了两组对应关系，就能反推密钥。课件给的公式：

$$a = (x - y)^{-1} \cdot (E(x) - E(y)) \% 26$$

$$b = (E(x) - a \cdot x) \% 26$$

思路其实很朴素：列两个方程，一相减， b 就被消掉了，只剩一个未知数 a 。

$$7a + b \equiv 1 \pmod{26}$$

$$12a + b \equiv 4 \pmod{26}$$

———— 两式相减 ————

$$5a \equiv 3 \pmod{26}$$

然后求 a ：两边乘 5^{-1} (不能"除以 5"，模运算没有除法)。这就是 1.4 节要练的手艺。

▼ 自测：已知 $E(7)=1$ 且 $E(12)=4$ ，求加密密钥 (a,b)

第一步：两式相减消掉 b

$$7a + b \equiv 1 \pmod{26} \quad \dots(1)$$

$$12a + b \equiv 4 \pmod{26} \quad \dots(2)$$

$$(2)-(1): 5a \equiv 3 \pmod{26}$$

第二步：求 $5^{-1} \% 26$

$$26 - 5 \times 5 = 1 \rightarrow 1 = 26 - 5 \times 5 = (-5) \times 5 + 1 \times 26$$

所以 $5^{-1} = -5 = 21$ 。验算： $5 \times 21 = 105 = 4 \times 26 + 1 \checkmark$

第三步：求 a

$a = 21 \times 3 = 63$ ， $63 \% 26 = 11$ ($63 - 52 = 11$)。检查 $\gcd(11, 26) = 1 \checkmark$ 合法。

第四步：回代求 b

$$b = (1 - 11 \times 7) \% 26 = (1 - 77) \% 26 = -76 \% 26 = -76 + 78 = 2。$$

答：加密密钥 $(a, b) = (11, 2)$

如果要解密密钥：再求 $a^{-1} = 11^{-1} \% 26 = 19$ (因 $11 \times 19 = 209 = 8 \times 26 + 1$)， $-a^{-1}b = -38 \equiv 14$ 。
所以解密密钥是 $(19, 14)$ ，即 $x = (19y + 14) \% 26$ 。

验算： $D(1) = 33 \% 26 = 7 \checkmark$ ； $D(4) = 90 \% 26 = 12 \checkmark$

③ Vigenère 密码：变着法儿换表

为什么需要它

回忆那个问题：单表代替保留了字母频率，所以能被统计攻破。

Vigenère 的思路是：别用一张表了，用一串密钥字母轮着换表。

先看个最小例子

明文字母 **cat**，密钥 **key** (三个字母)。

c 用密钥 **k** ($=10$) $\rightarrow 2 + 10 = 12 \rightarrow$ **m**

a 用密钥 **e** ($=4$) $\rightarrow 0 + 4 = 4 \rightarrow$ **e**

t 用密钥 **y** ($=24$) $\rightarrow 19 + 24 = 43 \% 26 = 17 \rightarrow$ **r**

密文：**mer**

看明白了吗：每个位置用的移位量都不一样 (10、4、24)，而不是统一的 3。

公式

密钥是序列 $k = (k_0, k_1, \dots, k_{m-1})$, m 是密钥长度。

$$\text{加密: } y_i = (x_i + k_{i \% m}) \% 26$$

$$\text{解密: } x_i = (y_i - k_{i \% m}) \% 26$$

密钥特征：加密密钥和解密密钥相同（因为只是加减法的关系）。

密钥空间： 26^m

那个 $i \% m$ 是什么意思

密钥不够长就循环使用。

密钥 **key** 长度是 3，但明文可能很长。那就把密钥一遍遍重复：

位置 i	0	1	2	3	4	5	6
用哪个	k	e	y	k	e	y	k

走到头就绕回开头，所以用 $i \% m$ 来取。

看它如何实现"多表代替"

动手做：明文 meetme，密钥 key

明文： m e e t m e

密钥： k e y k e y ← key 循环

明文数字： 12 4 4 19 12 4

密钥数字： 10 4 24 10 4 24

结果： 22 8 2 3 16 2

密文： w i c d q c

重点看那三个 e（位置 1、2、5）：

位置	明文	用的密钥字母	移位置	密文
1	e	e	4	i
2	e	y	24	c
5	e	y	24	c

同一个 **e**，在不同位置变成了不同的字母！这就是"多表代替"——频率被摊平了。

小心坑：为什么"同一个字母变成不同密文"？

很多人的第一反应是"因为密钥循环使用"。这个答案是错的，它说的是另一件事。

真正的原因是：密钥里不同位置的字母不一样。

- 位置 1 用密钥字母 **e** → 移 4 位 → **e** 变成 **i**
- 位置 2 用密钥字母 **y** → 移 24 位 → **e** 变成 **c**

用的密钥字母不同，移位量就不同，结果自然不同。

那“密钥循环”导致什么？正好相反——它导致“相同变成相同”：位置 2 和位置 5 相隔 3（正好是密钥长度），它们用的都是密钥字母 **y**，所以 **e** 两次都变成 **c**。

而这就是 **Vigenère** 的死穴，下一段就讲。

Vigenère 的死穴：周期性

问题出在哪

课件原话：

“有周期性的弱点：若两个字符出现的间隔是密钥长度的倍数，则它们将以同样的方法加密。”

翻译成人话：相隔正好 **m** 个位置的两处，用的是同一个密钥字母，加密方式完全一样。

如果这两处的明文也恰好相同，密文就相同——规律又出现了，统计攻击又活过来了。

大白话比喻

Vigenère 本来想“每个字母换一张表”，把频率摊平。但因为**密钥循环**，它实际上是“每 **m** 个字母，就把前面那套表**重复一遍**”。

就像一件迷彩服，花纹看着很乱，但你把图案平移 **m** 格，**它自己就重合了**。

攻击者只要找到这个 **m**，就能把“多表”拆成 **m** 个“单表”，逐个击破。

破译三步走

① 找密钥长度 **m** → ② 按 **m** 分组 → ③ 每组当移位密码破

为什么第 ③ 步可行？因为按 **m** 分组后，**每一组内部用的都是同一个密钥字母**——那它就是一个普通的移位密码，用频率攻击就能破。

怎么找 **m**? Kasiski 方法

核心观察：密文里如果有一段重复出现，那这两段之间的距离，很可能是密钥长度的倍数。

为什么？因为只有“距离是 **m** 的倍数”时，两处才“对齐”（用同一套密钥字母）。此时如果明文也相同，密文就相同。

用一个具体例子看。假设密钥 **abc** (**m=3**)：

位置	0	1	2	3	4	5	6	7	8
密钥字母	a	b	c	a	b	c	a	b	c

看，位置 0、3、6 用的都是 **a**（相差 3，是 **m** 的倍数）。

现在明文 **thecatthedog**，里面 **the** 出现在位置 0 和位置 6：

位置 0-2：明文 **the**，密钥 **abc** → 密文 **tig**

位置 6-8：明文 **the**，密钥 **abc** → 密文 **tig** ← 一模一样！

攻击者看到密文里 **tig** 出现了两次，距离是 6。于是他知道：**m 一定是 6 的约数**（2、3 或 6）。再找几处重复，取所有距离的公约数，就能锁定 $m = 3$ 。

小心坑

光"明文有重复"还不够，还要"距离对齐"。两个条件缺一不可。

如果 **the** 出现在位置 0 和位置 5（距离 5，不是 3 的倍数），那密钥字母不同（a 和 c），密文也不同，就不会产生重复。

另外：**Kasiski 需要文本足够长**。自然语言里重复片段很多（the、and、ing、tion...），文本一长就必然有重复。短文本用不了这招——那就换 Friedman 方法。

怎么找 m? Friedman 方法（换个思路）

Kasiski 靠"找具体片段"，Friedman 靠"看整体统计"。核心工具叫**重合指数**。

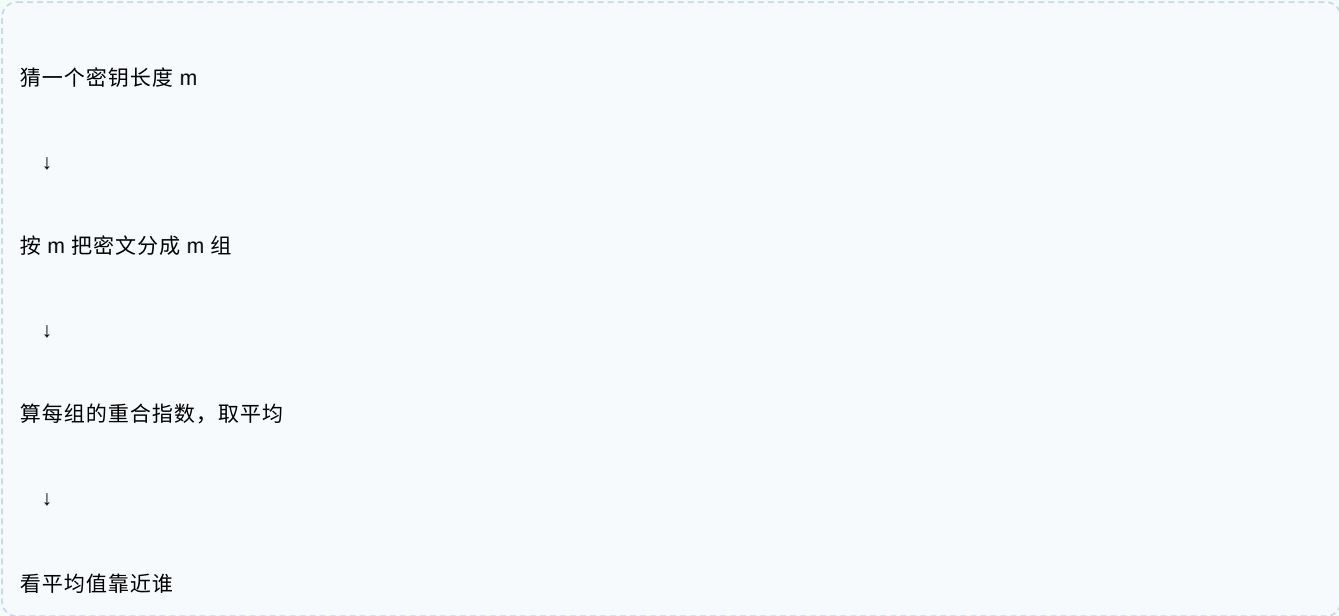
什么叫**重合指数**？就是一个概率问题：

从一段文字里随机抓两个字母，它们相同的概率是多少？

算法就是把 26 个字母各自的出现频率算出来，然后求"频率的平方和"。

文本类型	重合指数	为什么是这个值
完全随机的字符流	≈ 0.038	26 个字母概率都是 $1/26$ ，平方和 $= 26 \times (1/26)^2 = 1/26 \approx 0.038$
英文字符流	≈ 0.067	字母分布"扎堆"（e 最多，z、q 极少），所以撞上的概率更高
Vigenère 密文	≈ 0.047	被搅匀了一半，介于两者之间

怎么用它找 m



如果 **m 猜对了**：每组内部用的是同一个密钥字母 → 每组相当于一个**普通移位密码** → 而移位只是"整体平移"，**不改变字母分布的形状** → 所以每组还是"英文字符流" → 指数 ≈ 0.067 ✓

如果 **m 猜错了**：每组里混了不同密钥字母 → 等于被搅乱了 → 分布被摊平 → 指数下降，趋近 **0.038** ✗

结论：把不同的 m 都试一遍，哪个 m 让各组重合指数的平均值最接近 0.067，哪个就是正确答案。

前后呼应

为什么"移位不改变分布"这么关键？——这正是后面 6.3 节 CFB 模式和 4.2 节安全多方计算都会用到的思路：**平移/异或这类操作，只改变"位置"不改变"形状"**。

另外，课件还提到“多轮加密”：如果先用长度 m 的密钥加密，再用长度 n 的密钥加密，结果等于用长度 $\text{lcm}(m, n)$ 的密钥加密一次。如果 m 、 n 互素，就等于 $m \times n$ 。（所以多用几层不一定更安全，可能只是等价于一个更长的密钥。）

▼ 自测：为什么“猜对密钥长度后，每组重合指数会接近 0.067”？

因为猜对之后，每一组内部用的都是同一个密钥字母，所以每组就是一个普通移位密码；而移位只是给所有字母加同一个数，它只改变字母的“编号”，不改变字母出现的“相对频率”——所以每一组仍然是标准的英文字符流，重合指数自然回到 0.067。

反过来，猜错了就会把不同密钥字母混在一组，相当于把字母随机平移，分布被摊平，指数就掉向 0.038。

④ OTP 密码：理论上的“终极方案”

为什么需要它

Vigenère 被攻破的根子在于“密钥太短、会循环”。那如果密钥和明文一样长，而且只用一次呢？

那就没有任何周期可言了——这就是 OTP（One-Time Pad，一次一密）。

公式

加密解密和 Vigenère 完全一样，只是密钥的用法变了：

$$y_i = (x_i + k_i) \% 26 \quad x_i = (y_i - k_i) \% 26$$

课件原文：明文和密钥一样长，密钥本身只使用一次。

密钥空间： 26^n （ n 是明文长度）

大白话

注意那个密钥空间 26^n —— n 是明文长度。明文越长，密钥空间指数级增长。这个强度是压倒性的。

课件说它“克服了 Vigenère 密码易受重合指数攻击的缺点”，因为密钥不循环了，重合指数法就失效了。

它对应的是第 0 章讲的理论安全（无条件安全）——这是理论上唯一不可破的密码。

但它为什么用不了？

密钥要和明文一样长 → 传 1 万字的消息，就要 1 万字的密钥

密钥只能用一次 → 用完就扔，下次得重新生成、重新传递

密钥的产生、保存和传输成了天大的麻烦。

课件原话：“但密钥的产生、保存和传输非常重要。”

所以 OTP 是“理论上完美，实践上没法用”的典型代表。这也解释了第 0 章那句话：现实中所有系统用的都是“计算安全”，而不是“理论安全”。

▼ 自测：OTP 的密钥空间为什么是 26^n ，而不是 26^m ？

因为 OTP 的密钥长度不是固定的，而是和明文一样长 (n)。

对比一下：

- Vigenère：密钥是固定的 m 个字母，密钥空间 26^m ——**密钥可以循环用**
- OTP：密钥要和明文一样长 (n 个字母)，而且只能用一次，密钥空间 26^n

所以明文的长度直接决定密钥空间——这是它强度极高的原因，也是它没法实用的原因（消息越长，密钥越难搞）。

⑤ 换位密码：不动内容，只动位置

为什么需要它

前面讲的全是“代替”（换内容）。换位走的是另一条路：字母一个都不改，只把顺序打乱。

共性

- **原理**：将明文字符在消息内变换位置
- **标志**：明文和密文一样长（一个不多一个不少）
- **例子**：把文本翻转，`computersystems` → `SMETSYSRETUPMOC`

有意思的性质：课件说“唯密文攻击可能得到多种不同的破译结果；如 `live` – `evil` – `vile`”。

因为 `live`、`evil`、`vile` 都是同样 4 个字母的排列，攻击者即使知道用的是这 4 个字母，也分不清原文到底是哪个。

列换位：按行填格子，按列读出来

动手做：跟课件走一遍

明文 `canyouunderstand`（16 个字母）。

第 1 步：按 4 个字符一行排列

c a n y

o u u n

d e r s

t a n d

第 2 步：按列读出来

第1列：c o d t → "codt"

第2列：a u e a → "auea"

第3列：n u r n → "num"

第4列: y n s d → "ynsd"

密文: **codtaueanurnynsd** (即 CODTAUEANURNYNSD)

第3步: 解密就反过来 —— 把密文按4个一列排回去, 再按行读出, 就还原了。

一句话记住: 加密 = 按行填、按列读; 解密 = 按列填、按行读。

进阶版: 加个关键词决定"先读哪一列"

课件还展示了一个带关键词的版本, 用关键词 **type** 给每列贴标签:

列: 1 2 3 4

标签: t y p e

按标签的字母顺序排: $e < p < t < y$

所以读列的顺序是: 第4列(e) → 第3列(p) → 第1列(t) → 第2列(y)

ynsd + nurn + codt + auea = **YNSDNURNCODTAUEA**

这个关键词就是密钥。 不知道 **type** 这个词, 就不知道按什么顺序读列。

矩阵换位: 用"搬运表"指定每个位置搬到哪

先读懂那个矩阵

课件写的是:

$$f = (1\ 2\ 3\ 4 / \mathbf{2\ 4\ 1\ 3})$$

这个写法要这样读——上排是原位置, 下排是搬到哪:

上排 (原位置)	1	2	3	4
下排 (搬到哪)	2	4	1	3

逐列读: 第1位的字符搬到第2位; 第2位的搬到第4位; 第3位的搬到第1位; 第4位的搬到第3位。

就是一张"搬运指令表"。

动手做: 加密 cany 这一组

原位置	字符	搬到哪
1	c	$f(1) = 2$
2	a	$f(2) = 4$
3	n	$f(3) = 1$
4	y	$f(4) = 3$

现在看密文的四个位置:

第1位: n (原来在第3位)

第2位: c (原来在第1位)

第3位: y (原来在第4位)

第4位: a (原来在第2位)

→ 密文组: **ncya**

四组都这样处理: **cany|ouun|ders|tand** → **ncya|uonu|rdse|ntda**

密文: **NCYAUONURDSENTDA** ✓ 和课件一致

解密: 用"逆矩阵"搬回去

课件给的解密矩阵: $f^{-1} = (1\ 2\ 3\ 4 / 3\ 1\ 4\ 2)$

拿密文第一组 **n c y a** 试:

第1位 n → 搬回明文第 3 位

第2位 c → 搬回明文第 1 位

第3位 y → 搬回明文第 4 位

第4位 a → 搬回明文第 2 位

→ 明文: **cany** ✓ 还原了

为什么能还原? 课件写: $f \circ f^{-1} = (1\ 2\ 3\ 4 / 1\ 2\ 3\ 4)$ —— 最后这个矩阵上下排一模一样, 叫**恒等置换**: 谁都待在原位、没动。所以"搬过去再搬回来"就复原了。

前后呼应

这个"搬运表"其实就是 **1.4 节**要讲的**"置换"**。

矩阵换位里的东西	对应 1.4 节的概念
$(1\ 2\ 3\ 4 / 2\ 4\ 1\ 3)$	一个置换
$f \circ f^{-1}$	置换的复合
$(1\ 2\ 3\ 4 / 1\ 2\ 3\ 4)$	平凡置换 (单位元)
f^{-1}	逆置换

换位密码的本质, 就是用置换来加密。这也是为什么 1.4 节要专门讲"置换群"。

课件补充: CTF 里常见的几种编码

- **Base64**: 3 个 8 位字节转化为 4 个 6 位的字节, 之后在 6 位前面补两个 0, 形成 4 个 8 位字节。
- **猪圈密码** (Pigpen Cipher / 九宫格密码 / 共济会密码): 以格子为基础的简单替代式密码。
- **培根密码**: 明文字母被 5 个字符组成的序列替换。

- `jjencode` / `aaencode`: 转换成只有特定符号的字符串 / 常见的网络表情。

1.4 数学基础：这一章是全书的地基

先说实话

这一节最容易劝退人。原因是课本习惯“先给定义、再给公式”，而这些数学概念本身很抽象，一上来就顶不住。

但它的内容其实一点都不难，只是需要换个讲法。

我给这一节定的规矩是：每个概念都从一张图或一个具体数字开始，绝不先抛定义。

这一节为什么重要

第 2 章会直接把这些东西拿来用，而且不再解释。比如课件在讲 RC4 之后会说：

"线性同余发生器的周期为 a 在乘法群 Z_p^* 中的阶 l 。"

这一句话里藏了三个本节的概念：乘法群、 Z_p^* 、阶。

所以这一节看不懂，第 2 章就一定是天书。反过来，把这节的六个“零件”练熟，第 2 章就只是“换场景用零件”。

① 约简：一切从“转盘”开始

生活场景

你见过时钟吧？现在是 10 点，再过 5 个小时，是几点？

你会说“3 点”——而不是“15 点”。因为钟面上只有 12 个数字，走到 12 就绕回 1。

这个“绕回来”的动作，就是密码学里用得最多的运算：取余数（约简）。

为什么需要它

密码学要拿数字做运算，但结果不能无限大——得把它圈在一个固定范围里。

比如字母只有 26 个，那算出来的数字就必须落在 0~25 之内。所以每次算完都要“绕回来”。

定义

约简就是“除以模数取余数”：

$$r = n \% m \quad (\text{读作“}n \text{ 模 } m\text{”)}, \text{ 满足 } 0 \leq r < m$$

课件原话：**"在密码学中：任何整数模 m 的约简都是非负数。"**

动手做：负数怎么约简

求 -200 模 98 的约简。

方法：不断加 98，直到落进 0~97 的范围。

-200

$-200 + 98 = -102$ （还是负数，继续）

$-102 + 98 = -4$ （还是负的，继续）

$-4 + 98 = 94$ （落在 0~97 里了，停）

验算： $94 - (-200) = 294 = 3 \times 98$ ，差是 98 的整数倍 ✓

换个说法理解：把模 98 想成一个 98 格的转盘。从 -200 出发往正方向走，走三步（每次 +98）就上了转盘。

小心坑

有人算成 $-200 \div 98$ 余 -4 ，然后就把 -4 当答案了。不行——结果必须是非负的。

记一个万能公式：约简 = $((n \% m) + m) \% m$ 。这样负数也能一步到位。

这条“负数加模数变正”的规则，会在全书反复出现（求逆元、RSA、欧拉定理降指数……）。养成习惯，看到负数就条件反射地加上模数。

▼ 自测：求 $7 - 22$ 模 9 的约简

答：3

$$7 - 22 = -15$$

$$-15 + 9 = -6 \quad (\text{还是负，继续})$$

$$-6 + 9 = 3 \quad (\text{落在 } 0 \sim 8 \text{ 里})$$

或者一步到位： $-15 + 18 = 3$ ($18 = 2 \times 9$)。验算： $3 - (-15) = 18 = 2 \times 9$ ✓

② 同余：哪些数算“一伙的”

生活场景

时钟上，“15 点”和“3 点”是同一个时刻。

星期几也一样：“今天星期三”和“8 天后”也是星期三（因为 $8 = 1 \text{ 周} + 1 \text{ 天}$ ）。

这种“相差整数个周期，就当作同一个”的关系，叫同余。

定义

$$x \equiv y \pmod{m} \quad \text{当且仅当} \quad m \mid (x - y)$$

读作“x 模 m 同余于 y”。意思是：x 和 y 的差，能被 m 整除。

例子： $27 \equiv 6 \pmod{7}$ ，因为 $27 - 6 = 21 = 3 \times 7$ ✓

大白话

同余 = “在模 m 的世界里是同一个数”。

在模 7 的世界里，27 和 6 是同一个——因为它们都“落在转盘的同一格”（ $27 \% 7 = 6$ ， $6 \% 7 = 6$ ）。

同余的运算法则（这几条很实用）

最常用的两条

① 加法和乘法可以直接对每个数取模

$$(x + y) \% m = ((x \% m) + (y \% m)) \% m$$

所以你算 $(24 + 3) \% 26$ 的时候，不用管 27 有多大，直接算就完了。

② 想"约掉"一个因数，必须先验证它与模数互素

若 $a \cdot x \equiv a \cdot y \pmod{m}$ 且 $(a, m) = 1$ ，则 $x \equiv y \pmod{m}$

这条特别重要。举个反例看看不注意会怎样：

$15 \equiv 9 \pmod{6}$ ，两边都"约掉 3"

→ $5 \equiv 3 \pmod{6}$?

→ $5 - 3 = 2$ ，不能被 6 整除，**错了!**

问题在于 $\gcd(3, 6) = 3 \neq 1$ ，**3 和模数 6 不互素**，所以不能约。

小心坑：模运算里没有"除法"

这是初学者最容易栽的地方。 $a \cdot x \equiv b \pmod{m}$ 这种式子，两边不能"除以 a"。

正确做法是：两边乘以 a 的乘法逆元 a^{-1} 。

比如 $5a \equiv 3 \pmod{26}$ ，不能写成 $a = 3/5$ ，而要写成 $a \equiv 5^{-1} \times 3 \pmod{26}$ 。

至于怎么求逆元——就是下面第 ④ 条要讲的手艺。

▼ 自测：判断对错 —— " $23 \equiv 5 \pmod{9}$ ，两边都约掉 3，得 $23 \equiv 5 \pmod{3}$ "

对。这属于另一条规则： $x \equiv y \pmod{m}$ ，如果 $d \mid m$ ，则 $x \equiv y \pmod{d}$ 。

这里 $m = 9$ ， $d = 3$ ，而 3 确实整除 9，所以可以。

注意区别：

- 要改模数（比如从 $\pmod{9}$ 变成 $\pmod{3}$ ）→ 只要 d 整除原模数就行
- 要约掉因数（比如把 $15 \equiv 9$ 变成 $5 \equiv 3$ ）→ 必须这个因数与原模数互素

两条规则别混。做题时先问自己："我是想改模数，还是想约因数？"

③ 素数：为什么密码学这么看重它

核心事实：算术基本定理

每一个正整数都可以分解为素数的乘积，而且这种分解是唯一的。

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_m^{e_m}$$

例子： $3960 = 2^3 \times 3^2 \times 5 \times 11$ ； $400 = 2^4 \times 5^2$

为什么需要它

注意加粗的那句："而且这种分解是唯一的"。

这句话看着平淡，但它是 **RSA 全部安全性的来源**：

取两个大素数 p 和 q，乘起来得到 $n = p \times q$

(这一步很快，小学生都会乘法)

反过来：给你 n ，让你还原出 p 和 q

(这一步——难如登天)

这就是“**因子分解难题**”：正着算容易，反着算几乎不可能。RSA 就架在这个不对称上。

大白话

把两袋沙子混在一起很容易，把混好的沙子分开就难了。

素数就是密码学里那袋“分不开的沙子”。而且数字越大，越分不开——所以 RSA 要求 n 至少 1024 比特（约 309 位十进制数）。

课件补充的小知识

- **素数有无穷多个**：设 $p_1 \dots p_n$ 为所有素数，则 $q = p_1 p_2 \dots p_n + 1$ 必然包含一个“新”的素因子。
- **越大越稀**：[1,100] 有 25 个；[1000,1100] 有 16 个；[100000,100100] 有 6 个；[10000000,10000100] 只有 2 个。
- 中国古代数学家提出“若 $n \mid 2^n - 2$ ，则 n 为素数”——但 **$n = 341$ 时不成立**。这类“骗过检验的合数”就叫**伪素数**，专门骗检验的还有一类更狠的，叫**卡迈克尔数**（如 $561 = 3 \times 11 \times 17$ ）。

④ Euclid 算法与乘法逆元（重点中的重点）

为什么需要它

回忆前面遇到的两个麻烦：

- 仿射密码解密要用 a^{-1} (a 的逆元)
- 同余式 $5a \equiv 3 \pmod{26}$ 要求 a ，得两边乘 5^{-1}

求逆元是密码学最常用的手艺——RSA 算私钥、欧拉定理降指数、坏种子公式，全都要用。而 Euclid 算法就是求逆元的标准工具。

逆元的定义

m 模 n 的乘法逆 t 满足：

$$m \times t \% n = 1$$

存在条件： $(n, m) = 1$ ，也就是 m 和 n 互素。**不互素就没有逆元。**

课件还给了一个更好用的说法：

满足 $a \cdot x + b \cdot n = 1$ 的整数 a ，就是 x 模 n 的乘法逆元。

大白话

求逆元，就是“**找一个数，让它和原数相乘后，除以 n 余 1**”。

课件那句“满足 $ax + bn = 1$ 的 a 就是逆元”翻译过来是：

只要你能凑出“ x 的多少倍 加上 n 的多少倍 等于 1”这个式子，那个“ x 的多少倍”里 x 前面的系数，就是答案。

Euclid 算法就是“凑这个式子”的工具。

动手做：Euclid 算法的两个阶段

算法分两步走，写在纸上**左右两列**：

阶段	干什么	写在哪一列
① 辗转相除	大数除以小数、取余数，反复除到余数是 1	左列
② 回代	从"1 = ..."倒着往回想，把 1 拆成原来两个数的组合	右列

完整示范：求 $23^{-1} \% 100$ （课件原例）

左列：辗转相除

$$100 - 4 \times 23 = 8 \quad (100 \text{ 减去 } 4 \text{ 个 } 23, \text{ 剩 } 8)$$
$$23 - 2 \times 8 = 7 \quad (23 \text{ 减去 } 2 \text{ 个 } 8, \text{ 剩 } 7)$$
$$8 - 1 \times 7 = 1 \quad (8 \text{ 减去 } 1 \text{ 个 } 7, \text{ 剩 } 1)$$
$$7 - 7 \times 1 = 0 \quad (\text{余数 } 0, \text{ 停})$$

右列：回代——从"1 = 8 - 1×7"这一行出发，一路往上代：

$$1 = 8 - 1 \times 7$$
$$= 8 - 1 \times (23 - 2 \times 8) \quad \leftarrow \text{把 } 7 \text{ 换成它的来源}$$
$$= 3 \times 8 - 1 \times 23$$
$$= 3 \times (100 - 4 \times 23) - 1 \times 23 \quad \leftarrow \text{把 } 8 \text{ 换成它的来源}$$
$$= 3 \times 100 - 13 \times 23$$

读答案：式子 $3 \times 100 + (-13) \times 23 = 1$ 里，23 的系数是 **-13**。

所以 $23^{-1} = -13$ 。是负数，加 100 变正： $-13 + 100 = \mathbf{87}$ 。

验算： $23 \times 87 = 2001 = 20 \times 100 + 1 \checkmark$

再示范一个短的：求 $5^{-1} \% 26$

左列： $26 \div 5 = 5$ 余 1，一步就到 1！

$$26 - 5 \times 5 = 1$$
$$5 - 5 \times 1 = 0$$

右列：因为只除了一次，不用反复回代，直接把 1 整理成目标形式：

$$1 = 26 - 5 \times 5 = (-5) \times 5 + 1 \times 26$$

5 的系数是 -5 ，所以 $5^{-1} = -5 = 21$ （加 26 变正）。

$$\text{验算：} 5 \times 21 = 105 = 4 \times 26 + 1 \checkmark$$

难度取决于什么

对比上面两个例子： $23^{-1} \% 100$ 除了 4 步，所以要回代好几次； $5^{-1} \% 26$ 只除了 1 步，就不用回代。

所以手算时，如果发现“很快除到 1”，就说明这题很轻松。

小心坑（三条）

1. **必须先验证互素。** 算之前先看 $\gcd(m, n)$ 是不是 1。不是就直接写“无解”，别白费力气。
2. **不能只算到 gcd 就停。** 辗转相除本身只给最大公因子，回代才给逆元。
3. **答案是负数要变正。** 像上面 $23^{-1} = -13$ 要写成 87。

▼ 自测：求 $101^{-1} \% 234$

第一步：验互素 + 辗转相除

$$234 - 2 \times 101 = 32$$

$$101 - 3 \times 32 = 5$$

$$32 - 6 \times 5 = 2$$

$$5 - 2 \times 2 = 1 \quad \leftarrow \text{到 1 了}$$

$$2 - 2 \times 1 = 0$$

最后非零余数是 1，所以 $\gcd(101, 234) = 1 \checkmark$ 逆元存在。

第二步：回代

$$1 = 5 - 2 \times 2$$

$$= 5 - 2 \times (32 - 6 \times 5) = 13 \times 5 - 2 \times 32$$

$$= 13 \times (101 - 3 \times 32) - 2 \times 32 = 13 \times 101 - 41 \times 32$$

$$= 13 \times 101 - 41 \times (234 - 2 \times 101) = 95 \times 101 - 41 \times 234$$

第三步：读答案

101 的系数是 **95**，所以 $101^{-1} \% 234 = 95$ 。

$$\text{验算：} 101 \times 95 = 9595 = 41 \times 234 + 1 \checkmark$$

⑤ 群：一个"能自由做运算"的世界

生活场景

想象一个 6 格的转盘，格子上标着 0~5：



规则是："每次走几步，走过头就绕回 0"。

试几个：从 4 走 3 步 $\rightarrow 5 \rightarrow 0 \rightarrow 1$ ，答案是 1（因为 $4+3=7$ ， $7-6=1$ ）。

这个"6 格转盘的世界"，就叫做 Z_6 ，成员是 $\{0,1,2,3,4,5\}$ ，运算是"模 6 加法"。

为什么需要它

因为密码学要反复做模运算，但不是随便什么集合配随便什么运算都能"讲道理"。

"群"就是规定了：一个集合上的运算，要满足哪几条规矩，才算是一个能自由使用的世界。

群的 4 条规矩

拿 Z_6 的加法逐条检查：

- 1. 封闭性：任意两个成员相加，结果还在这个集合里。
 $\rightarrow 4+5=9\equiv 3$ ，还在 0~5 里 $\checkmark$ （转盘永远跑不出去）
- 2. 结合律： $(a+b)+c=a+(b+c)$ 。
 $\rightarrow$ 用转盘想：先走 $a+b$ 步再走 c 步，和先走 a 步再走 $b+c$ 步，走到的地方当然一样 $\checkmark$
- 3. 有单位元：有一个"什么都不做"的元素。
 $\rightarrow$ 加法里它是 0： $a+0=a$ $\checkmark$
- 4. 每个元素有逆元：每个元素都有一个"能抵消它"的伙伴。
 $\rightarrow$ 见下表 $\checkmark$

四条全过，所以 $\langle Z_6, + \rangle$ 是个群。再加一条交换律 $(a+b=b+a)$ ，就是个交换群。

动手做：找 Z_6 里每个元素的逆元

"逆元"的意思：找一个数，和它相加等于单位元（也就是 0）。

元素	它的逆元	验证
0	0	$0+0=0$ $\checkmark$
1	5	$1+5=6\equiv 0$ $\checkmark$
2	4	$2+4=6\equiv 0$ $\checkmark$
3	3	$3+3=6\equiv 0$ $\checkmark$ （自己抵消自己）
4	2	$4+2=6\equiv 0$ $\checkmark$
5	1	$5+1=6\equiv 0$ $\checkmark$

六个元素全都有逆元 $\checkmark$

那乘法呢？—— Z_6^*

课件还给了一个乘法群： $\langle Z_6^*, \times \rangle$ ，其中 $Z_6^* = \{1, 5\}$ 。

注意：这个集合只有 2 个元素，不是 6 个。

为什么只有 1 和 5？因为只有“和 6 互素的数”才有乘法逆元。

元素	与 6 的最大公因子	能进 Z_6^* 吗
0	6	✗
1	1	✓
2	2	✗
3	3	✗
4	2	✗
5	1	✓

检查它们的乘法逆元： $1 \times 1 = 1 \equiv 1 \checkmark$ ； $5 \times 5 = 25 = 24 + 1 \equiv 1 \checkmark$ 单位元是 1。

课件给的反例，很说明问题

课件说： $Z^* = Z \setminus \{0\}$ 不是群。

为什么？“去掉 0 的所有整数”配上普通乘法：

- 有单位元 1 ✓
- 但 2 的逆元是 $1/2$ —— 不在整数里！ ✗

所以它不满足“每个元素都有逆元”。

这个反例正好说明“互素”和“逆元”的关系——这也解释了你前面求逆元时，为什么必须先验证 $\gcd = 1$ 。

▼ 自测：在 Z_6 （模 6 加法）里，元素 2 的逆元是几？为什么？

答：4。因为 $2 + 4 = 6 \equiv 0$ （单位元）。

“逆元”的定义就是“和它运算后得到单位元”。加法里单位元是 0，所以找 $2 + ? \equiv 0 \pmod{6}$ ，答案就是 4。

顺便：这个逆元在密码学里叫“加法逆元”，和我们前面讲的“乘法逆元”（ $ax \equiv 1$ ）是两回事，注意区分。

⑥ 剩余系：把整数“分箱子”

最重要的一个比喻

桌上有 6 个箱子，标着号码：

[0] [1] [2] [3] [4] [5]

现在把每个整数按“除以 6 的余数”丢进对应的箱子。

动手做：看看每个箱子里装了什么

箱子 0：... -12, -6, 0, 6, 12, 18, ...

箱子 1: ... -11, -5, **1**, 7, 13, 19, ...

箱子 2: ... -10, -4, **2**, 8, 14, 20, ...

箱子 3: ... -9, -3, **3**, 9, 15, 21, ...

箱子 4: ... -8, -2, **4**, 10, 16, 22, ...

箱子 5: ... -7, -1, **5**, 11, 17, 23, ...

每个箱子里的数，叫做一个**剩余类**（"剩余"就是"余数"的意思——按余数分出来的类）。

同一个箱子里的数，在模 6 下是"同一个东西"： $0 \equiv 6 \equiv 12 \equiv -6 \pmod{6}$ 。

课件给了正式定义：模 m 的剩余类有 m 个（这里是 6 个）。

完全剩余系：每个箱子挑一个代表

动手做

每个箱子里有无穷多个数，太啰嗦。那就**每个箱子只挑一个代表**。

比如都挑"最小的非负数"：

箱子 0 $\rightarrow$ 0 箱子 1 $\rightarrow$ 1 箱子 2 $\rightarrow$ 2

箱子 3 $\rightarrow$ 3 箱子 4 $\rightarrow$ 4 箱子 5 $\rightarrow$ 5

$\rightarrow \{0, 1, 2, 3, 4, 5\}$

这就是一个"完全剩余系"。"完全"的意思是：6 个箱子全都有代表，一个都没落下。

换个挑法也行。比如都挑"最小的正数"：

箱子 0 $\rightarrow$ **6**（因为 0 不是正数） 箱子 1 $\rightarrow$ 1 箱子 2 $\rightarrow$ 2 ...

$\rightarrow \{6, 1, 2, 3, 4, 5\}$

这也是完全剩余系——只要"每个箱子恰好一个"，挑谁都行。

课件给的 5 种完全剩余系（以 $m=10$ 为例）

名称	内容
最小非负完全剩余系	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
最小正完全剩余系	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
最大非正完全剩余系	-9, -8, -7, -6, -5, -4, -3, -2, -1, 0
最大负完全剩余系	-10, -9, -8, -7, -6, -5, -4, -3, -2, -1
绝对值最小完全剩余系	-5 ... 0 ... 4 或 -4 ... 0 ... 5

注意：模 10 时"最小正"是 1~10（不是 0~9），因为 10 和 0 在同一箱，挑"最小正数"时会挑到 10。

课件定理 1 说：“m 个整数是模 m 的完全剩余系 $\iff$ 它们两两不同余。”翻译过来就是——只要 m 个数来自不同箱子，就必然每箱一个。

简化剩余系：只挑“互素”的箱子

为什么需要它

回忆上一节： Z_6 有 6 个箱子，但只有 {1, 5} 两箱能做乘法（有逆元）。
那我们把“能安全做乘除的那部分箱子”单独拎出来——这就是**简化剩余系**。

定义

Z_m^* （简化剩余系）= Z_m 中与 m 互素的那些元素。
课件例子： $Z_{10}^* = \{1, 3, 7, 9\}$ ； $Z_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

动手做：\$Z_8\$ 里有几个箱子？

8 个箱子，编号 0~7，逐个查“和 8 互素吗”：

箱子编号	0	1	2	3	4	5	6	7
与 8 互素？	×	✓	×	✓	×	✓	×	✓

$Z_8^* = \{1, 3, 5, 7\}$ ，共 4 个。
发现了吗——1、3、5、7 正好是 1~8 里的**奇数**。因为 8 的因子只有 2，所以“不是偶数”就跟它互素。

Z_n 和 Z_n^* 的区别（最容易混）

	Z_n	Z_n^*
含义	完全剩余系	简化剩余系
包含	所有 n 个剩余类	只含与 n 互素的
含 0 吗	含	不含
元素个数	n	$\varphi(n)$
配加法	是群 ✓	—
配乘法	不是群 ×	是群 ✓
用途	加法运算	乘除运算（RSA、DH 都靠它）

那个星号的作用：把“能做乘法逆元的那部分”筛出来，让乘法也构成群——也就是**能做除法**。

▼ 自测：模 10 有几个箱子？其中哪些和 10 互素？ Z_{10}^* 是哪些？

模 10 有 10 个箱子（0~9）。
10 = 2 × 5，所以“和 10 互素”就是“不是 2 的倍数、也不是 5 的倍数”：

1 ✓ 2 × 3 ✓ 4 × 5 × 6 × 7 ✓ 8 × 9 ✓

 $Z_{10}^* = \{1, 3, 7, 9\}$ ，共 4 个。（课件里也是这个例子）
验算一下它们都有乘法逆元：1×1 = 1 ✓；3×7 = 21 ≡ 1 ✓；9×9 = 81 ≡ 1 ✓ —— 7 和 3 互为逆元，9 自己和自己互为逆元。

⑦ 欧拉函数：就是"数个数"

为什么需要它

上一节说 Z_n^* 的元素个数是 $\varphi(n)$ 。那 $\varphi(n)$ 怎么算？

如果 n 很大（比如 10000），难道要一个个数吗？不用，有公式。

定义与公式

欧拉函数 $\varphi(n)$ = 1 到 n 里，与 n 互素的数的个数。也就是 Z_n^* 的元素个数。

规律一（素数）： p 为素数时， $\varphi(p) = p - 1$

规律二（素数的幂）： $\varphi(p^e) = (p - 1) \cdot p^{e-1}$

规律三（拼起来）：若 $(m, n) = 1$ ，则 $\varphi(mn) = \varphi(m) \cdot \varphi(n)$

先验证前两条规律

$\varphi(7) = ?$ （7 是素数）

1~7 里与 7 互素的：1,2,3,4,5,6 —— 全都可以，因为 7 是素数，1~6 里没有 7 的因子。

所以 $\varphi(7) = 6 = 7 - 1 \checkmark$

$\varphi(8) = ?$ （ $8 = 2^3$ ）

1~8 里与 8 不互素的是偶数：2, 4, 6, 8，共 4 个。

所以 $\varphi(8) = 8 - 4 = 4 \checkmark$ 和上一节数出来的一致。

用公式验证： $\varphi(2^3) = (2-1) \times 2^{3-1} = 1 \times 4 = 4 \checkmark$

这条公式为什么长这样？把 $8 - 4$ 改写一下：

$$8 - 4 = 8 \times (1/2) = 2^3 - 2^2 = 2^2 \times (2 - 1) = (p-1) \cdot p^{e-1}$$

就是"总数里减去一半（能被 p 整除的那些）"。

课件例题： $\varphi(10000)$ 怎么算

直接数？不可能。用三步走：

第一步：分解质因数 $10000 = 2^4 \times 5^4$

第二步：每项单独套公式

$$\varphi(2^4) = (2-1) \times 2^3 = 1 \times 8 = 8$$

$$\varphi(5^4) = (5-1) \times 5^3 = 4 \times 125 = 500$$

第三步：乘起来

$$\varphi(10000) = 8 \times 500 = 4000$$

（课件写的是 $8 \times 4 \times 125 = 4000$ ，结果一样）

小心坑

p^{e-1} 别漏掉。比如 $\varphi(8) = (2-1) \times 2^2 = 4$ ，不是 1； $\varphi(9) = (3-1) \times 3^1 = 6$ ，不是 2。

只有一次幂（ $e = 1$ ）时， $p^{e-1} = p^0 = 1$ ，才“自动消失”——这就是素数规律 $\varphi(p) = (p-1) \times 1 = p-1$ 的来历。

▼ 自测：算 $\varphi(18)$ 、 $\varphi(155)$ 、 $\varphi(9)$

$\varphi(18)$: $18 = 2 \times 3^2$

$$\varphi(18) = (2-1) \times 2^0 \times (3-1) \times 3^1 = 1 \times 1 \times 2 \times 3 = 6$$

验算：1~18 里与 18 互素的是 1,5,7,11,13,17，正好 6 个 ✓

$\varphi(155)$: $155 = 5 \times 31$

$$\varphi(155) = (5-1) \times 5^0 \times (31-1) \times 31^0 = 4 \times 1 \times 30 = 120$$

$\varphi(9)$: $9 = 3^2$

$$\varphi(9) = (3-1) \times 3^1 = 2 \times 3 = 6$$

验算：1~9 里与 9 互素的是 1,2,4,5,7,8，正好 6 个 ✓

一条规律：分解时哪一种幂次都要单独套公式，不能漏。（18 里 2 是一次幂、3 是二次幂，两种都出现了。）

⑧ 欧拉定理：一直转圈，终会回到起点

先看一个现象

取 $n = 9$ ， $x = 2$ 。我们把 2 的幂一次次算出来，每次模 9：

算到	2^1	2^2	2^3	2^4	2^5	2^6	2^7
值	2	4	8	16	32	64	128
模 9 后	2	4	8	7	5	1	2

串起来看：2 → 4 → 8 → 7 → 5 → 1 → 2 → 4 → ...

走了 6 步，回到 1！

发现一件很妙的事

刚才转圈经过的数字是：2, 4, 8, 7, 5, 1

排个序：1, 2, 4, 5, 7, 8

而 $Z_9^* = \{1, 2, 4, 5, 7, 8\}$

一模一样！

为什么？因为 $x = 2$ 与 9 互素，所以“每乘一次 2”结果还在 Z_9^* 里，永远不会跑到外面去。

（反例：如果取 $x = 3$ ，因为 $\gcd(3,9) = 3 \neq 1$ ，那么 $3 \times 3 = 9 \equiv 0$ ，跑到 0 去了，出圈了，永远回不来。）

欧拉定理

既然你一直待在 Z_n^* 里转，而 Z_n^* 一共只有 $\varphi(n)$ 个元素.....

那转 $\varphi(n)$ 步，一定回到 1。

写成公式：

$$x^{\varphi(n)} \equiv 1 \pmod{n} \quad (\text{前提: } (x, n) = 1)$$

费马小定理是它的特例：当 n 是素数时， $\varphi(n) = n-1$ ，于是

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

它的用途：降指数

问题：算 $3^{45} \bmod 10$ 。直接算？ 3^{45} 是个 22 位数。

用欧拉定理：

- ① 检查： $\gcd(3, 10) = 1 \checkmark$
- ② 算 $\varphi(10) = 4$ （因为 $Z_{10}^* = \{1, 3, 7, 9\}$ ）
- ③ 所以 $3^4 \equiv 1 \pmod{10}$ ，也就是“指数每加 4 就转一圈”
- ④ 把指数对 4 取模： $45 \div 4 = 11$ 余 1，所以 $45 \equiv 1 \pmod{4}$
- ⑤ 于是 $3^{45} = 3^{(4 \times 11 + 1)} = (3^4)^{11} \times 3^1 \equiv 1^{11} \times 3 = 3$

一个 45 次方的题，变成了算 3^1 。

小心坑

- 先看模数是不是素数。是素数才能直接用费马小定理（用 $p-1$ 当周期）。
- 底数先约简。比如算 $249^{197} \bmod 199$ ，先把 249 约简成 50，数字小很多。
- 指数取模后余数为 0 时，结果是 1（因为 $x^{\varphi(n)} \equiv 1$ ）。比如 $3^{48} \bmod 10$ ： $48 \bmod 4 = 0$ ，答案是 1。
- “指数差 1”是个信号。比如 $249^{197} \bmod 199$ 里， $197 = 198 - 1$ ，这种题往往要变成“求逆元”。

▼ 自测：用欧拉定理算 $3^{200} \bmod 7$

① $\gcd(3, 7) = 1 \checkmark$

② 7 是素数， $\varphi(7) = 6$

③ 所以 $3^6 \equiv 1 \pmod{7}$

④ $200 \div 6 = 33 \text{ 余 } 2$ ，所以 $200 \equiv 2 \pmod{6}$

⑤ $3^{200} = (3^6)^{33} \times 3^2 \equiv 1 \times 3^2 = 9$

⑥ $9 \% 7 = 2$

答：2

- 对比另外几个：
- $3^{45} \bmod 10$: $\varphi(10) = 4$, $45 \bmod 4 = 1 \rightarrow 3^1 = 3$
 - $2^{100} \bmod 13$: $\varphi(13) = 12$, $100 \bmod 12 = 4 \rightarrow 2^4 = 16 \equiv 3$
 - $249^{197} \bmod 199$: $\varphi(199) = 198$, $197 = 198 - 1$ ，所以 $\equiv 249^{-1}$; $249 \equiv 50$, $50^{-1} = 4 \rightarrow 4$

⑨ 阶与原根：更精确的"周期"

为什么需要它

欧拉定理说"转 $\varphi(n)$ 步一定回到 1"。但实际上——可能早就回到了。

比如刚才 $x=2, n=9$ ，是走了 6 步才回到 1（正好是 $\varphi(9)$ ）。但换个数可能更快。这个"实际走了几步"就叫阶。

定义

x 的阶：使 $x^t \equiv 1 \pmod{n}$ 成立的**最小正整数 t**，记作 $\text{ord}_n(x)$ 。

大白话：自乘多少次第一次回到 1。

动手做：求 $n = 9$ 时每个元素的阶

x	自乘过程	阶（走了几步）
1	1	1
2	2 → 4 → 8 → 7 → 5 → 1	6
4	4 → 7 → 1	3
5	5 → 7 → 8 → 4 → 2 → 1	6
7	7 → 4 → 1	3
8	8 → 1	2

而 $\varphi(9) = 6$ 。

看那一列阶：1, 6, 3, 6, 3, 2 —— 全都是 6 的约数！

(1|6 $\checkmark$, 2|6 $\checkmark$, 3|6 $\checkmark$, 6|6 $\checkmark$)

核心性质

阶一定整除 $\varphi(n)$ 。

课件原话："若 g 是模 n 的原根， $\text{ord}_n(g) = \varphi(n)$ ；若 g **不是** 模 n 原根，则 $\text{ord}_n(g) \mid \varphi(n)$ 。"

这也就顺便解释了欧拉定理为什么成立：既然每个元素的阶都整除 $\varphi(n)$ ，那"走 $\varphi(n)$ 步"必然是整数圈，当然回到 1。

原根（生成元）

阶能取到最大值 $\varphi(n)$ 的元素，就叫原根。

课件定义为："对正整数 n ，如果 g 是模 n 的原根，那么对于任意的 $x \in \mathbb{Z}_n^*$ ，都存在 d 使得 $g^d \equiv x \pmod{n}$ 。"

换成"转圈"的话： g 转一圈，能把 $\mathbb{Z}_n^*$ 里所有元素都走一遍。

验证： $n = 9$ 时， $x = 2$ 的圈是 $2 \rightarrow 4 \rightarrow 8 \rightarrow 7 \rightarrow 5 \rightarrow 1$ ，正好走遍了 $\mathbb{Z}_9^*$ 全部 6 个元素 $\rightarrow 2$ 是原根。

而 $x = 4$ 的圈只有 $4 \rightarrow 7 \rightarrow 1$ 三个 $\rightarrow$ 不是原根。

怎么快速判断一个数是不是原根？

不用一个个试！课件给了判定定理：

设 $\varphi(n)$ 的素因子是 $q_1, q_2, \dots, q_k$ ，则 g 是原根的充要条件是：

$$g^{\varphi(n)/q_i} \not\equiv 1 \pmod{n}, \text{ 对每个 } i \text{ 都成立}$$

大白话：只要"半路没提前回到 1"，就是原根。

动手做：验证 3 是不是模 41 的原根

41 是素数， $\varphi(41) = 40$ 。把 40 分解： $40 = 2^3 \times 5$ ，素因子是 2 和 5。

所以只需要验两个指数： $40/2 = 20$ 和 $40/5 = 8$ 。

算 $3^8 \bmod 41$ （逐步平方）：

$$3^2 = 9$$

$$3^4 = 9^2 = 81, 81 - 41 = 40 \equiv -1 \pmod{41}$$

$$3^8 = (3^4)^2 \equiv (-1)^2 = 1 \pmod{41}$$

糟糕， $3^8 \equiv 1$ ！

按定理，只要有一个 $g^{\varphi(n)/q_i}$ 等于 1，就**不是**原根。

结论：3 不是模 41 的原根。

（补充：因为 $3^4 \equiv -1 \neq 1$ 、 $3^8 \equiv 1$ ，所以 3 的阶正好是 8。而 $8 \mid 40$ ✓，符合"非原根的阶整除 $\varphi(n)$ "。）

小心坑（两条）

- 除数用"素因子"，不是"因子"。40 的因子很多（2、4、5、8、10、20...），但定理只让你用素因子 2 和 5，也就是验 3^{20} 和 3^8 。
- 有一个等于 1 就"否决"。不是"全都等于 1 才否决"，而是"只要一个等于 1，就不是原根"。

▼ 自测：n = 7, x = 3。3 的阶可能是 4 吗？

不可能。

因为阶必须整除 $\varphi(7) = 6$ 。而 4 不能整除 6 ($6 \div 4 = 1.5$ ，不是整数)。

所以 3 的阶只能是 6 的约数：1、2、3 或 6。

实际算一下： $3^1=3, 3^2=2, 3^3=6, 3^4=4, 3^5=5, 3^6=1$ —— 所以 3 的阶是 6，而 $6 = \varphi(7)$ ，所以 3 是模 7 的原根。

⑩ 快速指数算法：手算大指数的手法

为什么需要它

欧拉定理能降指数，但不是万能的。有时候指数还是很大（比如 3^{80} ），那就需要一个“手算”的技巧。

核心思路：把指数写成二进制，利用“平方”来快速累积。

课件给的三元组方法

设置三个变量 (X, E, Y)，初始值为 (x, e, 1)。然后循环：

情况	怎么做
E 是奇数	$Y = X \cdot Y$ ，然后 $E = E - 1$
E 是偶数	$X = X \cdot X$ ，然后 $E = E / 2$
E = 0	停，此时 Y 就是答案

口诀：遇到奇数 → 乘到 Y 里、然后减 1；遇到偶数 → X 自己平方、然后除 2。

三个变量分工：X 攒平方 ($x, x^2, x^4, x^8 \dots$)；E 负责减半、减到 0 就停；Y 攒答案。

动手做：算 x^5

X	E	Y	说明
x	5	1	初始
x	4	x	E=5 是奇数： $Y = x \cdot 1 = x$ ； $E = 5 - 1 = 4$
x^2	2	x	E=4 是偶数： $X = x \cdot x = x^2$ ； $E = 4 / 2 = 2$
x^4	1	x	E=2 是偶数： $X = x^2 \cdot x^2 = x^4$ ； $E = 2 / 2 = 1$
x^4	0	x^5	E=1 是奇数： $Y = x^4 \cdot x = x^5$ ； $E = 1 - 1 = 0 \rightarrow$ 停

只做了 4 步（4 次乘法），而不是乘 5 次。

课件里的真实算例（带取模）

课件演示了算 $3^{80} \bmod 1000$ 和 $3^{100} \bmod 1000$ ：

$3^{80} \bmod 1000$				$3^{100} \bmod 1000$			
3	80	1		3	100	1	
9	40	1		9	50	1	
81	20	1		81	25	1	

561	10	1	561	12	81
721	5	1	721	6	81
721	4	721	841	3	81
841	2	721	281	2	121
281	1	721	281	1	121

→ $281 \times 721 = 601$ → $281 \times 121 = 1$

(每一步都取了模，所以数字都不大。)

▼ 自测：用三元组算 x^6 ，Y 最后等于多少？

X	E	Y	说明
x	6	1	初始
x^2	3	1	E=6 偶: $X = x^2$, $E = 3$
x^2	2	x^2	E=3 奇: $Y = x^2 \cdot 1 = x^2$, $E = 2$
x^4	1	x^2	E=2 偶: $X = x^4$, $E = 1$
x^4	0	x^6	E=1 奇: $Y = x^4 \cdot x^2 = x^6$, $E = 0$ → 停

$Y = x^6$ ✓

验算: $x^6 = x^{4+2} = x^4 \cdot x^2$ ✓ 正好是最后一步乘出来的。

⑪ 置换群与洗牌

为什么需要它

回忆换位密码：它不改变字母，只改变位置。这种“改位置”的操作，数学上就叫**置换**。

而“连着做几次置换会回到原样”这个问题，正好对应“洗几次牌能复原”。

定义

设 $S = \{1, 2, \dots, n\}$ ，S 上任何**双射**（一一对应）都叫一个 n 元**置换**，写成两行：

$$(1 \ 2 \ 3 \ \dots \ n / \delta(1) \ \delta(2) \ \delta(3) \ \dots \ \delta(n))$$

S 上所有 $n!$ 个置换，配上“复合运算 $\circ$ ”，构成**对称群** $\langle S_n, \circ \rangle$ ；它的子群叫**置换群**。

n 元置换一共有 $n!$ 个（比如 3 元置换有 $3! = 6$ 个）。

不相交循环：把置换写“圆”起来

看这个置换: $g = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 / 4 \ 3 \ 2 \ 5 \ 7 \ 6 \ 1)$

逐个跟一下: $1 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 1$ (绕回来了!); $2 \rightarrow 3 \rightarrow 2$; $6 \rightarrow 6$

所以它可以写成： $g = (1\ 4\ 5\ 7)(2\ 3)(6)$

这叫**不相交循环分解**——“不相交”指各个循环里的数字没有重复。

课件说：任何置换都能表示成不相交循环的乘积，**而且不考虑次序时是唯一的**。

关键公式（考试常考）

置换的阶 = 各循环长度的最小公倍数（lcm）

上面这个例子里：循环长度是 4、2、1，所以阶 = $\text{lcm}(4,2,1) = 4$ 。

意思是：把这个置换连做 4 次，所有元素都回到原位。

动手做： S_5 里最大的阶是多少？

思路：把 5 拆成几段，让各段长度的 lcm 最大。

拆法	lcm
5	5
4+1	4
3+2	6 ← 最大
3+1+1	3
2+2+1	2
1+1+1+1+1	1

答： S_5 的最大阶是 6（形如 $(1\ 2\ 3)(4\ 5)$ ）。

为什么是 3+2 而不是别的？因为 3 和 2 **互素**， $\text{lcm}(3,2) = 3 \times 2 = 6$ ，乘积没浪费。换成 4+1，lcm 只有 4。

S_{10} 呢？同理试：最优是 **5+3+2**（正好用完 10 个数）， $\text{lcm}(5,3,2) = 30$ 。

洗牌：换个场景用同一个概念

完美快速洗牌

一副 $2n$ 张牌，编号 $1, 2, \dots, 2n$ 。**完美快速洗牌**的规则是：

$$f(x) = (2x) \bmod (2n + 1)$$

课件结论：若 e 为 2 模 $2n+1$ 的阶（即满足 $2^e \equiv 1 \pmod{2n+1}$ 的最小正整数），那经过 e 次洗牌后，所有牌第一次回到起始位置。

动手做：8 张牌（课件例）

$2n = 8$ ，所以模数 = $2n+1 = 9$ 。洗牌规则： $f(x) = (2x) \bmod 9$ 。

洗一次的结果：

1 2 3 4 5 6 7 8 → 2 4 6 8 1 3 5 7

（自己验一个：5 号位的牌 $f(5) = 10 \bmod 9 = 1$ ，所以 5 号位 → 1 号位 ✓）

跟踪 1 号的轨迹： $1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 7 \rightarrow 5 \rightarrow 1$

噢，走到 5 就回 1 了！只串了 6 个位置（1,2,4,8,7,5），还剩 3 和 6 没串到。

再另起一头，从 3 开始： $3 \rightarrow 6 \rightarrow 3$ （串了 2 个）

所以循环分解是 **(1 2 4 8 7 5)(3 6)**，长度分别是 6 和 2。

洗牌次数 = $\text{lcm}(6, 2) = 6$ 次 ✓ 和课件一致。

动手做：12 张牌（你之前问过的那道）

$2n = 12$ ，模数 = **13**。规则： $f(x) = (2x) \bmod 13$ 。

从 **1** 开始跟踪，一路走：

1 → 2 → 4 → 8 → 3 → 6 → 12 → 11 → 9 → 5 → 10 → 7 → 回到 1

（中间用到的取模： $16 \bmod 13 = 3$ ； $24 \bmod 13 = 11$ ； $22 \bmod 13 = 9$ ； $18 \bmod 13 = 5$ ； $20 \bmod 13 = 7$ ； $14 \bmod 13 = 1$ ）

这一趟把 **12 个位置全部串完了**，所以只有一个循环：

(1 2 4 8 3 6 12 11 9 5 10 7)

洗牌次数 = **12**。

两个容易错的地方

① 模数是 $2n+1$ ，不是 $2n$ 。12 张牌对应模 **13**，这一步错就全错。

② 中间会出现比 $2n$ 大的数。12 张牌的例子里出现了 16、24、22、18、20、14 —— 这些只是“乘 2 之后的中间量”，取模之后必然回到 1~12，不是错误。

顺便：如果只有一个循环，洗牌次数就是这个循环的长度；如果有多个循环（像 8 张牌那样），就要取**各循环长度的最小公倍数**。

▼ 自测：14 张牌的循环分解是什么？洗几次复原？

$2n = 14$ ，模数 = **15**。规则 $f(x) = (2x) \bmod 15$ 。

课件给了答案：**(1 2 4 8)(3 6 12 9)(5 10)(7 14 13 11)**

四段循环的长度分别是 4、4、2、4。

洗牌次数 = $\text{lcm}(4, 4, 2, 4) = 4$ 次。

验证一段：从 1 开始：1 → 2 → 4 → 8 → $16 \bmod 15 = 1$ ✓ 确实是长度 4 的循环。

再验一段：从 3 开始：3 → 6 → 12 → $24 \bmod 15 = 9$ → $18 \bmod 15 = 3$ ✓ 长度 4。

注意这里有 **4 个循环**，所以要取 lcm，不能只看某一个的长度。

1.4 节总结：这一节的六个“零件”

零件	一句话	后面用在哪
约简	取余数，负数要加模数变正	全书处处
同余	相差 m 的整数倍 = 同一个数	全书处处
Euclid / 逆元	凑出 $ax + by = 1$ ， a 就是逆元	仿射密码、RSA、坏种子
群 / 剩余系	分箱子；能自由做运算的世界	$\mathbb{Z}_p^*$ 、RSA、DH
欧拉函数 / 定理	数个数；转 $\phi(n)$ 步回到 1	降指数、RSA 可解性

阶 / 原根	实际走几步；走遍全圈的叫原根	周期、LFSR、洗牌
--------	----------------	------------

这六个零件练熟，第 2 章就只是"换场景用零件"而已。

第 2 章

现代密码算法

这一章要解决什么

第 1 章的古典密码，密钥空间小、规律容易被统计出来。现代密码要解决两件事：

- 1. 怎么让"改写"复杂到没有统计规律？（对称密码、序列密码）
- 2. 怎么在不见面的情况下共享密钥？（公钥密码）

另外还有两个工具：给文件录指纹的**数字摘要**，和用于抗否认的**数字签名**。

建议：这一章先抓"每个算法是干什么用的、靠什么难题撑住的"，具体轮数、盒子的细节记不住没关系。

2.1 对称密码：把明文切块再揉碎

先确认概念

对称密码 = 加密密钥和解密密钥**相同**（或者能互相推出来）。

课件用的比喻：**把文件放进保险柜、设密码锁上；开柜子要用同一个密码。**

特点：运算速度快、适合加密大量数据；但**密钥管理复杂**。

为什么现代密码要"分块"

古典密码是逐字母处理的。现代密码改成**把明文切成固定长度的块，一块一块加密**，这叫**分组密码**。

为什么？因为分块之后，可以让**每一块里的每一位都互相影响**——这就是抵抗统计攻击的关键。

核心思想：混淆与扩散

DES 的两个基本手段

混淆：复杂化密文与明文、密钥之间的统计特性

扩散：将每一位明文的影响尽可能分散到多个输出密文中

大白话

扩散 = **改一个字母，整段密文都变样**。

这样攻击者就没法"一个字母一个字母地啃"了——动一处，全盘皆动。

混淆 = **把密钥和密文的关系搅乱**。

这样攻击者没法从密文里"看出"密钥的痕迹。

这两个词听起来玄，其实就是两种"打乱"的手法。**后面所有现代密码都是围绕这两个字做文章**。

DES 的五个关键数字

DES（1977 年颁布，数据加密标准）

项目	数值	怎么记
分组长度	64 位	每次处理 64 个 0/1
密钥长度	64 位，但实际只有 56 位	有 8 位是奇偶校验位，不参与加密
迭代轮数	16 轮	把同一套操作重复 16 次

子分组	32 位（左右两半）	64 位劈成两半
子密钥数	16 个	每轮用一个

DES 走的三步

① **初始置换 IP**：按一张固定的表，把 64 位重新排一遍

↓

② **16 轮迭代的乘积变换**：核心部分，反复"揉"

↓

③ **逆初始置换 IP^{-1}** ：再排一遍，把顺序调回来

课件说 IP 和 IP^{-1} 的作用主要是"打乱输入的 ASCII 码字划分关系，并将明文校验码变成 IP 输出的一个字节"。

为什么要先打乱位置？ 因为要让原来相邻的位分散开，这样后面每一轮的影响才能扩散得更均匀。

核心：那 16 轮在干什么

把 64 位劈成左右两半 L 和 R，每轮这样操作：

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

逐项看懂：

- 右边那半原封不动变成左边的 ($L_i = R_{i-1}$)
- 新的右边 = 老的左边 异或 一个"用右边和子密钥算出来的东西"
- $\oplus$ 是按位异或，规则： $1 \oplus 0 = 1$, $0 \oplus 1 = 1$, $0 \oplus 0 = 0$, $1 \oplus 1 = 0$

这个结构叫 **Feistel 结构**，妙处在于：加密和解密用同一套操作，只是子密钥的顺序反过来。所以硬件实现很省事。

S 盒：DES 唯一真正"非线性"的地方

函数 f 里面最要紧的是一个 **S 盒（替换盒）**：

输入 48 位 → 分成 8 组，每组 6 位

每组过一个 S 盒 → 变成 4 位

$8 \times 4 = 32$ 位输出

每个 S 盒：输入 6 位，输出 4 位。

为什么 S 盒这么关键？ 因为其他步骤（置换、异或）都是线性的——线性的东西可以用数学方程解出来。而 S 盒是一张查表替换表，非线性的，让攻击者无法用解方程的办法攻破。

DES 的安全性，几乎全押在这 8 个 S 盒上。

DES 的问题

DES 的安全性完全依赖密钥，但 56 位的密钥太短了：

攻击方式	需要多少次尝试
穷举攻击	$2^{56} \approx 10^{17}$ 次
线性密码分析（Matsui 提出）	约 2^{43} 次
差分密码分析（Biham 与 Shamir 提出）	约 2^{47} 次

10^{17} 看着很大，但用专用硬件几个月就能算完。所以出现了三重 DES 等改进版。

IDEA：换个思路做混淆

IDEA 的五个关键数字

项目	数值
设计者 / 时间	来学嘉和 James Massey, 1990 年公布（1992 年更名 IDEA）
分组长度	64 位（分成 4 个 16 位子分组）
密钥长度	128 位（比 DES 长一倍多）
迭代轮数	8 轮
子密钥数	52 个
数学运算	异或、模 2^{16} 加、模 $2^{16}+1$ 乘

特点：

- 强化了抗差分分析能力，应用于 PGP 中
- 软件实现 IDEA 比 DES 快两倍
- 乘加结构（MA）保证了扩散的特点

IDEA 的聪明之处

DES 靠 S 盒（查表）来制造非线性，IDEA 靠“混合三种运算”来制造非线性。

为什么混三种运算就能非线性？因为在模运算里：

- 加法、乘法、异或，这三种运算的“结构”完全不同
- 把它们混在一起，就没法用统一的方法化简

课件说它“强化了抗差分分析能力”——差分分析专门找运算的规律，混了三种运算之后就难找了。

DES vs IDEA vs AES 对比表（背这张就够）

	DES	IDEA	AES
颁布年份	1977	1990	2002
分组长度	64	64	128
密钥长度	64（实际 56）	128	128/192/256
子分组	32 位	16 位	8 位（字节）
轮数	16	8	10/12/14
子密钥数	16	52	11/13/15
数学运算	S-Box、 $\oplus$	$\oplus$ 、+、 $\times$	S-Box、 $\oplus$ 、 $\times$

记忆窍门：AES 的轮数随密钥变长而增加（128 $\rightarrow$ 10 轮，192 $\rightarrow$ 12 轮，256 $\rightarrow$ 14 轮），密钥总数是轮数加 1。

课件给了个公式：轮数 $r = l/32 + 6$ （ l 是密钥位数）。验证： $l=128$ 时 $128/32+6 = 4+6 = 10 \checkmark$ ； $l=256$ 时 $8+6 = 14 \checkmark$

AES：现在的主流

AES 的四个步骤

AES 把 128 位明文排成 **4×4 的字节矩阵**（共 16 个字节），然后做四件事：

步骤	做什么	类比
字节替换 SubBytes	用一张 S 盒把每个字节替换成别的（非线性）	相当于 DES 的 S 盒—— 制造混淆
行移位 ShiftRows	第 1~4 行分别向左移动 0~3 列	制造扩散 （列内元素被打散到不同列）
列混合 MixColumns	用矩阵乘法把每列重新组合	制造扩散 （列内互相影响）
轮密钥加 AddRoundKey	和对应的子密钥 异或	混入密钥信息

这四步重复 10~14 轮（**最后一轮不做列混合**）。

用大白话理解这四个步骤

想象你在揉面团：

- **字节替换**：换材料（把面粉换成糖粉）——**改内容**
- **行移位**：把面团横着挪一挪——**换位置**
- **列混合**：竖着再把几股面揉到一起——**交叉影响**
- **轮密钥加**：撒一把"只有你有的调料"——**掺入密钥**

一遍不够，所以揉十几遍。**揉的遍数越多，越难从成品反推配方。**

AES 里那个奇怪的多项式是什么

你可能注意到课件里有这么一句："AES 中以字节为单位进行模 $x^8+x^4+x^3+x+1$ 运算"。

这个东西叫**不可约多项式**，是"列混合"那一步要用的。

为什么需要它？ 因为列混合要求"把字节做乘法，而且还能除回来"。

可字节是 8 位的（0~255），普通乘法会出界： $200 \times 200 = 40000$ ，早就超过 255 了。所以必须取模。

那模什么好呢？

- 模 256？不行。256 不是素数，很多数没有逆元，**除不回来**。
- 模 257？可以（257 是素数），但结果是 0~256，**257 个数，装不进 1 个字节**。

两难。于是密码学家请出了多项式：把一个字节的 8 个 bit 看成一个 **8 项多项式的系数**。

把字节变成多项式

课件给的例子：字节 **d4** = 二进制 11010100

从高到低对应 $x^7, x^6, \dots, x^0$ ：

1 1 0 1 0 1 0 0

↓（哪些位是 1，就取哪一项）

$x^7 + x^6 + x^4 + x^2$

在这个"多项式世界"里，运算规则变了：

运算	规则	例子
----	----	----

加法 / 减法	就是按位异或（因为模 2 下加和减一样）	$1110 \oplus 1001 = 0111$
乘 2 / 乘 x	就是按位左移一位	$(1001) \times (10) = 10010$
乘完溢出	减去（即异或）模多项式，把次数降下来	见下

课件算例：02 × d4 怎么算

$02 \times d4 = x \cdot f(x) = x^8 + x^7 + x^5 + x^3$ ← 先左移一位（次数变成 8，溢出了）

$= x^8 + x^7 + x^5 + x^3 - (x^8 + x^4 + x^3 + x + 1)$ ← 减去模多项式

$= x^7 + x^5 + x^4 + x + 1$

$= 10110011 \rightarrow b3$

课件在旁边标了“15 mod 11”作类比——意思就是：就像十进制里“15 模 11 降下来”，这里是“9 次多项式模掉一个 8 次多项式，降回 8 次以下”。

这样既保证了结果还是一个字节（8 位），又保证了每个非零元素都有逆元（能做除法）。

前后呼应

那个模多项式 $x^8 + x^4 + x^3 + x + 1$ 必须是不可约多项式（多项式世界里的“素数”）。

课件还专门验证了一遍：GF(2⁸) 中次数 ≤ 4 的不可约多项式有 $x, x+1, x^2+x+1, x^3+x+1, x^3+x^2+1, x^4+x+1, x^4+x^3+1$ ，都不能整除它——所以它确实不可约。

下一节讲 LFSR 的时候，还会遇到一个相关的词：“本原多项式”。到时候一起说。

▼ 自测：AES 的密钥是 192 位，那它要跑几轮？

12 轮。

用课件的公式 $r = l/32 + 6$ ：

$$r = 192/32 + 6 = 6 + 6 = 12$$

顺便记住三个对应关系：

- 密钥 128 位 → 10 轮 → 11 个子密钥
- 密钥 192 位 → 12 轮 → 13 个子密钥
- 密钥 256 位 → 14 轮 → 15 个子密钥

为什么密钥越长轮数越多？因为密钥越长，需要更多轮才能把密钥的信息“充分揉进”密文里，让攻击者更难找出规律。

2.2 序列密码：一边生成一边异或

先搞清楚它和分组密码的区别

	分组密码	序列密码
--	------	------

处理方式	把明文切块，一块一块加密	按字节或位逐个加密
代表	DES、IDEA、AES	RC4、LFSR
特点	扩散性好	简单快速

核心公式（一句话）

明文 ⊕ 密钥流 = 密文

这里的"密钥流"是一串看起来随机的字节，它怎么生成，就是各个算法的差别。

三种生成方式，产出不同类型的东西：

方式	产出
RC4	字节流
同余发生器	数据流（一系列数）
移位寄存器 LFSR	位流（0/1）

铁律：密钥流绝对不能重复使用

这是序列密码的命门，课件专门用一页讲。为什么？

假设攻击者知道了一段明文 P'，而你又用同一串密钥流加密了别的消息：

原始明文：P1 P2 P3 P4	新明文：P1 P' P2 P3 P4
原始密钥：k1 k2 k3 k4	新密钥：k1 k2 k3 k4 k5
原始密文：C1 C2 C3 C4	新密文：C1 C2' C3' C4' C5'

因为攻击者知道 P'，他就能：

$k2 = P' \oplus C2'$	← 先算出密钥
$P2 = C2 \oplus k2$	← 再用密钥解出明文
$k3 = P2 \oplus C3'$	← 又算出下一个密钥
$P3 = C3 \oplus k3$	← 又解出一个明文
... 一环扣一环，整段明文全暴露	

根子在于：C = P ⊕ k 这个式子，只要 P 和 C 知道其中之一，另一边就藏不住。

RC4：用一个 S 盒洗牌

RC4 参数

- 设计者：Ron Rivest, 1987 年（也是 RSA 里的那个 R）
- 简单，速度快，10 倍于 DES 的速度
- 面向字节的流密码，密钥独立于明文

- 应用广泛，很多 HTTPS 网站用过 RC4
- 核心是一个 **256 字节的 S 盒**： $S_0, S_1, \dots, S_{255}$ ，每个位置存的值是 **0 到 255 之间不重复的值**
- **密钥长度可变**

把 S 盒想象成一副扑克牌

256 个位置，装着 0~255 这 256 个数字，**每个恰好出现一次**——就像一副从 0 排到 255 的牌。

RC4 做的两件事：

1. **用密钥把牌洗乱**（初始化）
2. **不断再洗、顺手抽出一张**（生成密钥流）

第一步：S 盒初始化（拿密钥把牌洗乱）

- ① 先顺序填充： $S_0 = 0, S_1 = 1, \dots, S_{255} = 255$
- ② 用初始密钥 key 填充另一个 256 字节的数组 K（密钥不够长就循环重复）
- ③ 然后循环 256 次，用 K 来“拌”这个 S 盒：

$j = 0$

对于 $i = 0$ 至 255：

$j = (j + S_i + K_i) \bmod 256$

交换 S_i 和 S_j

意义：密钥相同，洗出来的牌序就相同（这符合“密钥独立于明文”）；密钥不同，就是两副完全不同的牌。

第二步：生成密钥流（反复洗牌 + 抽牌）

$i = j = 0$

$i = (i + 1) \bmod 256$ ← i 一格一格往前走

$j = (j + S_i) \bmod 256$ ← j 跳多远，由 S 盒里当前位置的值决定

交换 S_i 和 S_j ← 真正在洗牌

$t = (S_i + S_j) \bmod 256$ ← 算出取牌位置

$k = S_t$ ← 从 t 位置抽出一张牌，这就是这一轮的密钥流字节

加密：明文 $\oplus$ 密钥流 $\rightarrow$ 密文；解密反过来再来一次异或。

速度优势就在这：初始化要几百次运算，但之后每个明文字节只需要一次异或（加解密完全对称）。

三个容易漏的点

- 1. S 盒是一个"排列"，不是随便赋值——0~255 每个值恰好出现一次。初始化时是"交换"，不是"覆盖"，所以它可逆、可解密。
- 2. 必须先顺序填充，再打乱。如果跳过第一步直接打乱，牌序的随机性基础就不牢。
- 3. 到处是 mod 256——因为 S 盒只有 256 格，所有下标运算都要绕回来。

线性同余发生器：用一个公式反复算

公式

$$s_{n+1} = (a \cdot s_n + b) \bmod m$$

其中 $0 < a < m$ ， $0 \leq b < m$ ，初值 s_0 叫种子。

课件原话："a、b、m 的取值是产生高质量随机数的关键。"

四个符号的分工

符号	作用
a	乘数——每步"放大"多少
b	增量——每步固定加多少
m	模数——把结果圈回 0~m-1 的范围
s ₀	种子——序列的起点

为什么必须有 mod m？没有它序列会无限增大，计算机存不下；有了它，序列永远在 m 个数字里打转，才可能被当成随机数用。

动手做：跟着课件算一遍

取 $a=2, b=7, m=17$ ，则 $s_{n+1} = (2s_n + 7) \bmod 17$ 。取种子 $s_0 = 1$ ：

步骤	计算	结果
起点	—	1
1	$2 \times 1 + 7 = 9$	9
2	$2 \times 9 + 7 = 25, 25 - 17 = 8$	8
3	$2 \times 8 + 7 = 23, 23 - 17 = 6$	6
4	$2 \times 6 + 7 = 19, 19 - 17 = 2$	2
5	$2 \times 2 + 7 = 11$	11
6	$2 \times 11 + 7 = 29, 29 - 17 = 12$	12
7	$2 \times 12 + 7 = 31, 31 - 17 = 14$	14
8	$2 \times 14 + 7 = 35, 35 - 34 = 1$	1 ← 回到起点！

课件写的序列：1, 9, 8, 6, 2, 11, 12, 14, 1, 9, 8, 6, ...

注意：只走了 7 步就回到 1，后面开始原封不动地重复。周期只有 7。

对比一下另一组参数

课件还列了 $a=7, b=1$ ，种子 $s_0=14$ 的情况：

1, 8, 6, 9, 13, 7, 16, 11, 10, 3, 5, 2, 15, 4, 12, 0, **1, 8, ...**

这一串走了 **16 步**才回到 1。

同样是模 17，为什么差别这么大？ 这就是"周期"要讲的内容。

周期由什么决定

课件原话：线性同余发生器的周期为 a 在乘法群 Z_p^* 中的阶 l 。

这句话把 1.4 节的三个概念串起来了，逐句翻译：

课件原文	对应 1.4 节的哪个概念
" Z_p^* "	简化剩余系（第 ⑥ 条）
" a 在 Z_p^* 中的阶 l "	阶（第 ⑨ 条）—— a 自乘多少次回到 1
" p 为素数时 $l \mid p-1$ "	费马小定理（第 ⑧ 条）—— $a^{p-1} \equiv 1$
" a 为本原根时 $l = p-1$ "	原根（第 ⑨ 条）——阶最大的那个

回到上面两个例子（ $m=17$ ）：

- $a=2$ ：2 的阶是 8（因为 $2^8 = 256 = 15 \times 17 + 1$ ），而 $p-1 = 16$ 。8 只是 16 的一半 → **周期短（7）**
- $a=7$ ：7 是模 17 的本原根，阶 $= 16 = p-1$ → **周期最长（16）**

结论：想让周期尽量长，就挑本原根当 a 。

坏种子：一个必须躲开的坑

$$s_{\text{bad}} \equiv -(a-1)^{-1} \cdot b \pmod{p}$$

为什么叫"坏"？ 如果一个种子 s 满足"算完还是它自己"：

$$s = (a \cdot s + b) \pmod{p}$$

$$\rightarrow s - a \cdot s = b$$

$$\rightarrow s(1 - a) = b$$

$$\rightarrow s = b/(1-a) = -(a-1)^{-1} \cdot b \quad \leftarrow \text{正好就是课件那个公式}$$

种子落在这个点上，序列就永远是它自己，周期退化成 1。

动手做：课件例题

求 $s_{n+1} = 7s_n + 2 \pmod{13}$ 的坏种子。

$a=7, b=2, p=13$ ，代入公式：

$$s_{\text{bad}} = -(7-1)^{-1} \times 2 = -6^{-1} \times 2 \pmod{13}$$

先求 $6^{-1} \pmod{13}$ （用 Euclid）：

$$13 - 2 \times 6 = 1 \rightarrow 1 = 13 - 2 \times 6 = (-2) \times 6 + 1 \times 13$$

所以 $6^{-1} = -2 = \mathbf{11}$ (加 13 变正)。

$$s_{\text{bad}} = -11 \times 2 = -22, -22 + 26 = \mathbf{4}$$

验算：把 $s = 4$ 代进去, $(7 \times 4 + 2) = 30, 30 \bmod 13 = \mathbf{4}$ —— 果然算完还是 4, 卡死了 ✓

设计结论与本质缺陷

课件给的结论：

m 为大素数, a 为模 m 的本原根, s_0 不为坏种子, 以保证最大周期。

三条要求, 对应三个坑：

要求	躲开什么
m 取大素数	模数小 $\rightarrow$ 序列短; 非素数 $\rightarrow$ 阶的规律变乱
a 取本原根	不是本原根 $\rightarrow$ 阶只是 $p-1$ 的真因子, 周期缩水
s_0 避开坏种子	落进不动点 $\rightarrow$ 周期退化成 1

但最重要的是这句 (课件用红字点出)：

算法不具备随机性。

因为不管参数多好、周期多长, 它的本质就是一个**确定性递推公式**：知道 a, b, m 和任意一项, 后面全都能算出来。这就是它叫“**伪**随机数发生器”的原因。

LFSR：移位寄存器

为什么需要它

它产出的是“**位流**” (0/1), 而且用硬件实现特别简单——所以课件说它“广泛应用于**军事密码学**”。

组成与公式

由两部分组成：**移位寄存器**和**反馈函数**。

$$b_{n+1} = c_n b_n + c_{n-1} b_{n-1} + \dots + c_2 b_2 + c_1 b_1, \quad c_i \in \{0, 1\}$$

参与异或的位置叫**抽头**, 一串 c_i 叫**抽头序列**。

注意：模 2 下的加法就是**异或** ($1+1=0$)。

大白话：一条流水线

$b_4 \quad b_3 \quad b_2 \quad b_1$

| | | |

← 整体向左移

↑

⊕ ←—— 把"抽头"位置的值异或起来，补到最右端

每个时钟周期：所有位左移一格，最右边补入一个"算出来的新位"，最左边被挤出去的那位就是输出。

c_i 就是"开关"：等于 1 表示这一位参与异或，等于 0 表示不参与。

动手做：跟课件算一遍

4 位 LFSR，抽头第 1、4 位，即 $b_{n+1} = b_n + b_{n-3}$ （最左位和最右位异或）。初始值 1111。

步骤	寄存器状态	输出位	新 bit = 首位 ⊕ 末位
开始	1111	—	—
1	1110	1	$1 \oplus 1 = 0$
2	1100	0	$1 \oplus 0 = 1$
3	1001	0	$1 \oplus 0 = 1$
4	0010	1	$1 \oplus 1 = 0$
5	0100	0	$0 \oplus 0 = 0$
6	1000	0	$0 \oplus 0 = 0$

把每一步挤出去的位收集起来：**111101011001000...** ✓ 和课件一致。

注意：输出序列是"从右端挤出去的那一位"，**不是寄存器当前的内容**。这两者容易混。

周期：为什么是 2^n-1 而不是 2^n

课件原话： **n 位 LFSR 中，具有特殊抽头序列的才有最大周期 2^n-1** ，在该类抽头序列加上常数 1 形成的多项式必须是模 2 的本原多项式。

先说为什么减 1： n 位寄存器一共有 2^n 种状态，但全 0 状态是死胡同：

0000 → 新 bit = $0 \oplus 0 = 0$ → 0000（永远卡在这）

所以真正能用的是 $2^n - 1$ 种。**这一点和"坏种子"是同一个道理。**

什么是"本原多项式"

把抽头序列写成一个多项式。比如抽头第 1、4 位（抽头序列记为 (4,1,0)）：

抽头序列 (4, 1, 0) → 多项式 $x^4 + x + 1$

（第 4 位给 x^4 ，第 1 位给 x ，最后那个"0"就是课件说的"加上常数 1"。）

"本原多项式"的含义和"本原根"是一回事——它能让寄存器跑满所有非零状态，一个不漏。

课件对比了两个抽头：

多项式	课件标注	周期
$x^4 + x + 1$	—	$15 = 2^4 - 1$ ✓ 跑满
$x^4 + x^2 + x + 1$	有因子 $x^3+x^2+1, x+1$	只有 6

右边那个能被分解（不是不可约的，更谈不上本原），所以提前循环，周期短。

结论：同样 4 位寄存器，抽头选得对不对，周期天差地别。

课件还给了一组 32 位的：抽头序列 (32,7,5,3,2,1,0) 和 (32,7,6,2,0) 具有最大周期 $2^{32}-1$ 。

把两个“本原”对照起来看（这是最好的记法）

	整数世界（同余发生器）	多项式世界（LFSR）
结构	乘法群 $\mathbb{Z}_p^*$	多项式环 $\mathbb{F}_2[x]$
元素个数	$p-1$	2^n-1
跑满全部的资格	本原根	本原多项式
“钥匙”是什么	选 a	选抽头序列
最大周期	$p-1$	2^n-1

两边都在问同一个问题：“怎么选参数才能跑满最大周期”。名字换了，思想没换。

而且两边都要减 1，原因也一样——“全 0”那个位置是死胡同。

三种密钥流生成方式总对比

	RC4	线性同余	LFSR
产出	字节流	数据流	位流
随机性来源	256 字节 S 盒洗牌	线性递推公式	移位 + 异或反馈
内部状态	256 字节排列	1 个数	n 个 bit
最大周期	极长	$p-1$ (a 为本原根)	2^n-1 (本原多项式)
抗分析	非线性，较难	纯线性，易解	纯线性，易解

最关键的区别：RC4 的“状态”是一个 256 字节的排列，庞大且非线性；后两者的状态分别只有“一个数”和“n 个 bit”，而且都是纯线性的——攻击者拿到连续几项就能立方程解出来。

所以课件在原话里强调：同余发生器“算法不具备随机性”；而 LFSR 主要靠硬件工艺保密 + 多路组合来补安全性。

▼ 自测：为什么序列密码的密钥流绝对不能重复使用？

因为 $C = P \oplus k$ 这个式子是一把双刃剑：只要攻击者知道 P 和 C 中的任何一个，就能算出 k ；反之知道 k 就能算出 P 。

一旦你用同一段密钥流加密了两条消息，而攻击者知道其中一条的内容（比如“meet me at ten”这种格式已知的开头），他就能：

1. 用已知明文 $\oplus$ 密文 $\rightarrow$ 拿出密钥流
2. 再拿密钥流去解另一条消息 $\rightarrow$ 全部暴露

所以序列密码的使用规矩是：一次一密钥流，用完作废。这也是 6.3 / 6.4 节的 CFB、OFB 模式要处理的问题——它们本质上都是“把分组密码当密钥流发生器用”。

2.3 数字摘要：给文件录指纹

为什么需要它

回忆 1.1 节那个“完整性和鉴别”的场景：Alice 怎么让 Bob 知道消息**没被篡改**？

如果她把整段消息重新发一遍做对比，那既费流量又没意义（万一消息本身被改了，两份都是改过的）。

她只需要发一个“指纹”——消息的摘要。

大白话

数字摘要 = 文件的指纹。

- 不管文件多大（1KB 还是 1GB），算出来的指纹都是**固定长度**的
- 文件动一点点，指纹就**面目全非**
- 从指纹**反推不出文件内容**

所以：Bob 收到消息后自己算一遍指纹，和 Alice 寄来的对比。**对不上，就是被改过。**

散列函数的四条性质（必背）

性质	含义	别称
公开性	算法公开	—
单向性	由 $x \rightarrow h(x)$ 容易，但 $h(x) \rightarrow x$ 难	—
扩散性	输入 改变一个位 ，输出散列值中 一半的位 都会变	雪崩效应
抗碰撞性	$x \neq y$ 时， $h(x) \neq h(y)$ （两者相等的概率极小）	无碰撞性

作用：① 保证消息的完整性；② 用在数字签名中，**提高签名效率**。

课件还提到一种特殊形式：**消息鉴别码 MAC**——“与密钥有关的单向散列函数”。

怎么理解“雪崩效应”

输入改一个 bit，输出一半的 bit 都翻转——这叫雪崩。

为什么这个性质重要？ 因为攻击者没法“一点一点试”：他改一个字符想碰运气，结果整个指纹全变了，根本得不到任何“接近了”的反馈信号。

（课件在 2.1 节的作业里也问过：“分析 AES 算法中的雪崩效应”——同一个思想。）

MD5 与 SHA 系列

算法	输入值	输出（位）	备注
MD5	任意	128	Rivest 1992 年设计；5 个步骤、4 轮每轮 16 次
SHA1	2^{64}	160	NIST 1993 年采纳；基于 MD4；4 轮 80 次迭代
SHA224	2^{64}	224	—
SHA256	2^{64}	256	目前主流
SHA384	2^{128}	384	—
SHA512	2^{128}	512	—

SHA 流程五步：附加填充比特 → 附加长度 → 初始化缓冲区 → 80 次迭代 → 输出。

一句话概括：512 位分组，160 位输出，5 个 32 位寄存器，4 轮 80 次迭代。

“附加填充比特”是什么意思

散列函数要求输入长度是 512 的整数倍。但消息长度任意，所以要“补齐”。

规则：在消息后填一串 **1000...**，使长度满足 $|x| \equiv 448 \pmod{512}$ ，然后在后面再补 64 位表示原始长度。

课件举例： $x = \text{"abc"}$

a = 01100001 b = 01100010 c = 01100011

填充后: 01100001 01100010 01100011 1000...0000 (补 423 个 0)

再加 64 位长度: 000...11000

为什么要补长度? 防止"长度扩展攻击"——如果不记录原始长度, 攻击者可能构造出另一条消息得到同样的摘要。

安全性现状 (很重要)

王小云教授于 2004 年破译 MD5 和 SHA-1, 目前普遍采用 SHA256。

所以你在实际使用中看到 `sha256RSA` 这样的写法 (课件里问过"为什么是 sha256RSA"), 原因就是: **先用 SHA256 把消息压成 256 位摘要, 再用 RSA 对摘要签名**——既安全又高效。

▼ 自测: 为什么数字签名要"先摘要再签名", 而不是直接对整段消息签名?

为了效率。

RSA 这类公钥运算**非常慢** (比对称密码慢上千倍)。如果要签名的消息有 10 MB, 直接对它做 RSA 运算会慢得没法用。

改成: 先用 SHA256 把 10 MB 压成 256 位 (32 字节) 的摘要, 再对这 32 字节做一次 RSA 运算。**速度提升好几个数量级。**

那安全性会不会下降? 不会——因为散列函数有**抗碰撞性**: 攻击者没法找到另一条消息产生相同的摘要。所以他改不了消息。

这就是课件说的: "数字摘要用在数字签名中, **提高签名效率。**"

2.4 公钥密码: 不用见面的密钥

先回到那个死循环

第 0 章留了个问题: Alice 和 Bob 要通信, 得先有共享密钥。可密钥怎么安全地送过去?

想安全传密钥 → 得先加密

想加密 → 得先有密钥

→ 死循环

古典密码的解法是**"事先当面约好"**——所以只能熟人之间用。

1976 年, Diffie 和 Hellman 打破了这个循环。

公钥密码的基本设定

- 1976 年 Diffie 和 Hellman 提出
- **特点**: 加密密钥公开, **密钥管理简单**
- 代表算法: RSA、ElGamal、ECC 等
- **安全性**: 基于数学难题, **运算速度较慢**

算法

依赖的数学难题

Knapsack	背包问题（已证明不安全）
RSA	因子分解难题
ElGamal	离散对数难题
ECC	椭圆曲线点群上的离散对数难题

核心思想：一对钥匙，一公开一保密

公钥密码的巧妙之处在于：每个人有一对密钥——公钥和私钥。

- **公钥**：随便公开，贴在网站上、写在名片上都行
- **私钥**：只有自己知道，绝不外传

这两个钥匙的关系是：

用公钥加密的东西，只有对应的私钥能解开（反过来也行）

而且——从公钥推不出私钥。（这就是它安全的根基。）

用它来解决问题（两种用法）

用法一：保密通信

Bob 把自己的公钥 k_{Bp} 公开（谁都能拿）

Alice 用 Bob 的公钥加密消息： $c = E_{Bp}(m)$

只有 Bob 能用私钥 k_{Bv} 解密： $m = D_{Bv}(c)$

全程不需要事先共享任何秘密！

用法二：数字签名

Alice 用自己的**私钥**“加密”消息： $s = E_{Av}(m)$

任何人都能用 Alice 的**公钥**验证： $D_{Ap}(s)$ 是否等于 m

"只有她能做，但谁都能验" —— 这就是签名的本质

小心坑：两对密钥不要搞混

用途	用什么加密	用什么解密
保密通信	对方的公钥	自己的私钥
数字签名	自己的私钥	自己的公钥（别人来验）

记忆口诀：

保密 → **"我给别人发"**，所以用别人的公钥（只有他能开）

签名 → **"我证明是我发的"**，所以用我的私钥（只有我能做）

课件还特别提醒：**"保密通信的密钥对不同于数字签名的密钥对"**——实际上应该用两对不同的密钥。

2.4.2 RSA：最流行的公钥算法

RSA 的三步

【生成密钥】

- ① 选两个大素数 p 和 q
 - ② 算 $n = p \times q$ ，以及 $\varphi(n) = (p-1) \times (q-1)$
 - ③ 选一个 e ，要求 $\gcd(e, \varphi(n)) = 1$
 - ④ 算 $d = e^{-1} \bmod \varphi(n)$
- 公钥 (e, n) 对外公开；私钥 d 自己留好
(p 、 q 、 $\varphi(n)$ 也要严格保密)

【加密】 $c = m^e \bmod n$

【解密】 $m = c^d \bmod n$

动手做：跟课件算一个小例子

课件取 $p=3, q=5$ 。

$$n = 3 \times 5 = 15$$

$$\varphi(n) = (3-1) \times (5-1) = 2 \times 4 = 8$$

选 $e = 3$ ，检查 $\gcd(3, 8) = 1 \checkmark$

求 $d = 3^{-1} \bmod 8 \Rightarrow$ 因为 $3 \times 3 = 9 \equiv 1 \pmod{8}$ ，所以 $d = 3$

公钥 $(e, n) = (3, 15)$ ，私钥 $d = 3$ 。（这个例子太小了，现实中 p 、 q 是几百位的大素数。）

加密消息 $m = 7$ ：

$$c = 7^3 \bmod 15 = 343 \bmod 15$$

$$343 \div 15 = 22 \text{ 余 } 13 \Rightarrow c = 13$$

解密 $c = 13$ ：

$$m = 13^3 \bmod 15 = 2197 \bmod 15$$

$$2197 \div 15 = 146 \text{ 余 } 7 \Rightarrow m = 7 \checkmark \text{ 还原了}$$

为什么解密一定能还原？（用欧拉定理）

这是 RSA 最漂亮的地方。课件给了推导：

已知 $c = m^e$

而 $ed \equiv 1 \pmod{\varphi(n)}$ ，所以 $ed = 1 + k \cdot \varphi(n)$ (k 是某个整数)

$$c^d \bmod n = m^{ed} \bmod n$$

$$= m^{(1 + k \cdot \varphi(n))} \bmod n$$

$$= m \times (m^{\varphi(n)})^k \bmod n$$

$$= m \times 1^k \quad \leftarrow \text{欧拉定理: } m^{\varphi(n)} \equiv 1 \pmod{n}$$

$$= m \quad \checkmark$$

看到欧拉定理在哪用上了吗？就是那一步 $m^{\varphi(n)} \equiv 1$ 。

(严格来说还需要 m 与 n 互素这个前提；即使不互素，结论也成立，课件留了"思考：若 m 和 n 不互素？"这个问题。)

为什么从公钥推不出私钥？

这是 RSA 安全性的核心。看这条链：

公开的信息： e 和 n

↓ 想算私钥 d ？

$$d = e^{-1} \bmod \varphi(n) \quad \leftarrow \text{必须先知道 } \varphi(n)$$

↓ 想算 $\varphi(n)$ ？

$$\varphi(n) = (p-1)(q-1) \quad \leftarrow \text{必须先知道 } p \text{ 和 } q$$

↓ 想算 p 和 q ？

$$n = p \times q \quad \leftarrow \text{必须对 } n \text{ 做因子分解 (这就是难题)}$$

课件原话："已知 e 和 n ，要得到 d ，需知道 $\varphi(n)$ ；要得到 $\varphi(n)$ 则需要知道 n 的因子分解。当 n 很大时，这是一个难题。"

从安全性考虑，要求 n 至少 1024 比特。

回到 1.4 节那个"两袋沙子"的比喻

现在你应该能完整理解这句话了：

- 算 $n = p \times q$ ：很快（小学生都会）
- 从 n 还原出 p 和 q ：极难（ n 有 1024 位时，用最好的算法也算不完）

RSA 的全部安全性，就架在这个不对称上。

RSA 会被怎么攻击？

① 针对因子分解：Pollard's Rho 算法、Pollard's P-1 算法——都是想办法把 n 分解开。

② 针对协议的攻击（这个更危险，因为是“用法”出错）

- **共用模数 n** ：如果同一组人用同一个 n ，但 e 不同，攻击者可以用扩展 Euclid 求出 $re_1 + se_2 = 1$ ，然后 $(c_1)^r \times (c_2)^s = m \bmod n$ ，直接算出明文。
- **密钥使用不当**：混淆了加密密钥和签名密钥的使用。

结论：算法再强，用法错了也白搭。（这就是为什么课件的思考题问“既然 RSA 签名算法存在指数运算攻击，为什么仍然应用广泛？”）

2.4.1 公钥密码需要的数学工具

离散对数问题（DLP）

定义：若 $g^t \equiv x \pmod{n}$ ，则 t 称为 x 以 g 为底模 n 的**离散对数**。

举例： $3^2 \equiv 9 \pmod{13}$ ，所以 $\log_3 9 \equiv 2$

难题在哪：

- **易**：已知 g 、 t 、 n ，算 $x = g^t \bmod n$ （用快速指数算法，很快）
- **难**：已知 g 、 x 、 n ，反求 t （当 n 很大时，是个难题）

课件说：“很多密码技术的安全性建立在离散对数难题上，如 ELGamal、ECC 等。”

两个求离散对数的算法（知道名字和思路就行）

- **Baby-step Giant-step**：把求解空间“两头夹击”——先算一批 $b^0, b^1, \dots, b^{m-1}$ 存成表，再从另一边大步跳，跳到表里就找到了。
- **Pollard's Rho**：构造一个“看起来随机”的迭代，让它像 ρ 一样绕圈，一旦两次走到同一个点，就能解出答案。（这两个都是“比穷举快、但仍然很慢”的算法，所以离散对数依然是个难题。）

中国剩余定理 CRT（这个很实用）

来历：又称孙子定理，最早见于《孙子算经》的“物不知数”问题——“今有物不知其数，三三数之剩二，五五数之剩三，七七数之剩二，问物几何？”

问题：已知几个“余数方程”，求原来的数。

$$x \equiv a_1 \pmod{m_1}, x \equiv a_2 \pmod{m_2}, \dots \text{（要求各 } m_i \text{ 两两互素）}$$

公式：令 $M = m_1 m_2 \dots m_n$ ， $W_i = M/m_i$ ， $T_i = W_i^{-1} \bmod m_i$

$$x \equiv a_1 T_1 W_1 + \dots + a_n T_n W_n \pmod{M}$$

动手做：解“物不知数”

$$x \equiv 2 \pmod{3}, x \equiv 3 \pmod{5}, x \equiv 2 \pmod{7}$$

$$M = 3 \times 5 \times 7 = 105$$

$$W_1 = 105/3 = 35 \quad W_2 = 105/5 = 21 \quad W_3 = 105/7 = 15$$

$$T_1 = 35^{-1} \bmod 3 = (35 \bmod 3)^{-1} = 2^{-1} \bmod 3 = \mathbf{2} \quad (\text{因为 } 2 \times 2 = 4 \equiv 1)$$

$$T_2 = 21^{-1} \bmod 5 = (21 \bmod 5)^{-1} = 1^{-1} = \mathbf{1}$$

$$T_3 = 15^{-1} \bmod 7 = (15 \bmod 7)^{-1} = 1^{-1} = \mathbf{1}$$

$$x \equiv 2 \times 35 \times 2 + 3 \times 21 \times 1 + 2 \times 15 \times 1$$

$$= 140 + 63 + 30 = \mathbf{233}$$

$$233 \bmod 105 = 233 - 210 = \mathbf{23}$$

答：23（也就是"23 个"）

CRT 有两个重要用途

① **简化大数运算**：课件举例——计算 $47^{317} \bmod 713$ 。

因为 $713 = 23 \times 31$ ，可以拆成两个小问题：

$$x \equiv 47^{317} \pmod{23} \equiv 13^{17} \pmod{23} \equiv 6 \pmod{23}$$

$$x \equiv 47^{317} \pmod{31} \equiv 8^{17} \pmod{31} \equiv 2 \pmod{31}$$

再用 CRT 合起来 $\Rightarrow x \equiv \mathbf{374} \pmod{713}$

为什么要拆？ 因为模数小了，每次运算的数字都小很多，算起来快。

② **构造密钥分发协议**：第 4.3 节会讲怎么用 CRT 把密钥分发的代价从 $O(k)$ 降到 $O(1)$ 。

▼ 自测：RSA 里， $p=3$ 、 $q=5$ 、 $e=3$ ，求私钥 d ，并加密 $m=7$

- ① $n = 3 \times 5 = 15$
- ② $\varphi(n) = (3-1)(5-1) = 8$
- ③ 检查 $\gcd(3, 8) = 1 \checkmark$
- ④ $d = 3^{-1} \bmod 8$: $3 \times 3 = 9 \equiv 1 \pmod{8}$ ，所以 $d = 3$
- ⑤ 加密: $c = 7^3 \bmod 15 = 343 \bmod 15 = 13$
- ⑥ 验算解密: $13^3 \bmod 15 = 2197 \bmod 15 = 7 \checkmark$

公钥 (3, 15)，私钥 3。

思考：为什么 d 刚好等于 e ？因为 $\varphi(n) = 8$ ，而 3 自逆 ($3 \times 3 = 9 \equiv 1 \pmod{8}$)。这只是个小巧合，真实场景里 e 和 d 通常相差很大。

2.5 数字签名

为什么需要它

手写签名有三个作用：**证明是你签的、签完不能改、签完不能赖。**

网络世界也需要这三样，而且要更强——因为电子文件复制粘贴太容易了。

课件列的性质是：**可信，防伪造，防篡改，防抵赖，不可重用。**

RSA 签名的过程

【密钥生成】和 RSA 加密一样：公钥 (e, n) ，私钥 (d, p, q)

【签名】 $s = m^d \bmod n$ ← 用私钥

【验证】验证 $s^e \bmod n$ 是否等于 m ← 用公钥

对比一下 RSA 加密：

	加密	签名
第一步用什么	公钥 e	私钥 d
第二步用什么	私钥 d	公钥 e
用途	保密	证明身份

看出规律了吗——**正好反过来**。这就是为什么有人把签名叫“用私钥加密”。

RSA 签名会被怎么攻击？

攻击 I：利用指数运算的特点（骗取签名）

场景：Alice 想骗 Bob 对“借据 50 万”签字，但她知道 Bob 不会同意。

于是她换个办法：

- ① Alice 自己选一个 k ，计算 $y = k^e \bmod n$
- ② 构造 $m' = y \times m \bmod n$ (m 是那份借据)
- ③ 骗 Bob 对 m' 签名： $s(m') = (m')^d = (y \cdot m)^d = k \cdot m^d \bmod n$
- ④ Alice 用 k^{-1} 处理一下： $s(m') \times k^{-1} = k \cdot m^d \times k^{-1} = m^d = s(m)$

→ Alice 拿到了 Bob 对借据 m 的合法签名！

这个攻击的妙处：Bob 签的是 m' （一个“看起来无害”的东西），但他实际上等于签了 m 。

这不就是“盲签名”的思想吗？（第 5.2 节会专门讲——同一个数学技巧，用作攻击是“骗取签名”，用作设计就是“盲签名服务”。）

攻击 II：密钥使用不当

场景：Carol 想知道 Alice 发给 Bob 的秘密消息。

- ① Carol 选一个 r ，计算 $x = r^e \bmod n$ ， $y = x \cdot c \bmod n$
- ② 骗 Bob 对 y 签名： $u = y^d \bmod n$
- ③ Carol 计算： $r^{-1} \cdot u = r^{-1} \cdot y^d = r^{-1} \cdot (x \cdot c)^d = r^{-1} \cdot r \cdot m = m$

→ 秘密泄露

防范措施：加密密钥对 $\neq$ 签名密钥对。别用同一对密钥既加密又签名。

DSA 签名算法

DSA（Digital Signature Algorithm）是 ElGamal 和 Schnorr 签名方案的变形，1994 年被 NIST 采纳为数字签名标准 DSS，使用 SHA 作为散列函数。

【密钥参数】

p : L 位素数， $512 \leq L \leq 1024$ ，要求 $64 \mid L$

q : 160 位素数，且 $q \mid p-1$

$g: g = h^{((p-1)/q)} \bmod p$
私钥 d , 公钥 $e = g^d \bmod p$

【签名】选随机数 k

$r = (g^k \bmod p) \bmod q$

$s = (k^{-1}(H(m) + d \cdot r)) \bmod q$

【验证】

$w = s^{-1} \bmod q$

$u1 = (H(m) \times w) \bmod q$

$u2 = (r \times w) \bmod q$

验证: $r == ((g^{u1} \cdot e^{u2}) \bmod p) \bmod q$

DSA 的一个致命细节: k 不能重复

如果 Eve 拿到了用同一个 k 签名的两个消息:

$$s1 = k^{-1}(H(m1) + d \cdot r)$$

$$s2 = k^{-1}(H(m2) + d \cdot r)$$

两式相除 (或者交叉相乘):

$$s1 \cdot (H(m2) + d \cdot r) = s2 \cdot (H(m1) + d \cdot r)$$

→ 除了 d 之外全是已知量 → 可以解出私钥 d

结论: k 的随机产生很关键。(这个坑在现实中真的出过事——2010 年索尼的 PS3 就是因为 ECDSA 的 k 用了固定值而被破解。)

离散对数签名方案 (通用框架)

DSA 只是"离散对数签名方案"的一种。更一般的框架是:

选大素数 p 和 q , $q \mid p-1$

选 g 满足 $1 < g < p$ 且 $g^q = 1 \bmod p$

私钥 d , 公钥 $e = g^d \bmod p$

【签名】选随机数 k , $r = g^k \bmod p$, 取 $r' = r \bmod q$

签名等式: $a \cdot k = b + c \cdot d \pmod{q}$

【验证】

验证等式: $r'^a \bmod p$ 是否等于 $g^b \cdot e^c \bmod p$

其中 a、b、c 是 r'、m、s 的各种组合 $\Rightarrow$ 能产生**多达 13000 种变型**

所以 DSA、ElGamal 签名、Schnorr 签名，都是这个框架下的特例。

▼ 自测：RSA 加密和 RSA 签名，公钥私钥的用法有什么区别？

	RSA 加密（保密通信）	RSA 签名
第一步	用对方的公钥加密： $c = m^e \bmod n$	用自己的私钥签名： $s = m^d \bmod n$
第二步	用自己的私钥解密： $m = c^d \bmod n$	用自己的公钥验证： $s^e \bmod n == m$
目的	只有对方能看懂	只有我能生成、谁都能验证

口诀：

保密：我给别人发 $\rightarrow$ 用**别人的公钥**（只有他的私钥能开）

签名：我证明是我 $\rightarrow$ 用**我的私钥**（只有我能做）

后果：一个用公钥牵头的方向是“保密”，用私钥牵头的方向是“证明身份”。这两种用途**应该用不同的密钥对**——否则会出现前面那个“密钥使用不当”的攻击。

2.6 国密算法：SM 系列

背景

由国家密码局发布，主要有 SM 系列。算法公开的有 SM2、SM3、SM4，密钥长度和分组长度均为 128 位。

国密	对应国际算法	特点
SM1	AES	对称密码（SM7），芯片调用
SM2	ECC256	公钥密码，基于 ECC
SM3	SHA256	数字摘要算法
SM4	DES	对合运算，非平衡 Feistel 网络
SM9	ECC256	基于对的身份标识密码算法

一句话记住每个 SM

- **SM1**：对称加密，**芯片里用的**（不公开，靠硬件保密）
- **SM2**：公开密钥加密 + 签名，**对标 RSA/ECC**
- **SM3**：摘要算法，**对标 SHA-256**
- **SM4**：分组加密，**对标 AES/DES**
- **SM9**：用你的身份（比如邮箱）直接当公钥，不需要证书

SM4 与 SM3 的关键参数

SM4（2006 年公布，无限局域网产品使用）：

- 分组长度和密钥长度都是 128 位，32 轮，子块 32 位
- 轮函数： $X_{i+4} = X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i)$
- 非线性置换：4 个平行的 8 位 S 盒

SM3（2010 年公布，在 SHA-256 基础上改进）：

- 分组长度 512 位，摘要长度 256 位，64 次迭代

SM2（2010 年公布，基于椭圆曲线）：曲线方程 $y^2 = x^3 + ax + b \pmod{p}$

为什么这门课要讲国密

课件在引言里提到：**2019 年 10 月 26 日全国人大常委会通过《密码法》，2020 年 1 月 1 日起施行**，并把“密码安全教育纳入国民教育体系和公务员教育培训体系”。

所以在实际工作（尤其是政务、金融、能源等关键信息基础设施）中，需要用国密算法而不是国际算法。**这既是技术要求，也是合规要求。**

第 3 章

基本密码协议

用一个场景把整章串起来

课件给了一个很实在的场景：

Alice 想租房，在网上看到 Bob 的房源信息，他们想通过网络完成租房协议的签订。

请为他们设计一个安全高效的通信方案。这里冒出了一堆需求：

需求	对应哪个协议
(1) 协议内容保密	保密通信协议 (3.2)
(2) 能防止欺骗	密钥协商协议 (3.3)
(3) 保证签名的性质	身份鉴别协议 (3.5)
(4) 考虑公平性问题	数字签名协议 (3.4)
(5) 不需可信第三方	— (这是个约束条件)
(6) 要求性能最优	秘密共享协议 (3.6)

整章就是在回答这一个问题。带着这个场景看下去，会顺很多。

3.1 协议是什么

大白话

密码算法是"工具"，密码协议是"步骤"。

就像：锤子、钉子、锯子是工具；"做一张桌子"是步骤。

光有工具不会自动产生安全——你得按正确的步骤用。用错步骤，工具再好也白搭。

课件原话："密码协议又称为安全协议，由密码算法设计而成，在网络环境中提供各种安全服务。"

协议怎么分类

按仲裁者分：

- 基于对称密码 → 仲裁协议
- 基于公钥密码 → 裁决协议
- 区别：裁决协议中，仲裁者不直接参与协议执行过程
- 仲裁者：公正的第三方，如公证人、法院、KDC 等

按应用分：保密通信、密钥协商、身份鉴别、数字签名、秘密共享、盲签名、安全计算、匿名通信……（就是后面几章的内容）

协议会被怎么攻击

课件强调：不但能攻击密码算法，还能攻击协议本身。

类型	做法
被动攻击	窃听协议的一部分或全部，目的是获取消息，不影响协议的执行
主动攻击	改变协议以对自己有利，插入、删除、修改消息，破坏通信

骗子

协议中的一方，不遵守协议或者破坏协议

注意第三类：参与者自己使坏。前面讲算法时不需要考虑这个——算法假设双方都诚实地执行。而协议里，你得假设对方可能是骗子。

一个经典例子：共享模数 n 的隐患

RSA 算法本身很安全，但在协议里用错就有隐患。

场景：同一组用户之间共享 RSA 的公共模数 n （但各自 e 不同）。

攻击者拿到 Carol 发给 Alice 和 Bob 的同一条消息 m 的两个密文：

$$c1 = m^{e1} \bmod n$$

$$c2 = m^{e2} \bmod n$$

若 $e1$ 和 $e2$ 互素，存在 r 、 s 使得 $r \cdot e1 + s \cdot e2 = 1$ （扩展 Euclid）

$$\text{则 } (c1)^r \times (c2)^s = m^{(r \cdot e1 + s \cdot e2)} = m^{(1)} = m \bmod n$$

→ 攻击者不需要私钥，直接算出明文！

教训：算法安全 $\neq$ 协议安全。这就是为什么要有“密码协议”这门学问。

3.2 保密通信协议

协议流程（就 4 步）

- ① 通信双方协商密码算法和密钥
- ② 发送者将明文 m 加密成密文 c
- ③ 发送者发送密文消息 c
- ④ 接收者对密文 c 解密获得明文 m

关键问题：第 ① 步怎么做

后面三步没什么好说的，**难点全在第 ① 步——“协商密码算法和密钥”。**

课件用四种方案来回答这个问题，一次比一次好。

四种方案，看它们怎么一步步改进

方案	做法	问题 / 优点
① 古典密码	用简单的代替/换位密码，双方事先约定	简单，适合普通用户；但 需要保密算法和密钥
② 对称密码 + TTP	引入可信第三方 TTP。Alice、Bob 各与 TTP 共享一把密钥，消息经 TTP 中转	快；但 TTP 是瓶颈，安全性依赖 TTP ，可扩展性差
③ 公钥密码	Alice 用 Bob 的公钥加密： $c = E_{Bp}(m)$	不需要事先共享密钥 ；但速度慢，且 依赖公钥的可靠
④ 混合密码	用公钥密码保护和分发 会话密钥 ，用会话密钥（对称）加密消息	既快又安全 ——现实中就是这么用的

第 ④ 种方案：数字信封

这是现实中真正在用的方案，叫**数字信封**。理解它只要两步：

用公钥保护“钥匙”：把对称密钥 k 用 Bob 的公钥加密 $\rightarrow E_{Bp}(k)$

用对称密钥保护“信”：把消息 m 用 k 加密 $\rightarrow E_k(m)$

为什么要这样分工？

- 对称密码**快**，适合加密大量数据（消息）
- 公钥密码**安全但不快**，适合加密少量数据（只是一把密钥）

取两者之长，这就是“混合密码”。

但公钥方案有个大问题：中间人攻击

问题出在哪：Alice 怎么拿到 Bob 的公钥？如果从网上拿，怎么知道那个公钥真的是 Bob 的？

Alice 想拿 Bob 的公钥

↓

黑客截获请求，**把自己的公钥 k_{Mp} 冒充成 Bob 的**发给 Alice

↓

Alice 以为 k_{Mp} 是 Bob 的公钥，用它加密消息 $\rightarrow$ 黑客用自己的私钥解开

↓

黑客再用 Bob 的真公钥加密转发给 Bob

$\rightarrow$ **Alice 和 Bob 全程都不知道有人在中间偷看**

这就是经典的**中间人攻击（MITM）**。对策是**数字证书**（第 7.3 节）——让一个大家都信任的机构来“担保”公钥的身份。

▼ 自测：为什么现实中不直接用公钥密码加密消息，而要用“混合密码”？

因为公钥密码太慢。

课件原话：对称密码“加密速度快”，公钥密码“运算速度慢”。实际差距可能有上千倍。

所以现实中的做法是：

- 用公钥密码协商出一个会话密钥（只加密很短的一串数据）
- 用这个会话密钥 + 对称密码加密真正的消息（快）
- 通信结束就销毁会话密钥

打个比方：公钥密码像“打电话确认保险柜密码”（慢但要安全），对称密码像“用保险柜装货”（快）。你不会把所有货物都塞进电话线里。

顺带：因为会话密钥每次通信都换、用完就丢，所以它相当于“一次一密”的实用版本——安全性比长期密钥好得多。

3.3 密钥协商协议

这一节要解决什么

上一节的“混合密码”提到了“用公钥协商会话密钥”。那具体怎么协商？

课件列了五种方案，其中 **DH 协议是重中之重**——因为它第一次做到了“完全不需要事先共享任何秘密”。

五种方案对比

协议	做法	特点
基于公钥密码	Alice 用 Bob 的公钥加密密钥 k	裁决协议；依赖公钥可靠，怕中间人攻击
基于对称密码	借助 KDC，双方各与 KDC 共享一把密钥	仲裁协议；依赖 KDC，KDC 是瓶颈，可扩展性差
DH 协议	双方各选一个秘密数，交换“算出来的公开值”	不需要事先分配任何密钥；怕中间人攻击
EKE 协议	用共享密钥加密公钥	安全性较好（有验证步骤）
基于双线性对	用双线性映射构造	计算代价低，用于无线网络

DH 协议：魔术般的一步

先讲个“颜色”的比喻

这个比喻能让你秒懂 DH：

1. Alice 和 Bob 面前有一个公共的颜料桶（黄色，谁都看得见）
2. Alice 偷偷往桶里加自己的秘密颜料 A（红色），搅匀后寄给 Bob
3. Bob 偷偷往桶里加自己的秘密颜料 B（蓝色），搅匀后寄回 Alice
4. 两人最后得到的颜色一样（黄+红+蓝），但偷看的人不知道 A 和 B 是什么，所以他调不出这个颜色

神奇之处：整个过程全程公开，但最终结果只有他俩知道。

数学版本：跟课件走

前提：Alice 和 Bob 公开约定一个大素数 p 和一个原根 g 。（注意：这两个数不需要保密，公开都行。）

① Alice 偷偷选一个秘密数 x ，计算 $g^x \bmod p$ ，把这个结果发给 Bob

② Bob 偷偷选一个秘密数 y ，计算 $g^y \bmod p$ ，把这个结果发给 Alice

③ Alice 收到 g^y 后，算 $(g^y)^x = g^{xy} \bmod p$

④ Bob 收到 g^x 后，算 $(g^x)^y = g^{xy} \bmod p$

→ 两人得到同一个值： $g^{xy} \bmod p$ ，这就是共享密钥

课件给的小例子： $p=7, g=2 \rightarrow 3$ ； $p=7, g=3 \rightarrow 6$

为什么偷看的人算不出来？

攻击者能看到的是： p 、 g 、 $g^x \bmod p$ 、 $g^y \bmod p$ 。

他想算出 g^{xy} ，就必须知道 x 或 y 。但从 $g^x \bmod p$ 反推 x ——

这就是离散对数难题。

看，第 2.4 节的数学在这里用上了。

但 DH 挡不住中间人

DH 能防“偷听”，但防不了“插在中间”：

Alice 以为在跟 Bob 协商，实际上：

Alice $\leftrightarrow$ 黑客 $\leftrightarrow$ Bob

Alice 和黑客协商出一个密钥 k_1 ，黑客和 Bob 协商出 k_2 。

Alice 用 k_1 加密，黑客解开、用 k_2 重新加密给 Bob。

双方都以为在跟对方安全通信。

为什么？ 因为 DH 全程没有“验证对方身份”这一步——它只保证“协商出来的密钥是安全的”，但不保证“对面是谁”。

课件作业题就是这个：“分析 Diffie-Hellman 协议的安全性。”

EKE 协议：加一步验证

EKE (Encrypted Key Exchange, 加密密钥交换) 由 Steve Bellovin 和 Michael Merritt 设计。

思想：用双方事先共享的密钥 p 加密随机产生的公钥，然后建立会话密钥 k 。

① Alice $\rightarrow$ Bob: $A, E_p(kA_p)$ (用共享密钥加密自己的公钥)

- ② Bob → Alice: $E_p(E_{Ap}(k))$ (Bob 解密得 k_{Ap} , 选一个会话密钥 k , 用 k_{Ap} 加密后发回)
- ③ Alice → Bob: $E_k(RA)$ (Alice 解出 k , 发一个随机数 RA)
- ④ Bob → Alice: $E_k(RA, RB)$ (Bob 解密验 RA , 再发 RA 和 RB)
- ⑤ Alice → Bob: $E_k(RB)$ (Alice 验 RB)

多出来的那几步是干什么的？就是互相验证——通过交换随机数并验证，证明对方真的知道那个密钥，而不是中间人。

所以课件说："为什么说 EKE 协议的安全性较好？" 答案就是：它既有共享密钥的保障，又有公钥密码的便利，而且加了双向验证。

基于双线性对的协商

双线性映射 $\hat{e}$ 有三个性质：

- 双线性: $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$
- 非退化性: 存在 P, Q 使得 $\hat{e}(P, Q) \neq 1$
- 可计算性: 能高效算出 $\hat{e}(P, Q)$

三方密钥协商一句话解决：

$$k = \hat{e}(P_B, P_C)^a = \hat{e}(P_A, P_C)^b = \hat{e}(P_A, P_B)^c = \hat{e}(G, G)^{abc}$$

三个人各自用自己的私钥和另外两人的公钥，算出来同一个值。DH 想做三方的要很多轮，这里一步就行。

▼ 自测：DH 协议里，双方需要事先共享什么秘密吗？

不需要任何秘密——这正是 DH 最了不起的地方。

他们只需要公开约定两个数：大素数 p 和原根 g 。这两个数广播出去都无所谓。

然后双方各自偷偷选一个秘密数 (x 和 y)，交换计算出来的公开值 (g^x 和 g^y)，最后各自算出 g^{xy} 。

攻击者能看到全部通信内容，但还是算不出 g^{xy} ——因为那需要解离散对数问题。

这就是为什么 DH 是"密钥协商"而不是"密钥分发"：不需要有人"发送"密钥，而是双方"一起算"出同一个密钥。

3.4 数字签名协议

协议思想（一句话）

密文防止篡改，密钥代表身份

- 签名过程：把消息 m 用密钥加密得到 $s = E_k(m)$
- 验证过程：把 s 用密钥解密，和 m 对比

四种协议对比（背这张表）

协议	协议特点
----	------

① 基于对称密码	需要可信第三方，小规模场景； (单纯用它不能保证签名的性质)
② 基于公钥密码	利用 hash 函数提高效率，扩展性好
③ 带加密的数字签名	只有指定的授权人才能验证签名，保密
④ 签名者参与验证	签名需要在线验证，可防止复制和重用

为什么对称密码做签名不行？（这是重点）

课件用"Alice 向 Bob 借 50 万"来说明。

协议 1 的做法：Alice 和 Bob 共享密钥 k ，Alice 用 k 对借据加密作为签名。

问题在哪？ 因为 Bob 也知道 k 。

那意味着——Bob 可以自己伪造一份"Alice 的签名"。到时候 Alice 说"我没签"，Bob 拿出的签名完全看不出是谁做的。

缺了什么性质？ 缺"不可伪造"和"不可抵赖"。

怎么补救？ 引入可信第三方 TTP。TTP 与 Alice 共享 k_A 、与 Bob 共享 k_B ，由 TTP 在中间做证明（协议 1+）。

代价：需要仲裁人参与，扩展性差，只适合小规模环境。

协议 2：公钥密码做签名（推荐做法）

用 Alice 自己的私钥 k_{Av} 签名，公钥 k_{Ap} 验证：

Alice: $m \xrightarrow{E_{k_{Av}}(m)} c \xrightarrow{D_{k_{Ap}}(c)} m$: Bob

四条性质全都满足：

- ① 签名可信、不可伪造 → 由 Alice 的私钥保证（只有她有）
- ② 签名不可重用 → 签名是文件的函数（换个文件签名就不一样）
- ③ 签名文件不能改变 → 由私钥保证（改了就对不上）
- ④ 签名不可抵赖 → 只有 Alice 拥有私钥

缺点：运算速度慢，效率低（因为要对整段消息做 RSA 运算）。

协议 2+：加个 hash，效率立刻上去

- ① 签名方：对 m 算摘要 $m1 = H(m)$ （比如 sha256）
- ② 用私钥签名： $s = S_{k_{Av}}(m1)$ ← 只对 256 位摘要做运算，快
- ③ 发送 $m \parallel s$
- ④ 验证方：用公钥解出 $m2 = V_{k_{Ap}}(s)$
- ⑤ 再算一遍 $H(m)$ ，和 $m2$ 比较： $m1 == m2$?

这就是课件里那个 **sha256RSA** 的含义——先用 SHA256 摘要，再用 RSA 签名。

为什么安全？ 因为散列函数有抗碰撞性：攻击者找不到另一条消息产生相同摘要，所以他改不了 m 。

协议 3：带加密的数字签名（保密 + 签名）

就是把“保密通信”和“数字签名”拧在一起：

保密通信： $m \xrightarrow{E_{Bp}} c \xrightarrow{D_{Bv}} m$ （用对方公钥 / 自己私钥）

数字签名： $m \xrightarrow{S_{Av}} m||s \xrightarrow{V_{Ap}} m$ （用自己的私钥 / 公钥）

合起来：

签名方： $m \xrightarrow{S_{Av}} s \xrightarrow{E_{Bp}} c$ （先签后加密）

验证方： $c \xrightarrow{D_{Bv}} s \xrightarrow{V_{Ap}} m$

特点： 只有指定的授权人才能验证签名（因为只有他有对应的私钥解密）。

课件留了一道设计题：“**利用混合密码协议和哈希+公钥的数字签名协议优化带加密的数字签名协议。**” 思路就是：把公钥运算换成“公钥协商 + 对称加密”，把直接签名换成“hash + 签名”。

协议 4：签名者参与验证（Chaum 的方案）

目的： 让签名“必须在线验证”，从而防止复制和重用。

密钥关系：私钥 d ，公钥 $e = g^d \bmod p$

签名： $s = m^d \bmod p$

验证时（这一步需要签名方参与）：

- ① 验证方选 a, b ，计算 $x = s^a \cdot e^b$ ，发给签名方
- ② 签名方计算 $z = d^{-1} \bmod p-1$ ，返回 $y = x^z \bmod p$
- ③ 验证方检查： $y = m^a \cdot g^b \bmod p$?

为什么这样能防重用？ 因为每次验证都要签名方**实时参与**（他就是那个知道 d 的人）。签名被复制了也没用——离开签名方就没法完成验证。

两个有意思的协议应用

① 多个参与者签名（RSA 推广）

设 $n = pq$ ，取 $k_1 \times k_2 \times \dots \times k_m \equiv 1 \pmod{\phi(n)}$ （类比 $ed \equiv 1$ ）

则用其中若干个 k_i 加密的消息，可以用剩下的 k_j 解密

例子：Alice 有 k_1 ，Bob 有 k_2 、 k_3 。Alice 算 $m' = m^{k_1}$ 发给 Bob，Bob 用 $k_2 k_3$ 就能恢复 m ，还能把自己的签名加上（ $m'' = m^{k_1 k_2}$ ）。

② 阉下信道——这个很妙

场景：Alice 和 Bob 在监控下传消息。监控者也能“验证签名”，但他发现不了里面藏的秘密。

基于 ElGamal：Alice 私钥 d ，公钥 $e = g^d \bmod p$

签名： $r = g^k \bmod p$ ， $m = (dr + sk) \bmod p-1$

监控者验证： $e^r \cdot r^s = g^m \bmod p$ ✓（看起来正常）

Bob 额外算一步： $k = s^{-1}(m - dr) \bmod p-1$ ← 把 k 提取出来！

关键：那个 k （本来是签名用的随机数）被 Bob 当成了秘密信道来传信息。监控者看到的一切都“合法”。

▼ 自测：为什么基于对称密码的签名协议“不能保证签名的性质”？

因为双方共享同一把密钥，所以谁都能生成“对方的签名”。

具体说：Alice 和 Bob 共享密钥 k 。Alice 用 k 加密借据当签名。问题是 Bob 也有 k ——他完全可以自己加密一份借据，声称是 Alice 签的。

这导致两个问题：

- **可伪造：**Bob 能伪造 Alice 的签名
- **可抵赖：**Alice 也能说“那是 Bob 自己做的”

对比公钥签名：私钥只有 Alice 一个人有，所以别人做不出来——不可伪造；签名只有 Alice 能做——不可抵赖。这就是为什么公钥签名是主流。

（补救办法：引入可信第三方 TTP。但代价是“需要仲裁人参与，扩展性差，只适合小规模环境”。）

3.5 身份鉴别协议

先讲个古人的办法

课件用了几句暗号开场：

“天王盖地虎，宝塔镇河妖。”

“长江长江，我是黄河。”

“地瓜地瓜，我是土豆。”

这些就是最朴素的身份鉴别——对暗号。

身份鉴别的定义：确认实体是它所声明的，而非假冒。

单向 hash 链 (SKEY 的基础)

回忆散列函数的性质：**单向性**—— $x \rightarrow h(x)$ 易， $h(x) \rightarrow x$ 难。

把很多次哈希串起来，就是**单向 hash 链**：

$$r \rightarrow X_0 \rightarrow X_1 \rightarrow X_2 \rightarrow \dots \rightarrow X_{n-1} \rightarrow X_n$$

其中 $x_{i+1} = h(x_i)$ 。所以**沿着箭头走容易，倒着走很难**。

最简单的口令鉴别：PAP

PAP 协议的三个安全问题

PAP (Password Authentication Protocol) 的做法极其简单：**直接发"用户名 + 口令"**。

Alice — 用户名, 口令 —→ 公共网络 —→ 服务器

这有三个致命问题：

问题	攻击者怎么做	怎么防
① 攻击数据库	侵入服务器数据库，把口令全偷走	存 hash 值，不存明文口令
② 网络窃听	在公共网络里直接听到口令	加密后再传
③ 重放攻击	截获密文，原样再发一遍冒充身份	动态口令、SKEY 协议

逐个说清这三种防御

① 为什么存 hash 就行？

因为散列函数是单向的——数据库被偷了，攻击者拿到的是 $h(x)$ ，**反推不出原始口令 x** 。（登录时：服务器算 $h(\text{用户输入})$ 再和存的 $h(x)$ 比较，相等就通过。）

② 为什么加密就行？

窃听器拿到的是密文，看不懂。

③ 重放攻击为什么难防？

这个最麻烦。因为攻击者不需要懂内容——**他只要把截获的那串密文原样再发一次就行**。服务器收到后解密、验证、通过……攻击者就冒充成功了。

所以要引入**"动态"**因素：每次登录用的凭证都不一样，让截获的东西**用不了第二次**。

SKEY 协议：用一条哈希链来防重放

准备阶段：服务器（课件里用的是"银行"）根据一个随机数 r 算出一条链：

$$x_0 = h(r)$$

$$x_1 = h(x_0)$$

$$x_2 = h(x_1)$$

...

$$x_{100} = h(x_{99})$$

服务器只保存 x_{100} ，把 $x_0 \sim x_{99}$ 全部交给 Alice。

登录时（注意：倒着用！）：

第 1 次登录：Alice 发 x_{99}

服务器算 $h(x_{99})$ ，看是否等于保存的 $x_{100} \rightarrow$ 相等，通过

然后把 x_{99} 存下来，替换掉 x_{100}

第 2 次登录：Alice 发 x_{98}

服务器算 $h(x_{98})$ ，看是否等于保存的 $x_{99} \rightarrow$ 相等，通过

保存 x_{98}

...依此类推

为什么这样就能防重放？

因为每个数只能用一次，而且倒着用。

- 攻击者截获了 x_{99} ，想重放？不行——服务器已经把它存起来当“当前值”了，下次期待的是 x_{98} 。
- 攻击者想伪造？他需要算出 x_{98} ，但 $x_{98} = ?$ 他只知道 $x_{99} = h(x_{98})$ ，而散列函数是单向的，反推不出来。

两个好处（课件原话）：

1. 由于每个数只用一次，因此可以抵抗重放攻击
2. 服务器只保存一个最新散列值，节约存储开销（不用存一张大表）

课件作业题（很有意思）

"SKEY 协议中单向 hash 链的使用是反向的，如果采用正向使用是否可行，安全性如何？"

想一想：正向使用意味着 Alice 依次发 $x_1, x_2, x_3, \dots$ ，服务器要验证的话，得算 $h(x_1)$ 看是否等于 x_2 ——但验证需要保存下一个值。

问题在于：如果攻击者截获了 x_{99} ，他能算出 $x_{100} = h(x_{99})$ ，就能冒充 Alice 发送下一次的凭证！（因为是正向的，前进容易。）

所以正向使用不安全——恰恰是因为散列函数的“正向容易”这个性质。反向用才有意义：反向困难，攻击者算不出来。

挑战应答协议

用对称密码（前提：有共享密钥 k ）

① Bob → Alice: R_A (发一个随机数“挑战”)

② Alice 选随机数 R_B , 计算 $H_k(R_A, R_B, B) \rightarrow \text{Bob}$

③ Bob 计算并比较 R_B 、 $H_k(R_A, R_B, B)$

④ Bob → Alice: $H_k(R_B, A)$

⑤ Alice 计算并比较 $H_k(R_B, A)$

思想：根据共享密钥 k 确认身份。只有知道 k 的人才能算出正确的 $H_k(\dots)$ 。

为什么用随机数？因为**每次的 R_A 、 R_B 都不一样**，所以哈希结果每次都不同——截获了也没法重放。

用公钥密码（前提：有私钥 k_v ，公钥 k_p 公开）

Bob → Alice: 请求 + 一个随机数 r

Alice → Bob: $E_{A_v}(r)$ (用自己的**私钥**加密 r)

Bob 在数据库里查 Alice 的公钥 k_{A_p} ，解密验证

思想：拥有与公钥 k_p 对应的私钥 k_v 的人，即可确认身份。

为什么用随机数 r ？同样是为了防重放——每次的 r 都不同。

▼ 自测：为什么 SKEY 能同时解决“重放攻击”和“存储开销”两个问题？

抗重放：因为哈希链是**倒着用的**，每个数只用一次。攻击者截获了第 n 次的凭证，第 $n+1$ 次服务器期待的是另一个值，用不上。

省存储：普通做法需要服务器保存一张“用户 → 口令哈希”的大表；SKEY 只需要给每个用户存一个数（当前最新的链上值），用完就换成下一个。

而且它连“用户口令被偷”的风险都降低了——因为传输的不是口令本身，而是链上的一次性值，就算被截获了也不能重用。

这就是为什么课件在小结里把 SKEY 单独列出来：“**动态口令**”这个思路的代表。

3.6 秘密共享协议

三个很实在的场景

① **核按钮手提箱：**总统、国防部长、参谋总长中有**两个被按下按钮**，俄军战略部门就会收到核打击代码。

→ 如何实现“必须两人”这个限制？

② **航天测控网**横跨全球，密钥若集中存放一旦被敌方获取将危及航天器安全。

→ 怎么做到“分散存放”？

③ **密钥托管**：1993 年颁布的 EES 标准要求"加密系统必须让法律执行部门能方便地获得明文"，否则不允许推广。
→ **争议**：政府主张"托管一把密钥用于反恐"（国家安全）；民间质疑"后门大开侵犯隐私"（公民隐私）。

技术上的答案都是同一个：**门限分割 + 托管备案**（课件提到用 (3,5 门限) + (2,5 门限)）。

三个概念（特别容易混，先分清）

概念	什么意思
秘密共享	秘密分成 n 份， 足够多的 k 个成员 ($k \leq n$) 达成一致就能共享
秘密分割	秘密分成 n 份， 必须 n 份全部收齐 才能恢复
门限方案	分成 n 份， k 份（或更多）就能恢复 ； $k-1$ 份（或更少）绝对不能恢复

关键区别：秘密分割要求"全都要"；门限方案要求"够数就行"。

最简单的两种分割方法（先看思路）

① 简单分割——物理上切开

分割前：11011920161008610000

分割后：1101 1920 1610 0861 0000 （五段各拿一份）

问题：每份里直接含有明文片段，安全性差。拿到任意一段都能猜出信息。

② 加密分割——用异或藏起来

选 4 个随机数 $k_1 \sim k_4$ ，计算 $c = m \oplus k_1 \oplus k_2 \oplus k_3 \oplus k_4$

分出去五份： c, k_1, k_2, k_3, k_4

恢复时全部异或起来：

$c \oplus k_1 \oplus k_2 \oplus k_3 \oplus k_4 = m \oplus k_1 \oplus k_2 \oplus k_3 \oplus k_4 \oplus k_1 \oplus k_2 \oplus k_3 \oplus k_4 = m$

好处：明文被隐藏了。单独任何一份都看不出信息。

但还是不够好：必须收齐全部五份才能恢复。有没有"凑够几份就行"的办法？

Shamir 门限方案：用一条多项式藏秘密

核心思想（这个想通了，方案就懂了）

关键洞察：

- 1 个点确定不了直线（有无数条直线过这个点）
- 2 个点可以唯一确定一条直线
- 3 个点可以唯一确定一条抛物线（2 次曲线）
- k 个点可以唯一确定一条 $k-1$ 次曲线

那如果我们把秘密藏在这条曲线的某个位置上呢？

方案

【分割】

- ① 构造 $k-1$ 次多项式： $f(x) = m + a_1x + a_2x^2 + \dots + a_{k-1}x^{k-1} \pmod{p}$
(把秘密 m 放在常数项的位置，其余系数随便选)
- ② 计算 $D_1 = f(1)$, $D_2 = f(2)$, ..., $D_n = f(n)$
- ③ 把 $D_1 \sim D_n$ 分给 n 个人

【恢复】

- ① 至少收集 k 个份额
- ② 用拉格朗日插值还原出 $f(x)$
- ③ 秘密 $m = f(0)$

为什么 $m = f(0)$? 把 $x = 0$ 代进多项式： $f(0) = m + a_1 \cdot 0 + a_2 \cdot 0 + \dots = m \checkmark$

动手做：跟课件算一个 (3,5) 门限方案

课件用的多项式是： $f(x) = 13 + 10x + 2x^2 \pmod{17}$

(3 个份额能恢复，所以是 2 次多项式；5 是分给 5 个人。)

计算 5 个份额：

$$D_1 = f(1) = 13 + 10 + 2 = 25, 25 \bmod 17 = 8$$

$$D_2 = f(2) = 13 + 20 + 8 = 41, 41 \bmod 17 = 7$$

$$D_3 = f(3) = 13 + 30 + 18 = 61, 61 \bmod 17 = 10$$

$$D_4 = f(4) = 13 + 40 + 32 = 85, 85 \bmod 17 = 0$$

$$D_5 = f(5) = 13 + 50 + 50 = 113, 113 \bmod 17 = 11$$

把 8、7、10、0、11 这五个份额分给五个人。

恢复：拿其中 3 个（比如 $D_1=8$, $D_3=10$, $D_5=11$ ）

设 $f(x) = m + a_1x + a_2x^2$ ，代入三个点：

$$8 = m + a_1 + a_2 \quad (x=1)$$

$$10 = m + 3a_1 + 9a_2 \quad (x=3)$$

$$11 = m + 5a_1 + 25a_2 \quad (x=5)$$

三个方程解三个未知数，得到 $m = 13 \checkmark$

(也可以直接用拉格朗日插值算出 $f(0) = 13$ 。)

验证：13 正好是多项式 $f(x) = 13 + 10x + 2x^2$ 的常数项 $\checkmark$

为什么少于 k 份就恢复不出来？

因为 $k-1$ 个点确定不了 $k-1$ 次曲线。

比如 3 个份额的门限方案，2 个点确定不了 2 次曲线——有**无穷多条**抛物线经过这两个点。而每条抛物线在 $x = 0$ 处的值都不一样，你根本不知道秘密是哪个。

这就是门限方案安全性的数学基础。

Asmuth-Bloom 门限方案：用中国剩余定理

思路与课件算例

选 n 个两两互素的数 $m_1 < m_2 < \dots < m_n$ ，秘密 x 发布时就分发 $a_i = x \bmod m_i$ 。

恢复时用 **CRT**（中国剩余定理）解同余方程组。

课件算例： $m_1=9, m_2=11, m_3=13, m_4=14$ ，取 $k=3, n=4$ ，秘密 $x = 500$

$$a_1 = 500 \bmod 9 = 5 \quad (9 \times 55 = 495, 500 - 495 = 5)$$

$$a_2 = 500 \bmod 11 = 5 \quad (11 \times 45 = 495)$$

$$a_3 = 500 \bmod 13 = 6 \quad (13 \times 38 = 494)$$

$$a_4 = 500 \bmod 14 = 10 \quad (14 \times 35 = 490)$$

得到 (3,4) 门限方案：{(5,9) (5,11) (6,13) (10,14)}

动手做：拿 3 份恢复（够 $k=3$ ）

取 (5,9)、(5,11)、(6,13)：

$$x \equiv 5 \pmod{9}$$

$$x \equiv 5 \pmod{11}$$

$$x \equiv 6 \pmod{13}$$

用 CRT 解得 $x \equiv 500 \pmod{1287} \checkmark$ ($1287 = 9 \times 11 \times 13$)

动手做：只拿 2 份（不够）

取 (5,9)、(6,13)：

$$x \equiv 5 \pmod{9}$$

$$x \equiv 6 \pmod{13}$$

$$\text{CRT 解得 } x \equiv 32 \pmod{117}$$

但 $117 = 9 \times 13$ ，而 x 的真实范围是 $182 < x < 1287$ 。

所以满足条件的 x 可能是：266, 383, 500, 617, 734, ...

无法唯一确定！这就保证了安全性。

为什么“够 k 份就能恢复、不够就不行”？

课件的可解性分析用了两个数字：

- $S_k = k$ 个最小 m_i 的乘积（上界）
- $L_{k-1} = k-1$ 个最大 m_i 的乘积（下界）

方案设计时保证秘密 x 满足：

$$M_{t < k} \leq L_{k-1} < x < S_k \leq M_{t \geq k}$$

翻译：

- 拿到 $\geq k$ 份时，CRT 的解空间 M_t 大到能包住 $x \rightarrow$ 唯一确定
- 拿到 $< k$ 份时，解空间 M_t 小到装不下 $x \rightarrow$ 只能缩小范围，不能确定

（课件算例里： $182 < 500 < 1287$ ，而 2 份的解空间只有 117，装不下 500。）

DH 密钥托管协议

目的：公钥的托管，可靠性验证。

- ① Alice 选 5 个比 $p-1$ 小的整数 $S_1 \sim S_5$
- ② 她的私钥 $S = S_1 + S_2 + S_3 + S_4 + S_5$ ，公钥 $t = g^S \bmod p$
- ③ 计算 $t_i = g^{S_i} \bmod p$ ，把 (S_i, t_i) 分别交给 5 个托管人
- ④ 托管人对 t_i 签名后保存
- ⑤ Bob 验证： $t = t_1 \times t_2 \times t_3 \times t_4 \times t_5 \bmod p$ ？

（因为 $g^{(S_1+S_2+\dots+S_5)} = g^{S_1} \cdot g^{S_2} \cdot \dots$ ）

- ⑥ 成立则认可该公钥

妙在哪：私钥被“加法拆分”给了 5 个人，但公钥可以验证完整性——不用把 5 份凑齐就能确认“公钥没被篡改”。

四种方法对比（课件小结）

协议	特点
简单分割	秘密份额中包含明文，安全性差
加密分割	明文被隐藏，安全性好
门限方案	部分秘密份额可恢复出明文
DH 方案	公钥的托管，可靠性验证

秘密共享的作用：保证重要信息和秘密数据的安全。

▼ 自测：为什么 (3,5) 门限方案里，拿到 2 份恢复不出秘密？

因为 2 个点确定不了 2 次多项式。

(3,5) 门限用的是 $k-1 = 2$ 次多项式。2 个点能唯一确定一条直线（1 次），但确定不了抛物线（2 次）——有无数条抛物线经过这两个点。

具体地说：假设你拿到的是 $x=1$ 和 $x=3$ 处的值。任何一条形如 $f(x) = A + Bx + Cx^2$ 的曲线，只要满足 $f(1)$ 和 $f(3)$ 等于你手上的值，都是“可能答案”。而每条曲线在 $x=0$ 处的值（也就是秘密）都不同。

所以：

- 拿到 k 份 → 唯一确定曲线 → 唯一确定 $f(0)$
- 拿到 $k-1$ 份 → 有无穷多条可能的曲线 → 秘密可能是任何值

注意：这是信息论层面的安全——不是“计算上很难”，而是“数学上就没有唯一答案”。

第 4 章

中级密码协议

这一章要解决什么

第 3 章的协议都建立在"事先有某个密钥"或者"能拿到对方的公钥"之上。这一章继续追问几个更刁钻的问题：

- 不协商密钥，能不能保密通信？（三次传输）
- 不想让对方知道我在算你的数据，但又要一起算出结果？（安全多方计算）
- 广播环境下，怎么把消息只发给其中几个人，代价还最小？（密钥分发）
- 怎么承诺一件事，保证后来不能反悔、对方现在也不能偷看？（位承诺）

4.1 三次传输协议

Shamir 三次传输：不协商密钥也能通信

为什么需要它

前面所有的保密通信协议，第一步都是"协商密钥"。那能不能跳掉这一步？

Shamir 说：可以。前提是——加密算法满足交换律。

什么叫"加密满足交换律"

意思是：先加密 A 再加密 B，和先加密 B 再加密 A，结果一样。

写成公式： $E_B(E_A(m)) = E_A(E_B(m))$

就像上锁：你先挂一把锁，我再挂一把锁；开锁的时候，谁先开无所谓。

协议流程（三次传输，名字就是这么来的）

- ① Alice 用自己的密钥 k_A 加密消息 m ： $E_{k_A}(m) \longrightarrow \text{Bob}$
- ② Bob 用自己的密钥 k_B 再加密一层： $E_{k_B}(E_{k_A}(m)) \longrightarrow \text{Alice}$
- ③ Alice 用自己的 k_A 解密掉自己那层： $E_{k_B}(m) \longrightarrow \text{Bob}$
- ④ Bob 用自己的 k_B 解密： $m \checkmark$

看明白了吗：全程没有一个"密钥"被传输过，但两人通过"重重加密、依次解密"完成了保密通信。

课件原文总结：三次传输协议的特点——**重重加密，依次解密；加解密的运算顺序可交换。**

两个要注意的地方

- ① 它不能防止中间人攻击。因为协议里没有任何"验证对方身份"的步骤。攻击者可以插在中间，跟 Alice 做一套、跟 Bob 做另一套。
- ② 用什么加密算法很关键。课件留了一道分析题："分析 Shamir 三次传输协议中的加密算法如果采用异或运算导致的安全隐患。"

想想异或行不行：如果 $E_{k_A}(m) = m \oplus k_A$ ，那么第②步 Bob 拿到的是 $m \oplus k_A$ ，他再异或 k_B 得到 $m \oplus k_A \oplus k_B$ 。第③步 Alice 用 k_A 解密得到 $m \oplus k_B$ 。第④步 Bob 解出 m 。

看起来能work（异或确实满足交换律）。**但问题是：**整个过程传输的三段密文如果都被截获，攻击者可以两两异或推出密钥信息——**安全性远不如"真正有密钥保护的加密算法"**（比如 RSA），因为异或太"线性"了，没有非线性保护。

匿名密钥分配协议：不让 KDC 知道你的密钥

为什么需要它

很多协议里的密钥由可信第三方 TTP（或叫 KDC）产生和分配。但这样有个隐患：**KDC 自己就知道了所有人用的密钥**。万一 KDC 被攻破或者使坏呢？

目标：既利用 KDC 产生密钥，又让 KDC 不知道用户最终选的是哪个。

协议流程（双层加密）

① KDC 产生足够长的密钥序列，每个都用**自己的公钥 K_p** 加密后发到网上： $E_{K_p}(k_1), E_{K_p}(k_2), \dots$

② Alice 随便挑其中一个，用**自己的公钥 A_p** 再加密一层：

$$E_{A_p}(E_{K_p}(k_a)) \longrightarrow \text{KDC}$$

③ KDC 用自己的私钥 K_v 解掉自己那层，得到 $E_{A_p}(k_a)$ ，但**看不到 k_a** ，送回给 Alice

④ Alice 用**自己的私钥 A_v** 解掉自己那层，得到 k_a

关键在哪：KDC 用私钥解密后，得到的是"被 Alice 公钥锁着的东西"——他能认出这是自己发的某个密钥，但不知道是哪一个（因为他没有 Alice 的私钥）。

协议特点：保证密钥的保密性，减弱对 KDC 安全性的依赖。

智力扑克协议：网上打牌怎么防作弊

为什么需要它

Alice 和 Bob 通过网络玩五张牌的游戏。难点在哪？

- **发牌要公平：**洗牌不能作弊
- **双方都看不到对方的牌，**但又必须知道自己拿的是什么
- **牌必须是真的：**不能重复、不能被偷看

协议流程（双层加密 + 盲选）

① Alice 产生 52 个消息 $m_1 \sim m_{52}$ （代表 52 张牌），用**自己的公钥 A_p** 加密，全部发给 Bob：

$$C_1 \sim C_{52} = E_{A_p}(m_i)$$

- ② Bob 随机挑 5 张（他不知道是哪 5 张，因为他解不开），再用**自己的公钥 Bp** 加密一层，发回给 Alice：

$$E_{B_p}(E_{A_p}(m_i))$$

- ③ Alice 用自己的私钥 Av 解掉自己那层 $\rightarrow E_{B_p}(m_i) \rightarrow$ 发给 Bob

Bob 用自己的 Bv 解密 $\rightarrow$ **他知道自己拿到了哪 5 张**

- ④ Bob 再随机挑 5 张（剩下的），用 E_{A_p} 加密后发给 Alice $\rightarrow$ Alice 用自己的私钥解开 $\rightarrow$ **她知道自己的牌**

妙处：Bob 挑牌时是“盲挑”的（他解不开 Alice 的加密）；Alice 也不会知道 Bob 挑的是哪 5 张。

不经意传输协议（OT）：发送方不知道你收到哪个

为什么需要它

这个需求听起来很怪：**Alice 有两个秘密 m_1 和 m_2 ，想给 Bob 其中一个，但不想知道 Bob 选了哪个。**

现实中哪里用得上？——**隐私保护的安全计算**。比如你不想让服务器知道你查了哪条记录，但又得从它那里拿到数据。

课件列了它的各种变体：1/2-out-of-1、1-out-of-2、1-out-of-n、m-out-of-n。

协议流程（基于两对公钥）

- ① Alice 产生两对密钥：kp1/kv1 和 kp2/kv2，把两个公钥都发给 Bob
- ② Bob 自己选一个对称密钥 k，用**其中一对公钥**（比如 kp1）加密后发回： $E_{p_1}(k)$
- ③ Alice 用两把私钥分别解密：
- $$D_{v_1}(E_{p_1}(k)) = \mathbf{k} \quad \leftarrow \text{这把解对了}$$
- $$D_{v_2}(E_{p_1}(k)) = \mathbf{k'} \quad \leftarrow \text{这把解出来是乱码}$$
- ④ Alice 发送： $E_k(m_1), E_k(m_2)$
- ⑤ Bob 用 k 解密 —— **只有 m_1 能解开， m_2 解出来是乱码**

看明白了吗：Alice 不知道 Bob 用了哪对公钥，所以不知道他拿到的是 m_1 还是 m_2 ；Bob 也只能拿到其中一个。

4.1 小结

协议	协议特点
Shamir 三次传输	不需协商密钥，三次传输实现保密通信；不能防止中间人攻击
匿名密钥分配	保证密钥的保密性， 减弱对 KDC 安全性的依赖
智能扑克协议	公钥密码算法， 避免网络游戏的作弊
不经意传输协议	公钥 + 对称密码，实现消息保密传输，常用于 公平性和安全计算

▼ 自测：Shamir 三次传输协议为什么能"不协商密钥"就通信？

因为它利用了"加密算法满足交换律"这个性质。

核心思路是：Alice 和 Bob 各自给自己的消息"上一把锁"，谁也不需要把钥匙给对方。

Alice 上锁 (kA) → Bob 再上锁 (kB) → Alice 开自己的锁 (kA) → Bob 开自己的锁 (kB)

→ 回到原始消息

因为"两次上锁"和"两次开锁"的**顺序可以交换**，所以 Alice 能在 Bob 还没解开自己那层的情况下，先解掉自己那层。

为什么它不能防中间人？ 因为它只解决"传输保密"，没有解决"对面是谁"——攻击者完全可以分别跟 Alice 和 Bob 各做一次三次传输，然后转发。

4.2 安全多方计算：一起算，但谁都看不到对方的数据

三个很贴近生活的场景

- ① **平均薪资**：某部门 Alice、Bob、Carol 想知道三人的平均工资，但公司规定不许向任何人泄露自己的工资。
- ② **姚氏百万富翁**：Alice 和 Bob 想知道谁的工资高，但不能泄露各自的工资。
- ③ **隐私查询**：你拥有多本电子书（都是"我有"这个事实），我想买其中一本，但不想让你知道我要买哪一本。

这三个问题的共同点：**要算一个结果，但谁都不想暴露自己的原始数据。**

应用背景与解法

这个问题还有个更实际的版本：**数据挖掘 / 隐私保护**。比如"购买了该商品的用户还购买了……"、"根据浏览猜我喜欢……"——这些都需要在**不泄露用户隐私**的前提下完成计算。

还有一个相关技术叫**同态加密**：

$$D(E(a) \otimes E(b)) = a \otimes b$$

什么意思？ 你可以直接对"加密后的数据"做运算，算完解密，得到的就是"对原始数据做同样运算"的结果。**全程不用解密。**

类型	算法
乘法同态	RSA、ElGamal
加法同态	ECC、Paillier
全同态	Gentry 算法

应用例子：离线支付时验证“钱包余额 $\geq$ 支付金额”，但不需要知道具体余额是多少。

核心设计思想（一句话）

把需要计算的数据“盲化”；处理完之后再去掉盲化因子。

怎么盲化？课件说：加上或乘以随机数、哈希或加密。

目标：在计算的同时保证数据的保密性。

“盲化”这个词很形象——就是给数据戴个眼罩，让经手的人看不清，但计算照样能做。

应用 1：平均薪资的保密计算（最简单也最典型）

Alice 有 s_1 ，Bob 有 s_2 ，Carol 有 s_3

- ① Bob 随机选一个大数 r ，把自己工资盲化： $s_2 + r$ ，传给 Alice
- ② Alice 加上自己的： $s_1 + s_2 + r$ ，传给 Carol
- ③ Carol 加上自己的： $s_1 + s_2 + s_3 + r$ ，公开公布
- ④ 最后算： $((s_1 + s_2 + s_3 + r) - r) / 3 =$ 平均工资

为什么安全？因为每个人看到的都是“带着随机数 r 的和”，从里面推不出任何一个人的单独工资。

比如 Alice 拿到的是 $s_2 + r$ —— 她知道 $s_2 + r$ 这个值，但不知道 r 是多少，所以不知道 s_2 。

最后公开的也是 $s_1 + s_2 + s_3 + r$ ，还是不知道 r 。

只有把 r 去掉，才能得到总和。而 r 是 Bob 私下选的。

应用 2：离散对数的保密计算

场景：Alice 想让 KDC（密钥分配中心，算力强）帮忙找回私钥 x ，但不想让 KDC 知道 x 。公钥 y 与私钥 x 的关系是 $y = g^x \bmod p$ 。

- ① Alice 选一个随机数 $r < p$ ，把公钥盲化： $y' \equiv y \cdot g^r \pmod{p}$ ，发给 KDC
- ② KDC 算出 x' 满足 $g^{x'} \equiv y' \pmod{p}$ ，返回给 Alice
- ③ Alice 去掉盲化因子： $x \equiv (x' - r) \bmod (p-1)$ ，得到真正的 x

可解性验证（用费马小定理）：

$$g^{x'} \equiv y' \equiv y \cdot g^r \equiv g^x \cdot g^r \equiv g^{x+r} \pmod{p}$$

$$\Rightarrow g^{x'-x-r} \equiv 1 \pmod{p} \quad \leftarrow \text{两边都除以 } g^{x+r}$$

$$\Rightarrow x' - x - r = k(p-1) \quad \leftarrow \text{费马小定理：} g^t \equiv 1 \text{ 时 } t \text{ 是 } p-1 \text{ 的倍数}$$

$\implies x \equiv (x' - r) \pmod{p-1}$ $\leftarrow$ 模掉 $p-1$ 就得到 x

KDC 只看到 y' 和 x' ——他能算出 x' ，但不知道 r ，所以永远得不到真正的 x 。

应用 3：姚氏百万富翁问题（最经典）

问题：Alice 有 x ，Bob 有 y ，在不泄露 x 和 y 的情况下比较大小。

课件参数： $1 \leq x, y \leq 4$, $x=4$, $y=2$, Bob 公钥 $k_p=7$ 、私钥 $k_v=23$, $n=55$

协议流程：

Alice 侧：

随机选 $k = 39$ ，计算 $c = 39^7 \bmod 55 = 19$ （用 Bob 的公钥加密）

把 $c - x = 19 - 4 = 15$ 发给 Bob

Bob 侧：

计算 4 个数： $z_i = D(c - x + i)$, $i = 1-4$ （用自己的私钥解密）

$$z_1 = 16^{23} \bmod 55 = 26$$

$$z_2 = 17^{23} \bmod 55 = 18$$

$$z_3 = 18^{23} \bmod 55 = 2$$

$$z_4 = 19^{23} \bmod 55 = 39$$

对 $i > y$ 的项，各加 1（ $y=2$ ，所以 $z_3 \rightarrow 3$, $z_4 \rightarrow 40$ ）

发送：26, 18, 3, 40

Alice 侧：

检查第 $x = 4$ 个数是否等于 $k = 39$

第 4 个数是 $40 \neq 39 \implies x > y \checkmark$ （ $4 > 2$ ，正确）

判定规则：

检查结果	结论
第 x 个数 等于 k	$x \leq y$
第 x 个数 不等于 k	$x > y$

为什么这样能判断？

- 如果 $x \leq y$ ，那么第 x 个数没被加 1，它等于 $D(c-x+x) = D(c) = k$
- 如果 $x > y$ ，那么第 x 个数被加了 1，等于 $k+1 \neq k$

妙在哪：把"比较大小"巧妙地转换成了"位置的先后关系"。Alice 只看到一串数字，从里面推不出 y 是多少（因为她不知道哪些被加了 1）。

4.2 小结

安全多方计算协议的流程与特点：首先将数据盲化，数据处理完后再去掉盲化因子。在计算的同时保证数据的保密性。

协议	协议特点
平均薪资的保密计算	采用盲化因子，数据需要保密传输，实现在保密个人薪资的前提下计算平均工资
离散对数的保密计算	采用盲化因子，根据 费马小定理 还原真实结果
姚氏百万富翁问题	利用两个数据构造相关的 位置关系 ，采用加密的方式在保密前提下比较

▼ 自测：平均工资的例子为什么是安全的？加了随机数 r 有什么用？

随机数 r 就是"盲化因子"，它让每个人看到的東西都变成一个"混合了别人数据、但自己拆不开"的值。

具体看：

- Alice 看到 $s_2 + r$ —— 里面混了 Bob 的工资和 r ，但她不知道 r ，所以推不出 s_2
- Carol 看到 $s_1 + s_2 + r$ —— 更拆不开
- 最后公开的 $s_1 + s_2 + s_3 + r$ —— 只要 r 不泄露，还是拆不出任何一个人的工资

关键： r 是 Bob 私下选的，没有人知道它；算完最后一步把它减掉，和就出来了。

这和 1.3 节的 OTP、4.1 节的匿名密钥分配是**同一个思想**——用一个随机数把数据"藏起来"，用完再拿掉。这个技巧叫"盲化"，是密码协议里最常用的手法之一。

4.3 密钥分发协议：广播环境下怎么省代价

场景

一个网络里有 A、B、C、D、E、F 共 6 人，两两之间都有共享密钥。

A 想把消息 m 广播给 B、C、E 三个人（D 和 F 不能收到）。怎么做代价最小？

（课件还提到实际背景：无线传感器网络里，节点可能新加入、可能退出、可能叛变。）

方案一：传统做法

A 选一个会话密钥 k_s ，用它加密消息： $E_{k_s}(m)$

再分别用与 B、C、E 的共享密钥加密 k_s ： $E_{k_B}(k_s), E_{k_C}(k_s), E_{k_E}(k_s)$

全部广播出去： $E_{k_s}(m) \parallel E_{k_B}(k_s) \parallel E_{k_C}(k_s) \parallel E_{k_E}(k_s)$

代价：A 做 4 次加密计算，广播 4 个数据包。接收方要做 3 或 4 次解密。

方案二：用中国剩余定理（CRT）

思路：不用一个个加密，而是**构造一个数 R**，让有权限的人“模一下”就能拿到密钥，没权限的人拿不到。

$$R \equiv k_s \pmod{k_B} \quad \leftarrow B \text{ 有 } k_B, \text{ 可以算 } R \bmod k_B \text{ 得到密钥}$$

$$R \equiv k_s \pmod{k_C} \quad \leftarrow C \text{ 同理}$$

$$R \equiv k_s \pmod{k_E} \quad \leftarrow E \text{ 同理}$$

$$R \equiv 0 \pmod{k_D} \quad \leftarrow D \text{ 没有权限，算出来是 } 0$$

$$R \equiv 0 \pmod{k_F} \quad \leftarrow F \text{ 同理}$$

广播： $E_{k_s}(m) \parallel R$

代价：A 做 **1 次加密 + 1 次 CRT 计算**，广播 **2 个数据包**。接收方做 1 次模运算，0 或 1 次解密。

为什么 R 能被好几个人同时用？

因为 CRT 的神奇之处：

一个数 R，可以同时满足好几个不同的同余条件，而且每个有权限的人只看自己那一条。

这不是魔法——CRT 就是这个意思：用一组两两互素的模数，把一个数拆成多个“影子”，谁拿着对应的模数，谁就能从 R 里取出自己的那份。

性能对比（这个结论很重要）

假设网络有 n 人，A 要发给其中 k 人：

方式	通信代价	计算代价
传统密钥分发	$k+1$	发送方 $(k+1)$ 次加密；接收方 $(k+1)$ 或 k 次解密
基于 CRT	2	发送方 1 次加密 + 1 次 CRT；接收方 1 次模运算，0 或 1 次解密

关键结论：

- 传统做法：代价与用户数 k 成**线性关系**，是 $O(k)$
- CRT 做法：代价与用户数 k 成**常数关系**，是 $O(1)$

什么意思？ 如果 A 要发给 1000 个人，传统做法要广播 1001 个包；CRT 做法**永远只广播 2 个包**。

应用：课件说它不仅可以用于广播/组播保密通信，还可以用于**密钥更新**。

▼ 自测：为什么基于 CRT 的密钥分发能把通信代价从 $O(k)$ 降到 $O(1)$ ？

因为它用一个“大数 R ”同时把密钥交给了所有人，而不是一个个加密。

传统做法：要给 k 个人，就得用 k 把不同的密钥分别加密 k 份密钥——所以是 $k+1$ 个包（加上消息本身）。

CRT 做法：构造一个 R ，让它满足 k 个同余条件（ $R \equiv k_s \pmod{k_i}$ ）。所有人看到的都是同一个 R ，但只有拿到对应 k_i 的人才能从 R 里“取出”密钥。所以只需要广播 1 个 $R + 1$ 个加密消息 = 2 个包，与人多人少无关。

本质区别：传统做法是“给每个人单独打包”，CRT 做法是“造一个大家都能用、但只有有权者能读的公共容器”。

4.4 位承诺协议：先锁住一个想法，以后不能改

生活场景

网络竞标：你要投标，但不想让别人知道你投了多少钱；同时你要保证，等开标的时候不能再改价格。

怎么做到“先说定了，但先不说是是什么”？

这就是位承诺协议要解决的问题：Alice 对 Bob 承诺一个位（0 或 1），直到某个时间后才揭示，承诺后不能改变想法。

课件说它的目的是“敏感信息保护机制中，实现机密性和不可否认性”。

方案一：用单向函数（哈希）

承诺阶段

- ① Alice 产生要承诺的位 b ，以及两个随机位串 r_1 、 r_2
- ② 计算哈希值 $H(r_1, r_2, b)$ ，把 $H(r_1, r_2, b)$ 和 r_1 一起发给 Bob

揭示阶段

- ③ Alice 把 r_1 、 r_2 、 b 全部发给 Bob
- ④ Bob 重新算一遍 $H(r_1, r_2, b)$ ，和之前收到的对比 → 一致就确认

两个随机数各有什么用？（这里最容易被绕）

随机数	防的是谁	原理
r_1	防 Alice 自己反悔	因为 r_1 提前交给了 Bob，Alice 想改 b 就得连 r_1 一起改，但 Bob 手里已经有了老的 r_1 ，对不上
r_2	防 Bob 偷看	因为 b 只有 2 种可能，Bob 完全可以自己算 $H(r_1, r_2, 0)$ 和 $H(r_1, r_2, 1)$ 去猜。加了 r_2 之后，他得同时猜中 r_2 才能试出来——而 r_2 是随机的，猜不着

课件原话：

- “随机数 r_1 使得 Alice 不能篡改承诺结果”

- "随机数 r_2 让 Bob 不能猜测承诺结果"

方案二：用对称密码

承诺阶段

- ① Alice 产生位 b 、随机密钥 k 、随机位串 r
- ② 计算 $c = E_k(r, b)$
- ③ 把 r 发给 Bob；再发 c

揭示阶段

- ④ Alice 把 k 发给 Bob
- ⑤ Bob 解密获得 b ，并用 r 验证有效性

对称密码方案的对应关系（课件原话）：

- "随机数 r 保证 Alice 不能篡改承诺结果"
- "密钥 k 用来让 Bob 验证承诺结果"

两种方法对照（背这张表）

采用算法	防篡改	防猜测
散列函数	靠随机数 r_1	靠随机数 r_2
对称密码	靠随机数 r	靠密钥 k （没 k 解不出来，也验证不了）

统一的理解：位承诺要同时满足两件相反的事——

- 对承诺者：说了不能改（**绑住自己**）
- 对接收者：现在猜不出来（**锁住秘密**）

所以需要"一个公开的东西（证明你没改）" + "一个私藏的东西（让他猜不着）"。

▼ 自测：位承诺协议里，为什么要"两个随机数"（哈希方案）？

因为位承诺要同时防两个人，而这两个需求方向相反。

防 Alice（承诺者）反悔：需要有些东西提前公开出去，这样她想改也改不了。这就是 r_1 的作用——它在承诺阶段就交给 Bob 了。

防 Bob（接收者）猜测：需要有些东西保持秘密。为什么？因为 b 只有 0 和 1 两种可能——如果 Bob 知道 r_1 ，他可以直接算 $H(r_1, ?, 0)$ 和 $H(r_1, ?, 1)$ ，只要猜出另一个参数就能确定 b 。 r_2 就是那个"猜不着"的参数，它一直保密到揭示阶段。

所以： r_1 公开（绑住 Alice）， r_2 保密（挡住 Bob）。

对称密码方案对应的是： r 公开（绑住 Alice）， k 保密（挡住 Bob，同时用于验证）。

第 5 章

高级密码协议

这一章要解决什么

这一章的协议都有一个共同点：它们处理的都是“信任”和“身份”的关系，而不只是“保密”。

- **零知识证明**：我怎么证明我知道秘密，但不告诉你秘密是什么？
- **盲签名**：你怎么给我签字，但不知道你签的是什么？
- **同时签约**：签合同怎么保证“同时生效”，谁都不能先跑？
- **匿名投票**：怎么让投票既匿名又防作弊？
- **匿名通信**：怎么发消息而不暴露“是谁发的”？
- **区块链**：怎么在没有中心机构的情况下，让所有人都认可同一本账？

5.1 零知识证明

为什么需要它

现实中有很多场景，你要证明“我知道某事”，但不能说出来：

- 证明我知道这个账户的密码——但我不想把密码告诉你
- 证明我年龄超过 18 岁——但我不想说出具体生日
- 证明我拥有这间房——但我不想给你钥匙

零知识证明：A 要向 B 证明她拥有一些秘密，且在证明中不泄漏任何信息。

著名的洞穴协议（这个例子一下子就懂了）

想象一个环形洞穴，有左右两个入口 C 和 D，中间有一道门，只有会念咒语的人才能打开。

B（站在洞外）

A（进洞）

C ———— 门 ———— D

协议过程：

- ① B 站在洞外，A 走进洞里
- ② A 自己选择从左边（C）或右边（D）进
- ③ B 进洞后，**随机指定** A 从某一边出来
- ④ A 按要求执行 —— **如果她会咒语，她一定能做到**

⑤ 重复 (1)~(4) n 次

为什么这能证明？

- 如果 A 会咒语 → 无论 B 要求从哪边出，她都能打开门绕过去 → **每次都成功**
- 如果 A 不会咒语 → 她进去时在哪边，就只能从哪边出来。所以每次被指定的边**刚好是她所在的那边**的概率是 $1/2$ 。重复 n 次都碰巧猜对的概率是 $(1/2)^n$ ，趋近于 0

最关键的一点：整个过程，B **没有获得关于咒语的任何信息**——他只知道“A 一定知道咒语”，但不知道咒语是什么。这就是“零知识”——**知识为零，信心满满**。

三个关键特征

特征	含义
完整性	如果 A 真的知道秘密，她一定能通过验证
可靠性	如果 A 不知道秘密，她通过的概率极低 ($1/2^n$)
零知识性	验证者除了“知道 A 知道”之外， 得不到任何额外信息

▼ 自测：为什么洞穴协议里要“重复 n 次”？

因为单次只能把欺骗概率降到 $1/2$ ，不够低。

一次验证中，不会咒语的人蒙对的概率是 $1/2$ ——这等于掷硬币，太容易蒙混过关。

重复 n 次之后，每次都蒙对的概率是 $(1/2)^n$ ：

- $n = 10$ 时约 0.1%
- $n = 20$ 时约百万分之一
- $n = 30$ 时约十亿分之一

为什么要“交互式”多次？ 因为零知识证明的“零知识”性质是通过**随机性 + 重复**来实现的——验证者每次都随机提问，证明者不能提前准备，只能靠真本事。

（顺便：现代有“非交互式零知识证明”技术，把多次交互压成一个证明，区块链的隐私交易就用到它。）

5.2 盲签名

为什么需要它

签名本来就是为了“知道你签了什么”。但有些场景正好相反——**我要你签字，但不想让你知道内容：**

- **公证员：**只要证明“某时刻公证过这个文件”，并不关心内容
- **电子现金：**银行要给你签一张钞票，但不能知道这张钞票给了谁
- **匿名投票：**机构要给你签一张选票，但不能知道你会投给谁

课件原话：**签名者不知道所签文件的内容。**

完全盲签名

前提：签名函数和乘法函数满足交换律。

- ① Alice 把消息"盲化": $m' = m \times b$ (b 是盲化因子)
- ② 把 m' 发给 Bob, Bob 对它签名: 得到 $S_{BV}(m')$
- ③ Alice 去掉盲因子 b , 就得到对原文 m 的签名: $S_{BV}(m)$

在 RSA 里就是: $m' = m \cdot k^e \bmod n$

特点:

- Alice 手中的签名有效 (别人验证时能通过)
- Bob 无法把"签署行为"和"所签文件"联系起来 (他签的时候看到的是 m' , 不是 m)

大白话: 隔着信封签字

你在一张纸上写了内容, 把它装进一个带复写纸的信封, 让别人在信封外面签字。他一签, 压力透过复写纸, 就在里面的纸上留下了签名。

他签的时候看不到里面的内容, 但签完之后里面的文件确实有他的签名。

这就是盲签名。

但完全盲签名有个漏洞: Bob 可能被骗

如果 Bob 什么都不知道就签, Alice 完全可以骗他去签一份"Bob 欠 Alice 100 万"的文件。

怎么防? 课件给了办法: 分割选择法。

分割选择法: 签一个, 检查剩下的

- ① Alice 准备 n 个文件 $m_1 \sim m_n$, 各配一个随机数 $k_1 \sim k_n$, 把它们盲化
- ② 全部发给 Bob
- ③ Bob 随机挑 $n-1$ 个, 要求 Alice 公开对应的 k_i
- ④ Alice 公开 $n-1$ 个 k_i —— Bob 逐个验证, 确认这些"没做手脚"
- ⑤ Bob 对剩下那一个 (Bob 不知道它对应什么内容) 签名
- ⑥ Alice 收到签名后还原出对 m_p 的合法签名

为什么这就安全了? 因为如果 Alice 想作弊 (比如把 n 个文件里塞一个"欠条"), 那 Bob 检查的 $n-1$ 个里很可能包括它。

Alice 能成功欺骗的概率是 $1/n$ —— n 越大越安全。

("分割选择"这个名字的意思就是: 把一批文件分成"检查的"和"签名的"两部分。)

前后呼应

还记得 2.5 节讲的 RSA 签名攻击 I 吗? ——Alice 骗 Bob 对一个"无害消息" m' 签名, 实际上等于拿到了对"借据" m 的签名。

那是同一个数学技巧，用作攻击就是"骗取签名"，用作设计就是"盲签名服务"。

所以课件在那道作业题里问："既然 RSA 签名算法存在指数运算攻击，为什么仍然应用广泛？"——因为这个"攻击"换个场景就是有用的功能。

▼ 自测：盲签名里，为什么 Bob 无法把"签名行为"和"文件内容"联系起来？

因为他看到的是"盲化后的 m' "，而不是原文 m 。

具体说：

- Bob 签字时看到的是 $m' = m \times b$ (b 是 Alice 私选的随机因子)
- 签完之后，Alice 把 b 去掉，得到对 m 的签名
- Bob 手里只留下"我签了一个像 m' 这样的数"的记录，但他不知道 m 是什么（因为不知道 b ）

更关键的是：即使以后 Alice 拿签名去用，Bob 对不上号——因为签名看起来是对 m 的，而 m' 从来没出现过在公开场合。

但这也带来一个问题：Bob 完全不知道自己签了什么，可能被骗。所以要配合"分割选择法"——先让他检查大部分，只对剩下一个"盲签"。

5.3 同时签约

生活场景

Alice 和 Bob 要签合同，内容都商量好了。但现在僵住了：

两个人都想等对方先签。

因为先签的人有风险——万一对方拿到我的签名后反悔怎么办？

方案一：带仲裁者（最简单）

引入 TTP：

① 双方各自把已签的合约交给 TTP

② TTP 签好两份副本

③ 分发给双方

→ 双方手里都有一份"两人都签过的合约"

简单可靠，但需要信任 TTP（而且 TTP 可能是瓶颈）。

方案二：面对面（最土的办法）

Alice 和 Bob 依次在合约上逐个签上自己名字的字母。

A 签 "A" → B 签 "B" → A 签 "I" → B 签 "o" → ...

一步一步交替，谁都没法"骗到"对方完整的签名

但这要求两人同时在场。

方案三：非面对面，用"概率递增"

思路： 每一步只增加一点点"接受概率"，让谁先违约都不划算。

协议中双方交换下面形式的签名：

"我同意以概率 p 接受这个合约约束"

① 双方协商一个完成日期，以及双方都愿意接受的概率差

② Alice 发 $p_A = a$

Bob 回 $p_B = p_A + b$

Alice 回 $p_A = p_B + a$

③ 重复到双方都收到 $p = 1$ 的消息，或者到了完成日期

判定规则： 如果在完成日期之前完成协议，双方受合约约束；否则法官随机选一个 p ，若 $p < (p_A, p_B)$ 则双方受约束。

妙处： 因为每一步只加一点点概率，所以"早违约"和"晚违约"的收益差很小。谁也没动力去赌那一点点时间差。

方案四：用密码技术（最优雅）

① Alice 和 Bob 各自随机选 $2n$ 个密钥，分成 n 组 (k_i, k'_i) ，

对 n 对消息 L_i 和 R_i 分别用 k_i 和 k'_i 加密，传送给对方

② 采用不经意传输协议，把 $2n$ 个密钥中的一半传送给对方

③ 用收到的密钥解密出一半消息

④ 如果能解密其中的任意一对消息，**合约被签署**

⑤ 双方交换在第 ② 步中使用的私钥，验证未欺骗

为什么这能做到"同时"？

因为信息是"要么都给，要么都不给"——第 ③ 步解出一半消息后，双方都处于"我拿到了半个合同"的状态。此时谁想退出，对方也拿不出完整的合同来害他。**要么双方都完成，要么都作废。**

(这里用到的"不经意传输"就是 4.1 节讲的 OT 协议。)

▼ 自测：方案三（概率递增）为什么能让双方"同时"受约束？

因为它的核心思想是"让时间差变得不值钱"。

看两个极端：

- 如果协议设计成"一次性全签" → 先签的人风险极大，没人愿意先签
- 如果协议设计成"每次只加 0.0001 的概率" → 双方都无所谓谁先谁后

方案三就是这样：Alice 发 $p_A = a$ ，Bob 回 $p_B = p_A + b$ ，Alice 再回 $p_A = p_B + a$ 每次只前进一小步。走到 $p = 1$ 时，双方受约束程度是一样的。

如果有人中途想跑呢？那么"跑掉"的那个只比对方多赚了极小的一步（比如 0.001 的概率优势），而对方可以拿着这份合约去找法官——法官随机选 p 来判定。这么小的收益不值得冒险。

关键设计思想：把"要么全有要么全无"的博弈，改造成了"连续小步"的博弈。

5.4 匿名投票

投票需要满足什么

课件列了五条（这是本节的考核要点）：

授权、防篡改、投一次、确保选票计算、保密

其中前四条都好理解。"保密"这一条最难——它要求实现"匿名"：既知道有人投了票，又不能知道是谁投的。

第一版：简单网络投票协议 I

Voter 填写选票 t ，用中央制表机构 CTF 的公钥加密，发送 $V, E_{TP}(t)$

CTF 解密、统计、公布结果

特点：主办方不知道投票者身份——匿名性没问题。

但问题来了：

- 投票次数不定（一个人可以投很多次）
- 无法确认投票者身份（张三可以冒充李四）

第二版：协议 I + 数字签名

Voter 填写选票 t ，先签名再用 CTF 公钥加密： $V, S_{VV}(E_{TP}(t))$

CTF 解密、验证签名后统计、公布结果

改进：保证了投票者身份和投票次数（因为签名唯一）。

但又冒出新问题：CTF 现在知道"谁投了谁的票"——因为签名暴露了身份，而且解密的选票内容也在他手里，两者可以对上号。

第三版：匿名投票协议（用盲签名）

设计思路对照现场投票：

现场投票	网络投票
Voter 领票	Voter 构建选票
Voter 填票	CTF 签名选票
Voter 投出选票	Voter 密投选票
CTF 唱票计票	CTF 解密选票
CTF 公布结果	CTF 核实选票、公布结果

关键设计：选票的构建

每张选票带一个**识别号**（比如 11011011）+ 选择标记（√）

由 CTF 签名后有效

匿名怎么保证：

- 采用 **盲签名协议**进行选票签名
- **切断投票者与选票的对应关系**，但能鉴别
- 签名者知道在干什么，但不知签名的内容

防重复投票：CTF 通过**"验证重复"**来保证一人一票——同一张识别号的选票只能用一次。

投票效果

最终这个方案实现了：**授权、防篡改、保密、投一次、确保选票计算。**

课件说它**"使投票具有现场投票的效果"**。

5.4 小结：三个方案的对比

投票协议	特点
投票协议 I	投票者身份和次数 不确定
投票协议 II	CTF 知道投票者与选票的关系（ 无法匿名 ）
匿名投票协议	保证了投票的五个特性（用盲签名切断身份与选票的联系）

这个**"逐版改进"**的过程本身就是**最好的学习材料**——你会发现每个方案解决了上一版的问题，但引入了新的问题，最后用盲签名才同时满足所有需求。

▼ 自测：为什么"投票协议 II"虽然保证了身份和次数，却无法匿名？

因为它把"身份"和"选票"绑在一起了。

协议 II 的做法是：投票者**先签名，再加密**。CTF 收到后：

1. 解密（用自己的私钥）→ 看到里面的选票内容和签名
2. 验证签名 → 知道这是谁

所以 CTF 手里同时有"选票内容"和"投票人身份"，**两者一一对应**——他可以完全知道"张三投了谁"。

匿名投票协议怎么解决的？用盲签名：CTF 签字的时候看不到选票内容（只有盲化后的形式），等投票者去掉盲因子后真正投票时，CTF 收到的是一张"自己签过、但认不出是哪张"的选票。

核心思路：把"授权"和"投出"这两个动作**在时间上分开**——先拿到"已授权的空白选票"，过一会儿再匿名投出去。这样 CTF 无法把两次动作对上号。

5.5 匿名通信

为什么需要它

前面的协议保护"内容"不被看到。但有时候，光是"**谁在跟谁说话**"这件事本身就是敏感信息。

匿名通信的目标：**隐藏通信关系**——包括"谁在发"、"谁在收"、"他们在通信"这三件事。

先看一个有意思的入门问题

场景：Alice、Bob 和 Carol 在餐馆进餐，账单被匿名支付了。能否确定是三人中的一个付的账，还是一个局外人（比如 NSA）付的？

游戏规则：三人两两抛硬币，每个人说出"自己看到的另外两人的硬币是相同还是不同"。**付账的人说假话。**

判断依据：说"不同"的人数

奇数 → 三人中的一个在付账

偶数 → 局外人付账

这个例子说明：匿名不等于"什么都查不出来"，而是"**只能推断出有限的信息**"（知道有人付账，但不知道是谁）。

匿名通信的核心技巧：异或抵消

规则：**不发消息的人发送共享密钥的异或值；发消息的人发送共享密钥与消息 m 的异或值**

三人两两共享密钥： k_{12} 、 k_{13} 、 k_{23}

Alice 发送： $m_1 = k_{13} \oplus k_{12}$

Bob 发送: $m_2 = k_{12} \oplus k_{23} \oplus m$ ← 真正发消息的人

Carol 发送: $m_3 = k_{13} \oplus k_{23}$

网关 GW 计算: $m_1 \oplus m_2 \oplus m_3 = m$ ← 密钥互相抵消, 只剩消息

为什么密钥会抵消? 因为每个密钥都出现了两次: k_{12} 在 m_1 和 m_2 里各一次, 异或两次就抵消了。

外面的人看到的是: 三个人都发了消息, 但无法判断是哪一个真正发了内容。

(这就是 DC-Net 的基本思想。)

三个基本概念 (必背, 容易混)

概念	含义	大白话
匿名性 Anonymity	一种在一组对象中不被识别的状态	藏在人群里, 看不出是哪个
不可关联性 Unlinkability	不能获得系统中对象之间是否相关的判断	看不出这两件事是同一个人干的
不可观察性 Unobservability	不能观察到对象是否发送或接收了消息	连"他有没有在通信"都不知道

难点从低到高: 匿名 < 不可关联 < 不可观察。不可观察最难——它连"有人在通信"这件事都要藏起来。

三种"匿名" (也要分清)

- **通信关系安全**: 消息的发送者与接收者的保密性
- **身份匿名** (Identity Privacy): 源节点和目的节点的身份信息保密, 它们也不知道路径上中间节点的真实身份 (发送者匿名、接收者匿名、通信关系匿名)
- **位置匿名** (Location Privacy): 源节点和目的节点的具体位置保密, 其他节点不知道它们到源节点或目的节点的跳数
- **路由匿名** (Route Privacy): 攻击者不能追踪数据包到源节点或目的节点

四种实现机制

① **代理机制**——借助可信第三方隐蔽真实身份

- **单代理**: Penet、Anonymizer、LPWA
- **多代理**: OR、Hordes、Morphmix (代理链, 更安全)

② **MIX 机制**——这是最经典也最优雅的方案

MIX 是一些特殊节点, 它做两件事:

- 通过**加密或填充**改变消息的外观
- 通过**延迟或重排序**改变消息的顺序

这样就能隐藏输入和输出的对应关系, 攻击者无法推断通信关系。

发送者把消息层层加密:

$E_1(M_1, E_2(M_2, \dots, E_l(M_l, E_R(M_R)) \dots))$

Sender → MIX1 → MIX2 → ... → MIXi → ... → MIXI → Recipient

剥掉一层 剥掉一层 剥掉一层

每一个 MIX 只能剥掉自己那层，只知道"上家"和"下家"，不知道全貌。

应用：匿名电子邮件（Babel、MixMaster、MixMinion）、匿名连接服务（OR、Tarzan、MorphMix、WonGoo）、匿名 Web 浏览（Crowds、Webmix）。

③ **Crowds 系统**——基于 P2P 对等网络思想

消息在不同的 Jondo 之间**随机转发**，转发给目的节点的概率为 p_i ，

最终被发送给目的节点，发送者就被隐藏。

特点：对称加密机制；缺点是**转发路径可能过长**。

④ **广播/多播机制：**DC-Net、Hordes、P5 系统等。每个节点发送所有共享密钥的异或值，发送者把消息加进去（就是前面那个技巧）。

▼ 自测：MIX 机制为什么能隐藏通信关系？它做了哪两件事？

MIX 通过同时改变消息的"外观"和"顺序"来切断输入输出的对应关系。

① **改变外观（加密 / 填充）：**让所有消息看起来一样长、一样"随机"。否则攻击者可以靠"消息长度"来匹配上家和下家。

② **改变顺序（延迟 / 重排序）：**不是收到就立刻转发，而是攒一批再打乱顺序转发出去。这样攻击者没法靠"时间先后"来匹配。

为什么这两件事缺一不可？

- 只改外观不改顺序 → 攻击者还能靠时间对应关系追踪
- 只改顺序不改外观 → 攻击者还能靠内容/长度对应关系追踪

合起来，攻击者面对的是一个"每次输出都可能是任何一个输入的"黑箱——**这就是匿名集（Anonymity Set）**的思想：把你藏在一群同样可疑的人里。

5.6 区块链

先说清它要解决什么问题

电子支付：银行账户上做数字的加法和减法，只有银行有记账权。

问题：这样依赖大量第三方中介机构，既引入中心化风险，又提升传输成本。

新型货币的目标：创造一种新型支付体系，大家都有权来记账，比如比特币。

怎么解决"去中心化记账"这个难题？ 课件原话：通过构造一个以"竞争 — 记账 — 奖励"为核心的经济系统。

大白话：把"记账"变成一场比赛

传统做法是"请一个大家都信的会计"（银行）。

区块链的做法是"谁能解出一道难题，谁就有权记这一页账，还能拿到奖励"。

因为题目很难，得花算力才能解出；因为解出来有奖励，大家都有动力参与；因为记完的账所有人都会核对并抄一份，所以改不动。

不需要信任任何人，只需要信任"题目够难"和"大家都会核对"。

数字现金协议：用盲签名（先看这个，它是区块链的思想源头）

要求：匿名，消息可以确认，但不可跟踪。同时银行和政府不愿意放弃审计追踪控制。

- ① Alice 准备 n 个装有 1000 元的汇票（每个前面有复写纸，装成信封）
- ② 把 n 个信封交给银行
- ③ 银行开启 $n-1$ 个确认，在**最后 1 个**上签名，并从 Alice 账上扣除 1000 元
- ④ 银行把签名的信封还回
- ⑤ Alice 把汇票给商人
- ⑥ 商人检查银行签名
- ⑦ 商人把汇票交给银行
- ⑧ 银行验证汇票，将 1000 元划入商人帐户

为什么匿名？ 因为银行签的那个信封他没打开过——不知道该汇票最终会流向谁。

三个问题及解决方案（层层递进）

问题 1：双重花费——Alice 把同一张汇票花两次。

解决：在每张汇票上包含一个不同且唯一的随机字符串。银行签名时确认，验证签名时检查这个串的唯一性。

问题 2：发现双重花费，但不知道是谁在骗谁——是 Alice 骗商人，还是商人试图骗银行？

解决：商人检查银行签名后，要求 Alice 在汇票上写一个识别串（商人无法改变它）。银行验证时：

数据库中无该串 → 划帐，并保存识别串和唯一串

数据库中有该串 → 拒收，根据识别串判断：

与数据库相同 → 商人欺骗

与数据库不同 → Alice 欺骗

问题 3：Alice 第二次花一张"有同样识别串"的汇票来陷害商人。

解决：用秘密分割技术在数字汇票中隐藏 Alice 的名字。

Alice 准备 n 张汇票，每张上：

面值 1000

唯一字符串 x

识别字符串 $l_i = (l_{iL}, l_{iR}) \leftarrow$ 分成左右两半

任意一对 l_i 都能揭示 Alice 的身份

使用时：Alice 只给出识别串的左半或右半

银行核对：

无重复 $\rightarrow$ 划帐

有重复 $\rightarrow$ 看识别串：相同（商人骗）、不同（找出一对 l_i ，揭示 Alice 身份）

欺骗概率：签名阶段 $1/n$ ；使用阶段 $1/2n$

妙在哪：正常使用只泄露"半个身份"，无法定位；只有在双重花费时才凑齐两半，暴露身份。

什么是区块链（课件给的两个定义）

科学百科（2019）：区块链是分布式数据存储、点对点传输、共识机制、加密算法等计算机技术的新型应用模式。

中国区块链技术和应用发展白皮书（2016）：广义来讲，区块链技术是利用块链式数据结构来验证与存储数据，利用分布式节点共识算法来生成和更新数据，利用密码学的方式保证数据传输和访问的安全，利用由自动化脚本代码组成的智能合约来编程和操作数据的一种全新的分布式基础架构与计算范式。

三个关键词拆解：

- 点对点的分布式网络（节点同步数据）
- 一份所有交易的账簿数据（区块链）
- 一个去中心化的交易验证（基于密码技术）

核心理想：**不再是"信任一个有足够信用度的个体"，而是"信任一群人"（51%）。**

基本步骤：广播 $\rightarrow$ 记载 $\rightarrow$ 公开

① **广播交易**：将每笔交易都在全网广播

② **节点记载**：节点接收交易信息，记载此次信息，**获得奖励**

③ **公开账簿**：其他节点核对确认，交易达到 **6 个确认以上**就成功记录在案

→ 形成一条不可更改的完整区块数据链，可溯源

区块里的内容大概是：A 存入 80 元 → A 取出 30 元 → A 存入 20 元……（每一笔都记着）

共识机制（三种）

定义：用于在分散的计算机网络中达成协议、信任和安全的方法。

作用：确保所有参与者都能够对网络事件进行有效记录。

机制	原理	缺点
工作量证明 POW (Proof of Work)	节点通过计算随机哈希散列的数值解争夺记账权	达成共识周期长，资源浪费
权益证明 POS (Proof of Stake)	根据每个节点拥有代币的比例和时间，等比例降低挖矿难度	—
股权授权证明 DPOS (Delegated PoS)	全体节点投票选出代表来代理确认区块	—

比特币：挖矿到底在干什么

起源：2008 年，一个化名为中本聪的人，在《比特币：一个点对点的电子现金系统》论文中提出。2009 年系统开始运行。

记账权：工作量证明 POW。

挖矿：寻找一个数 Nonce，使得

$$h(\text{Nonce} \parallel \text{Prev_Hash} \parallel \text{Tx}) < \text{Bits}$$

符号	含义
$h()$	SHA256 哈希函数
Bits	目标值，是一个 48 个 0 开始的比特串。每 10 分钟产生一个块；每 2016 个块（两周）调整一次难度
Nonce	一个答案，对每个区块唯一。第一个找出 Nonce 的用户得到奖金

大白话：就是不断试不同的 Nonce，算哈希，看结果是不是“以 48 个 0 开头”。因为哈希是随机的，只能一个个试——这就是“挖矿”。

奖励递减：50 BTC（210000 个块，约 4 年）→ 25 BTC → 12.5 BTC → …… 公式是 $50 \times (1/2)^n$ 。预计 2140 年全部 BTC 被挖完（ 21×10^6 个）。

区块链中的五类密码技术

#	技术	作用
1	哈希函数 SHA256	用于数据完整性和加密；共识计算的工作量证明；块之间的链接
2	Merkle 树	保存信息的哈希值，验证区块中的交易
3	公钥密码	去中心化的保证
4	数字签名 ECDSA	交易拥有者的证明
5	零知识证明	在不公开秘密密码的情况下验证其在已知限制内

这门课讲的所有东西，在区块链里都能找到用处——这就是为什么区块链出现在密码学课的最后一节。

区块链的特点

特点	说明
----	----

高透明性	公有区块链系统开放，节点共同维护，除私有信息被加密外，任何人可通过公开接口查询 (同时也带来隐私泄露问题)
高自治性	基于协商一致的规范和协议，各参与方在 不需要信任任何人 的环境下自由安全交易
高信任性	按时间先后顺序标记且难以篡改，既能保证可追溯性，又能减少造假 (交易不可撤销)
高安全性	节点参与数据变动记录，保留一份相同且完整的账本，单个节点不会影响整个账本
交易确认高延迟	—
淡化监管	—

应用领域：金融领域、票据管理、产品溯源、存证取证、数据共享。

▼ 自测：为什么区块链说"不需要信任任何人"？

因为它把"信任人"换成了"信任机制"。

传统金融要信任银行（一个中心机构）。区块链不需要信任任何单个节点，因为它依赖两个客观事实：

- 1. **题目够难**：想篡改历史，得重新算出从篡改点之后所有区块的 POW——需要超过全网 51% 的算力，成本极高。
- 2. **大家都在核对**：每个节点都保存一份完整账本。你的账本跟别人的不一样，别人就不认。

所以课件说它的核心思想是"**信任一个有足够信用度的个体，或信任一群人（51%）**"——从"信任人"变成了"信任多数人的算力"。

而保证这一切能运转的技术基础，正是这门课讲的东西：**SHA256（挖矿和链接）、数字签名（证明交易归属）、Merkle 树（验证交易）、公钥密码（去中心化）**。

第 6 章

分组密码加密模式

先看这个问题是怎么冒出来的

DES 每次只能处理 **64 位**，AES 每次只能处理 **128 位**。

可你的消息可能是 **10 MB**。那怎么办？

直接切块、一块块加密不就行了？——问题就在于“怎么切、块与块之间要不要发生关系”。

不同的切法，安全性天差地别。这就是“加密模式”要讲的东西。

6.1 为什么需要“模式”

分组加密模式概述

对称密码算法的输入长度固定，而明文消息长度不定，因此加密时要**对明文分组再进行加密运算**。

- DES：64 位一组
- AES：128 位一组

四种基本模式：ECB（基本密码）、CBC（一些反馈）、CFB（一些简单运算）、OFB。

填充方案：最后一块不够怎么办

消息长度不太可能正好是分组长度的整数倍，所以最后一块可能是“半个”。

两种填充方案：

① **规则模式**：全 0、全 1，或者 0,1 交替填充。

XXX**00005** ← 最后的 5 表示“填充了 5 个字节”

② **密文挪用**：把前一个分组的密文“借”一部分来补足最后一个短分组。

▼ 自测：为什么要记录“填充了几个字节”？

因为解密后要能去掉填充，还原出原始消息。

如果只补全 0，解密后你不知道末尾的 0 是“填充的”还是“原消息本来就有的”，那就还原不出准确长度了。

所以填充方案要把“补了多少”这个信息也编进去（比如最后那个字节写 5，表示补了 5 个字节）。这属于“填充格式”的标准约定。

6.2 ECB：各扫门前雪

模式机制

ECB 又叫**电子密码本模式**（Electronic Codebook）。最简单，每一分组独立进行加密或解密处理。

明文分组: $P_1 \ P_2 \ \dots \ P_{i-1} \ P_i \ \dots$

$E_k \ E_k \ \dots \ E_k \ E_k \ \leftarrow$ 每一块都用同样的密钥独立加密

密文分组: $C_1 \ C_2 \ \dots \ C_{i-1} \ C_i \ \dots$

加密: $C_i = E_k(P_i)$

解密: $P_i = D_k(C_i)$

注意: 每一块互相独立——一样的明文块, 加密出来一定是一样的密文块。

优点 (很实在)

- 对加密数据库等随机存取文件非常方便——因为每块独立, 你可以只解密第 100 块而不用管前面 99 块
- 密文块损坏, 仅对应明文块损坏——错误不会蔓延

致命缺点: 易受重放攻击

攻击者可以在不知道密钥的情况下修改被加密的消息, 以欺骗接收者。

课件举了个银行转账的例子。假设报文这样分块:

1~4	5~8	9~12	13~16	17~20	21~24
时间标记	A 银行	B 银行	储户姓名	帐号	金额

黑客的行动:

- ① 用“从 A 银行转账到自己在 B 银行的账户”这个操作, 重复几次、记录下这几段密文
- ② 这些密文块里, 包含了“黑客”和“999999” (他的账号)
- ③ 他截获别人 (张三) 的转账报文, 把“储户姓名”和“账号”这两个密文块替换成自己的

结果:

黑客自己的转账: 20160801 A银行 B银行 黑客 999999 10元

张三的转账: 20160815 A银行 B银行 张三 888888 10000元

黑客伪造的: 20160815 A银行 B银行 黑客 999999 10000元

↑ 替换过 ↑ 替换过 ↑ 原样保留

为什么能成功？因为 ECB 每一块独立加密，密文块可以任意拼接、替换，解密后每一块仍然合法。银行收到后解密，看到的是"数据完整、格式正确"的一条转账指令。

防范

- 银行频繁改变密钥（增加攻击者收集难度）
- 最好的方法是采用分组链接方式（比如 CBC）

为什么"分组链接"能防？因为它让每一块的加密都依赖前一块——你换掉一块，后面全乱，解密出来就是垃圾，一眼能看出被篡改。

▼ 自测：为什么 ECB 模式下"相同的明文块"会得到"相同的密文块"？这会带来什么问题？

因为 ECB 每一块都是独立加密的，用的是同一个密钥，没有任何"历史信息"参与。

$C_i = E_k(P_i)$ ——同样的 P、同样的 k，结果当然相同。

问题有两个层面：

- ① 泄露了数据的模式。经典例子：如果加密一张纯色背景的图片，ECB 会让"相同颜色的区域"变成完全一样的密文块，图像的轮廓在密文里是可见的（著名的"ECB 企鹅图"）。
- ② 可以被"拼接替换"。因为每块独立，攻击者可以拿别的报文里的密文块换进来，解密后仍然是合法的块——这就是课件里那个银行转账的攻击。

解法：用 CBC 这类"分组链接"模式，让每块的加密都依赖前面所有块。

6.3 CBC：接力棒式加密

模式机制

CBC（Cipher Block Chaining，密码分组链接）的核心思想：把前一分组的加密结果反馈到当前分组的加密中。

明文分组： $P_1 \quad P_2 \quad \dots \quad P_{i-1} \quad P_i \quad \dots$

$\oplus \quad \oplus \quad \oplus \quad \oplus$

$E_k \quad E_k \quad E_k \quad E_k$

密文分组： $C_1 \quad C_2 \quad \dots \quad C_{i-1} \quad C_i \quad \dots$

↑

IV（初始向量）

加密： $C_i = E_k(P_i \oplus C_{i-1})$

解密： $P_i = C_{i-1} \oplus D_k(C_i)$

大白话：接力赛跑

每跑完一棒，都要"交接棒"给下一棒：

- 第一棒要先有个起跑点（IV）
- 每一棒的路线 = 自己的明文 **异或** 上一棒交来的结果
- 所以任何一棒出问题，后面全乱

这就解决了 ECB 的两个问题：

- 相同的明文块，因为前面的"接力棒"不同，会得到**不同的密文块** → 看不出数据模式
- 想替换某一块？后面全部解错 → 一眼看出被篡改

初始向量 IV 的两个要点

要点	说明
作用	采用 IV 将 每个消息唯一化 （同一条明文，换个 IV 就得到完全不同的密文）
要不要保密	IV 不需保密，可以以明文形式与密文一起传送

为什么 IV 可以公开？因为它的作用只是"把相同明文区分开"，本身不承担保密职责。解密时 IV 也只是"被异或掉"，泄露 IV 不泄露密钥。

反过来说：如果 IV 每次都一样（或者固定不变），那 CBC 就退化成了 ECB——相同明文开头仍然得到相同的密文开头。

但 CBC 也有代价：错误会传播

ECB 的优点之一是"密文块损坏，仅对应明文块损坏"。CBC 就没有这个优点了——

因为每一块解出来要依赖前一块的密文，所以第 i 块密文出错，会让第 i 块和第 $i+1$ 块的明文都出错。

课件作业题就是这个："分析 CBC 模式中的错误传播。"

这其实是个 trade-off：ECB 错误不传播但安全性差；CBC 安全但错误会传播。现实里要安全，所以用 CBC。

▼ 自测：CBC 模式里 IV 为什么可以明文传输？

因为 IV 只负责"让每条消息不一样"，不负责保密。

看加密公式： $C_1 = E_k(P_1 \oplus IV)$ 。解密时 $P_1 = IV \oplus D_k(C_1)$ 。

IV 的作用是：如果没有它，两条消息的明文开头相同，密文开头就会相同（退化成 ECB）。有了 IV，两条消息的密文就完全不同了。

关键点：IV 和密钥是**两件独立的东西**。攻击者知道了 IV，他还是没有密钥 k ，解不开任何东西。

那什么时候 IV 必须保密？严格说，IV**不需要保密，但需要"不可预测"**——如果 IV 可以预测（比如每次都从 0 开始递增），攻击者能做"选择明文攻击"来分析。所以实践中的做法是：IV 公开传输，但要用随机数生成。

6.4 CFB 与 OFB：两种反馈

CFB (Cipher FeedBack, 密码反馈)

明文分组： $P_1 \ P_2 \ \dots \ P_i \ \dots$

IV $\rightarrow$ Ek $\rightarrow$ $\oplus$ $\oplus$ $\oplus$

密文分组: C₁ C₂ ... C_i ...

加密: C_i = P_i $\oplus$ E_k(C_{i-1})

解密: P_i = C_i $\oplus$ E_k(C_{i-1})

特点: 反馈在分组内和分组间进行 (可以实现"逐字节加密")。

和 CBC 的关键区别:

- CBC: 先异或再加密 (C = E_k(P $\oplus$ C_{prev}))
- CFB: 先加密再异或 (C = P $\oplus$ E_k(C_{prev}))

OFB (Output FeedBack, 输出反馈)

明文分组: P₁ P₂ ... P_i ...

$\oplus$ $\oplus$ $\oplus$

IV $\rightarrow$ Ek $\rightarrow$ Ek $\rightarrow$ Ek ...

密文分组: C₁ C₂ ... C_i ...

加密: C_i = P_i $\oplus$ S_i, 其中 S_i = E_k(S_{i-1})

解密: P_i = C_i $\oplus$ S_i, 其中 S_i = E_k(S_{i-1})

特点: 与 CFB 类似, 但反馈机制独立于明文和密文, 因此可以把大部分工作提前到明文到达之前。

CFB 和 OFB 到底差在哪? (这是最容易混的地方)

关键看那个"反馈量"是从哪来的。

	反馈量	能不能预计算
CFB	上一块的密文 C _{i-1}	不能——因为密文要等收到明文才产生
OFB	上一块的输出 S _{i-1} (只和密钥、IV 有关)	能——整条"密钥流"可以提前算好

举个例子体会差别:

CFB 像什么? 像"边听边翻译"——必须听到对方说了什么, 才能决定下一句怎么翻译。来一句话才能算一块。

OFB 像什么? 像"提前把整本密码本抄好"——等到消息来了, 直接查表异或就行。延迟更低, 适合实时通信。

OFB 的本质: 它其实退化成了一种序列密码——把分组密码 E_k 当成"密钥流生成器"来用。(回忆 2.2 节: C = P $\oplus$ 密钥流。这里 S_i 就是密钥流。)

一个重要提醒

你在课件上可能看到 CFB 的解密式写作 $P_i = C_i \oplus D_k(C_{i-1})$ ，用的是解密函数 D_k 。

标准形式的 CFB 应该是两个方向都用加密函数 E_k ——因为 CFB 把分组密码当作“密钥流发生器”来用，只从正向调用它。

这个细节不必纠结，理解“先加密再异或”这个结构就行。答题时按课件写法即可。

CBC vs CFB 对照表

	CBC	CFB
加密	$C_i = E_k(P_i \oplus C_{i-1})$	$C_i = P_i \oplus E_k(C_{i-1})$
解密	$P_i = C_{i-1} \oplus D_k(C_i)$	$P_i = C_i \oplus E_k(C_{i-1})$
用到解密函数吗	要 (D_k)	不用 (两个方向都只用 E_k)
能流式处理吗	不行 (必须凑满一块)	能 (可以逐字节)

记忆窍门：

CBC 的先异或再加密 (所以解密要用 D_k)

CFB 的先加密再异或 (所以两个方向都用 E_k)

▼ 自测：为什么 OFB 可以“提前计算”，而 CFB 不行？

因为它们“反馈量”来源不同。

- CFB 的反馈量是上一块的密文 C_{i-1} 。而密文要等明文来了、异或完才有——所以必须按顺序、一块一块地算。
- OFB 的反馈量是上一块的输出 S_{i-1} ，而 $S_i = E_k(S_{i-1})$ 只跟密钥和 IV 有关，完全不需要明文。所以可以提前把整条密钥流 $S_1, S_2, S_3 \dots$ 全部算好。

好处：等消息真正到达时，只需要做一次异或就能出密文——延迟极低，适合实时音视频这类场景。

本质：OFB 实际上就是把分组密码“当序列密码用”—— S_i 就是密钥流。这也意味着它继承了序列密码的特性：密钥流绝对不能用两次 (同 IV、同密钥会产生同样的流)。

6.5 其他模式

BC 模式 (Block Chaining, 分组链接)

简单地将分组密码算法的输入，与前面所有密文分组的异或值相异或。

$$\text{加密: } C_i = E_k(P_i \oplus F_i) \quad F_{i+1} = F_i \oplus C_i$$

$$\text{解密: } P_i = F_i \oplus D_k(C_i) \quad F_{i+1} = F_i \oplus C_i$$

其中 $F_1 = IV$

和 CBC 的区别：CBC 只跟前一块密文异或；BC 跟前面所有密文的累积异或值异或。

PCBC 模式 (扩散分组密码链接)

与 CBC 不同的是：加密前，把前面的明文分组、密文分组，以及当前明文分组一起相异或。

加密: $C_i = E_k(P_i \oplus C_{i-1} \oplus P_{i-1})$

解密: $P_i = C_{i-1} \oplus P_{i-1} \oplus D_k(C_i)$

课件提到: **PCBC 模式被用在 Kerberos 4 中, 但 Kerberos 5 发现了其存在问题, 用 CBC 模式取代。**

PCBC 的问题: 交换两个密文分组, 错误会抵消

这个攻击很巧妙, 值得跟着算一遍。

原文: $P_1 P_2 P_3 P_4$ 原密文: $C_1 C_2 C_3 C_4$

【攻击】把中间两个密文分组交换:

新密文: $C_1' = C_1$ $C_2' = C_3$ $C_3' = C_2$ $C_4' = C_4$

【解密】(逐个算)

$P_2' = C_1' \oplus P_1' \oplus D_k(C_2') = C_1 \oplus P_1 \oplus P_3 \oplus C_2 \oplus P_2$

$P_3' = C_2' \oplus P_2' \oplus D_k(C_3')$

$= C_3 \oplus C_1 \oplus P_1 \oplus P_3 \oplus C_2 \oplus P_2 \oplus P_2 \oplus C_1 \oplus P_1$

$= C_3 \oplus P_3 \oplus C_2$ ← 很多项两两抵消

$P_4' = C_3' \oplus P_3' \oplus D_k(C_4')$

$= C_2 \oplus C_3 \oplus P_3 \oplus C_2 \oplus P_4 \oplus C_3 \oplus P_3$

$= P_4$ ← 居然完好无损!

问题在哪: 交换了中间两块密文, 但 **P_4 解密出来是正确的**。攻击者可以利用这一点——让接收方以为“收到的是正确的 P_4 ”, 而实际上中间的内容已经被改动过了 (P_2 、 P_3 变成了乱码或错值)。

根子是“明文和密文异或的性质”: **PCBC 把前一块的明文也掺进去了, 结果反而让错误在异或中相互抵消。**

六种模式总览 (存这张表)

模式	加密公式	解密公式	特点
ECB	$C_i = E_k(P_i)$	$P_i = D_k(C_i)$	分组独立; 随机存取方便; 易受重放攻击

CBC	$C_i = E_k(P_i \oplus C_{i-1})$	$P_i = C_{i-1} \oplus D_k(C_i)$	需要 IV；IV 可明文传输；错误会传播
CFB	$C_i = P_i \oplus E_k(C_{i-1})$	$P_i = C_i \oplus E_k(C_{i-1})$	反馈来自密文；不能预计算；可流式
OFB	$C_i = P_i \oplus S_i, S_i = E_k(S_{i-1})$	$P_i = C_i \oplus S_i$	反馈独立于明文密文；可预计算密钥流
BC	$C_i = E_k(P_i \oplus F_i), F_{i+1} = F_i \oplus C_i$	$P_i = F_i \oplus D_k(C_i)$	$F_1 = IV$ ；与前面所有密文异或
PCBC	$C_i = E_k(P_i \oplus C_{i-1} \oplus P_{i-1})$	$P_i = C_{i-1} \oplus P_{i-1} \oplus D_k(C_i)$	Kerberos4 用过；交换密文分组错误会抵消

▼ 自测：ECB、CBC、CFB、OFB 四种模式，各自的"密码块之间有没有联系"？

模式	块之间有关系吗	怎么建立关系
ECB	没有，完全独立	—
CBC	有	加密前，先和前一块的密文异或
CFB	有	用前一块的密文加密后，再和明文异或
OFB	有（但和明文无关）	用前一块的输出继续加密，生成密钥流

记忆线索：ECB"各扫门前雪" → 独立；CBC/CFB"接力棒" → 依赖前一块；OFB"自己转自己的" → 只依赖自己和密钥。

实用选择：

- 要随机存取（如加密数据库） → ECB（但要小心重放）
- 通用加密 → **CBC（最常用）**
- 流式/逐字节 → CFB
- 需要预计算、低延迟 → OFB

第 7 章

密钥管理

为什么这门课要用一整章讲“管钥匙”

回到第 0 章那句话：**数据的安全基于密钥，而不是密码算法。**

那推论就很清楚了——**密钥是整条链上最脆弱的一环。**算法再强，密钥管不好就全废。

这一章就是讲“密钥从生到死，每一步要注意什么”。

7.1 密钥的一生

密钥生命周期（六个阶段，必背）

密钥生成 → 密钥存储 → 密钥传输 → 密钥使用 → 密钥更新 → 密钥销毁

每个阶段都有对应的性能需求：

阶段	性能需求	通俗说
密钥生成	有效性	造出来的钥匙得真好使
密钥存储	安全性	存的地方得没人能偷到
密钥传输	可靠性	送过去的路上不能丢、不能错
密钥使用	多样性	不同用途用不同的钥匙
密钥更新	高效性	换钥匙不能太麻烦
密钥销毁	彻底性	作废就得真作废，不能残留

密码系统的安全性来自哪三个方面

来源	说明
密码算法	对称密码靠 算法强度 ；公钥密码靠 数学难题 （因子分解、离散对数、单向陷门）
密钥空间	靠 穷举攻击 的难度——也就是密钥长度

课件还提了两个“权衡”：

- **数学难题的困难性**——数论和复杂度理论的发展会改变它（以前认为安全的，将来可能被攻破）
- **密钥长度与代价之间的权衡**——破译的明文消息的价值 vs 破译所花的代价

密钥生成（一）：普通密钥（就是你的口令）

问题：人们天生会选弱密钥

课件说得很直接：人们选择自己的密钥时，通常选择一个弱密钥，如选择的是 **zs111** 而不是 ***9(hH/A。**

为什么？因为好记。而好记的另一面就是——**好猜。**

三种针对口令的攻击

攻击	做法
----	----

字典攻击	根据用户信息尝试最可能的密钥： · 用户姓名、电话、身份证、帐户等有关信息 · 各种单词的不同置换形式，如大小写、简写
撞库攻击	攻击者先收集互联网已泄露的用户+密码信息， 生成对应的字典表 ；再用字典中罗列的用户和密码， 尝试批量登录其他网站 。
拖库攻击	攻击者入侵有价值的网站， 把注册用户的资料数据库全部盗走 的行为。（互联网各种信息泄露事件）

三个词的关系：先"拖库"拿到一批账号密码，再用它们做成"字典"去"撞库"其他网站。**这就是为什么"一个密码走天下"极其危险——一个网站泄露，你所有账号都危险。**

怎么办：怎么造一个安全口令

课件的要求：密钥既容易记忆，又难以被猜中。

建议：

- 加入标点符号
- 由较长短语的首字母组成字母串
- 根据安全级别设置 3 个左右的口令

Stand high and see far. → ^Shasf?520^

Chuan guo shui wu hen. → ^Cgsw#118^

看看这个方法好在哪：

- 对你自己：只要记住那句英文，字母串就出来了
- 对攻击者：这不是任何词典里的单词，也不是常见的姓名生日，字典攻击无效
- 加了标点和数字：连"纯字母串"的规律也没有

分三档管理：

级别	用于
最高级别	网银等重要密码
中等级别	QQ、微信、重要邮箱等登录密码
普通级别	临时注册使用的一次性密码

为什么要分三档？因为不同场合的安全要求不同，而且分成三档能避免"所有账号用同一个密码"——一个泄露不至于全盘皆输。

密钥生成（二）：随机密钥

PRNG 的本质局限

随机密钥一般应用于各种网络通信中。

"任意由确定过程生成的随机数序列，从实际意义上讲都不是随机的。"

所以叫伪随机数发生器 PRNG（pseudo-random number generators），要求是：随机数产生的周期足够长。

两种常用的 PRNG：

- 线性同余发生器（见 2.2 节）
- 反馈移位寄存器 LFSR（见 2.2 节）

回到 2.2 节那句话

现在你应该完全理解“算法不具备随机性”这句话了。

计算机是确定的：给它同样的输入，它永远给出同样的输出。所以计算机**不可能产生真正的随机数**——只能产生“看起来随机、但其实是确定的”序列。

唯一的要求就是周期够长：让攻击者在实用时间内等不到重复。

（真正的随机数只能从物理世界获取：大气噪声、放射性衰变、鼠标移动轨迹……这些叫“真随机数发生器”。）

其余四个阶段

存储 / 传输 / 使用 / 更新 / 销毁

阶段	问题	方法
存储与备份	如何保存密钥？如何避免被窃取？	密钥托管、门限方案（见 3.6 节）
传输与验证	如何保证密钥传输的安全和可靠性？	密钥交换、密钥分发（见第 3、4 章）
使用	密钥用多久？	区分长期密钥和会话密钥；通信中使用密钥交换协议， 会话密钥不保存
更新	什么时候换？怎么换？	不能长期高频率地使用同一个密钥 ；从旧密钥中产生新密钥， 如用 Hash 函数
销毁	怎么彻底作废？	物理的 ：粉碎载体； 软件的 ：采用特殊的删除程序

两个容易忽略的点

① “会话密钥不保存”为什么重要？

因为保存着的密钥越多，“被偷的目标”越多。会话密钥用完就忘，等于**把攻击面缩到最小**。（这也呼应了 OTP 的思想——密钥用过就丢。）

② 为什么“不能长期高频率使用同一个密钥”？

因为用得越多，攻击者能收集到的“样本”越多。密码分析的很多方法就是靠大量数据找规律（回忆 1.3 节 Vigenère 的破译——文本越长越容易被破）。

顺便：课件说“从旧密钥中产生新的密钥，如用 Hash 函数”——这就是为什么 SKEY 协议（3.5 节）能用一条哈希链来生成一次性口令。

▼ 自测：为什么"一个密码走天下"很危险？用课件里的术语说。

因为"拖库" + "撞库"的组合拳。

拖库：攻击者入侵某个网站，把它的用户数据库整个偷走。这一步你防不了——不是你的错。

撞库：他拿着偷来的"用户名+密码"列表，去**批量试其他网站**。因为很多人所有网站用同一个密码，所以成功率很高。

所以：你在任何一个不太重要的网站上用的密码，都可能成为攻破你银行账户的钥匙。

对策就是课件建议的"根据安全级别设置 3 个左右的口令"：

- 最高级别（网银）——一个独特且复杂的
- 中等级别（微信、邮箱）——另一个
- 普通级别（临时注册）——可以随便一点

核心原则：重要账号和普通账号的密码必须不同，把"一损俱损"的风险切断。

7.2 Kerberos：一个管钥匙的管家

为什么需要它

回到 3.2 节那个问题：对称密码协议需要"双方共享密钥"，但人一多，密钥管理就成了灾难（ n 人需要 $n(n-1)/2$ 把密钥）。

引入一个可信第三方（TTP）就能简化——但 TTP 每次都要参与，容易成瓶颈。

Kerberos 就是这个思路的一个成熟实现。

Kerberos 概述

Kerberos：应用于 TCP/IP 网络的可信第三方鉴别协议。

最初是麻省理工学院为 Athena 项目开发的模型，基于对称密码。

它有一个保存所有客户及其共享密钥的数据库，能：

- 产生消息，向一个实体证实另一个实体的身份
- 产生会话密钥，加密双方之间的通信

用生活场景理解 Kerberos

想象一个大型办公楼的门禁系统：

1. **进门先到前台**（AS，身份认证服务器）：出示你的工牌，前台核对无误后给你一张"通行证"（TGT，票据许可票据）
2. **想去某个部门**，拿着通行证到"票据处"（TGS，票据发放服务器）换一张"该部门的门卡"（ST，服务票据）
3. **拿门卡进部门**（访问服务器），部门核对门卡后提供服务

好处：你不用在每个部门都登记一次，只要前台认你就行。一次认证，处处通行。

而且：那张"通行证"是有时效的，过期作废——降低了被盗用的风险。

两类凭证

① **票据**（TGT，Ticket Granting Ticket）：用作向服务器提供身份鉴别，包括

$$T_{c,tgs} = \{ c, tgs, addr, realm, t, K_{c,tgs} \}$$



② **鉴别码 (Authenticator)**：客户每次需要使用服务器上的服务时，都要产生一个鉴别码。

$$A_{c,s} = \{ c, s, realm, t \}$$

为什么要有两类？ TGT 是“长期凭证”（可以重复用一段时间），Authenticator 是“每次现做的一次性凭证”。

为什么要现做？防重放——和 3.5 节 SKEY 的思路一样，一次性才安全。

六步协议（这是考点）

- ① Client → AS : c, tgs, n
- ② AS → Client : $\{K_{c,tgs}, n\}K_c, \{T_{c,tgs}\}K_{tgs}$
- ③ Client → TGS : $\{A_{c,s}\}K_{c,tgs}, \{T_{c,tgs}\}K_{tgs}, s, n$
- ④ TGS → Client : $\{K_{c,s}, n\}K_{c,tgs}, \{T_{c,s}\}K_s$
- ⑤ Client → Server : $\{A_{c,s}\}K_{c,s}, \{T_{c,s}\}K_s$
- ⑥ Server → Client : $\{m, t, k\}K_{c,s}$

三步概括：

1. 获得**票据许可票据 TGT** —— 第 ①② 步
2. 获得**服务票据 ST** —— 第 ③④ 步
3. **请求服务**（双向验证）—— 第 ⑤⑥ 步

看那些大括号：为什么每样东西都被加密了

注意协议里到处都是 $\{...\}K$ 的写法，意思是“用 K 加密后的内容”。

背后的逻辑很简单：在公开网络里传输，任何东西都必须加密。而且必须用“只有目标方知道的那把密钥”来加密。

记号	用谁的密钥	谁能解开
K_c	客户与 AS 的长期共享密钥	只有客户（和 AS）
K_{tgs}	TGS 的长期密钥	只有 TGS——所以客户拿到 TGT 也打不开，只能转交
$K_{c,tgs}$	客户与 TGS 的会话密钥	客户和 TGS
$K_{c,s}$	客户与服务间的会话密钥	客户和服务端

妙处：客户拿到的 TGT 是用 **TGS 的密钥加密的**——他自己打不开，只能原样交给 TGS。这就防止了客户伪造或篡改票据。

▼ 自测：Kerberos 里为什么要分成 AS 和 TGS 两个服务器？

为了"一次认证，多次使用"——避免用户每次访问服务都要重新输密码。

如果只有一个服务器：你每访问一个服务，都要向它证明一次身份（可能要输密码、可能它要承担全部认证压力）。

Kerberos 拆成两个阶段：

- **AS（认证服务器）**：只负责"验证你是你"，发一张**通行证（TGT）**给你。这一步需要你的长期密钥（相当于密码）。
- **TGS（票据发放服务器）**：你拿着 TGT 来找它，它给你发**某个具体服务的门卡（ST）**。这一步用的是会话密钥，**不需要再输密码**。

好处：

1. 长期密钥（相当于密码）只用一次，**减少泄露风险**
2. 认证服务器和授权服务器分开，**各司其职、压力分散**
3. TGT 有时效性，**过期作废**

这就是为什么它叫"票据"系统——像剧院一样，先在售票处拿票，再拿票进场。**不用每次进场都重新买票。**

7.3 数字证书：给公钥发一张身份证

从一个很常见的现象说起

你用不同浏览器访问同一个银行网站，有时会看到**两种不同的结果**：

- 一种是：地址栏显示"安全"，可以正常访问
- 另一种是：弹出警告"此网站的安全证书有问题"，问你要不要继续

为什么会有这个差别？ 这就是数字证书在起作用。

问题出在"公钥的可靠性"

回忆 3.2 节的中间人攻击：Alice 想拿到 Bob 的公钥，但她怎么知道网上那个公钥真的是 Bob 的？

课件列了三种获取公钥的途径：

- 从对方处获得
- 从可信的中央数据库获得
- 从自己的私人数据库获得

问题：如何保证可靠性？ 答案是——找一个大家都信的第三方来"担保"。

数字证书是什么

定义：简称为证书，由可信的**证书机关 CA** 将公钥的值**绑定**到持有对应私钥的个人、设备或服务的身份。

大白话：证书就是**公钥的身份证**。它说明"这个公钥确实属于张三"。

课件用"名片"和"身份证"来打比方——这就是它"身份鉴别"的用途。

X.509 标准

- ISO 鉴别框架：**X.509 标准**，**1988 年公布**
- 是国际电信联盟-电信（ITU-T）部分标准和国际标准化组织（ISO）的证书格式标准
- 定义了公开密钥的证书结构

- 是 **PKI**（公钥基础设施）的重要组成部分

证书里有那些内容

字段	说明
版本	V1~V3
用于签名的算法	签发者签名所用的算法
序列号	Internet 中唯一
颁发者信息	包括通用名、组织单位、组织和国家
有效期	生效与失效时间
证书所有者信息（公钥）	持有者身份 + 公钥值
扩展项	V2 中引入 ，解决 重复名称 问题
扩展 / 密钥用途等	V3 中引入
颁发者的签名	保证证书的 完整性 （防篡改）

核心内容其实就三样：**谁（身份）、他的公钥是什么、CA 的签名（证明这份证书是真的）。**

访问银行网站的完整流程

- ① 用户向服务器发起连接
- ② 服务器把**自己的证书**发过来
- ③ 浏览器检查本机证书存储区，看这个证书是不是“可信 CA”签发的
- ④ 如果 CA 可信，就把银行证书存下来，并建立安全会话

为什么“可信 CA 签发”就能信任？因为浏览器出厂时就内置了一份“**受信任根证书颁发机构**”列表——这些是各大操作系统/浏览器厂商审查过、认为可靠的机构。

所以那两种不同结果的原因就是：

- 浏览器认得这张证书的签发者 → 显示“安全”
- 签发者不在受信任列表里（或者证书过期、域名不符）→ 弹警告

证书链：信任是怎么层层传递的

课件提了三个问题：

- 通过证书对身份的信任程度？
- 谁可以作为唯一可信实体？
- 证书链可以有多长？

答案：一个唯一可信任的实体 → **可信任代理机构 CA**。

主机把“颁发者证书”放到主机的**受信任根证书颁发机构的证书存储区**，

将其指定为受信任的根颁发机构。

中间的或从属的证书颁发机构受到信任的条件是：

他们拥有受信任根证书颁发机构的有效证书路径。

大白话：像"介绍信"——你不认识这个人，但你信任的老板给他签了介绍信；而老板又是你信任的、更高级的人签发的……一路追到那个你真正信任的"根"。

这解决了"公钥怎么信任"的问题：不需要认识每个人，只需要信任少数几个"根 CA"。

7.3 小结

- **数字证书的结构：**颁发者、使用者的信息（公钥）、颁发者的签名……
- **数字证书的作用：**用来为实现安全信息交换**建立身份并创建信任**。
- **数字证书的应用：**受信任根证书的存储和中间证书的信任条件。

为什么它同时解决了 3.2 节的中间人攻击？ 因为攻击者没法伪造一份"银行证书"——他没有银行的私钥，签不出 CA 认可的那份签名。

全课结语

课件的最后问了一句："密码学及应用，学这个有什么用？"

它列了几个应用场景：**保密通信、访问控制、隐私保护、安全支付……**

回头看整门课，其实都在回答第 0 章提出的三个问题：

问题	对应的答案
怎么改写才安全？	第 1、2 章：古典密码 → 现代分组/序列密码 → 公钥密码
攻击者会怎么打？	每一章的安全性分析：从频率攻击到因子分解
密钥怎么管理？	第 3~7 章：协商、分发、共享、鉴别、托管、证书

而贯穿始终的那句话是：数据的安全基于密钥，而不是密码算法。

▼ 自测：数字证书为什么能防止中间人攻击？

因为攻击者伪造不出一份"CA 认可的证书"。

中间人攻击的做法是：把自己的公钥冒充成 Bob 的公钥发给 Alice。

但如果引入了证书：

- Bob 的公钥是由 **CA 签发的证书**里带的
- 证书上有 **CA 的签名**——这个签名只有 CA 能做（CA 有自己的私钥）
- 攻击者想伪造，就得伪造 CA 的签名 → **他没有 CA 的私钥，做不到**

所以信任链变成了：

Alice 信任 CA（因为浏览器内置了受信任根证书）

↓

CA 用签名担保"这个公钥确实是 Bob 的"

↓

所以 Alice 可以放心地相信这个公钥

注意一个细节：这里用的是"数字签名"（抗伪造、抗篡改），正好对应 3.4 节讲的签名协议。所以课件在第 3.4 节的预习思考里问："数字证书可以较好地防止中间人攻击，数字证书中的数字签名是如何防止篡改的？"——答案就是散列函数的抗碰撞性 + 签名的不可伪造性。

练习与答疑

A. 手算练习册（带分步答案）

怎么练

密码学的手算题都很小（模 26、模 17 这种），拿纸笔算两三个就通了。
下面 10 道题按难度排的，前 4 道是基础，后 6 道是综合。建议先自己做，卡住了再点开看。

第 1 题 约简

求 -200 模 98 的约简。

▼ 看答案

$$-200 + 98 = -102 \quad (\text{还是负, 继续})$$
$$-102 + 98 = -4 \quad (\text{还是负, 继续})$$
$$-4 + 98 = 94 \quad (\text{落在 } 0\sim97 \text{ 了})$$

答：94 验算： $94 - (-200) = 294 = 3 \times 98 \checkmark$
窍门：也可以一步到位 —— $-200 + 3 \times 98 = 94$ 。做法是“看能加几个 98 才变成正数”。

第 2 题 求逆元 (Euclid)

求 $101^{-1} \% 234$ 。

▼ 看答案

第一步：验互素 + 辗转相除

$$234 - 2 \times 101 = 32$$

$$101 - 3 \times 32 = 5$$

$$32 - 6 \times 5 = 2$$

$$5 - 2 \times 2 = 1 \quad \leftarrow \text{到 } 1 \text{ 了}$$

$$2 - 2 \times 1 = 0$$

最后非零余数是 $1 \implies \gcd(101, 234) = 1 \checkmark$ 逆元存在。

第二步：回代

$$1 = 5 - 2 \times 2$$

$$= 5 - 2(32 - 6 \times 5) = 13 \times 5 - 2 \times 32$$

$$= 13(101 - 3 \times 32) - 2 \times 32 = 13 \times 101 - 41 \times 32$$

$$= 13 \times 101 - 41(234 - 2 \times 101) = 95 \times 101 - 41 \times 234$$

第三步：读系数 101 的系数是 95。

答：95 验算： $101 \times 95 = 9595 = 41 \times 234 + 1 \checkmark$

第 3 题 求欧拉函数

求 $\varphi(18)$ 、 $\varphi(155)$ 、 $\varphi(9)$ 。

▼ 看答案

$\varphi(18)$: $18 = 2 \times 3^2$

$$\varphi(18) = (2-1) \times 2^0 \times (3-1) \times 3^1 = 1 \times 1 \times 2 \times 3 = 6$$

验证: 1~18 里与 18 互素的是 1,5,7,11,13,17, 共 6 个 ✓

$\varphi(155)$: $155 = 5 \times 31$

$$\varphi(155) = (5-1) \times 5^0 \times (31-1) \times 31^0 = 4 \times 1 \times 30 \times 1 = 120$$

$\varphi(9)$: $9 = 3^2$

$$\varphi(9) = (3-1) \times 3^1 = 2 \times 3 = 6$$

验证: 1~9 里与 9 互素的是 1,2,4,5,7,8, 共 6 个 ✓

提醒: 分解后每个 p^e 都要单独套公式, p^{e-1} 千万别漏。

第 4 题 降指数 (欧拉定理)

算 $249^{197} \% 199$ 。

▼ 看答案

199 是素数, $\varphi(199) = 198$ 。由费马小定理: $249^{198} \equiv 1 \pmod{199}$ 。

思路: 指数 197 只差 1 就到 198 了, 所以反过来用:

$$249^{198} = 249 \times 249^{197} \equiv 1$$

$$\Rightarrow 249^{197} \equiv 249^{-1} \pmod{199} \quad \leftarrow \text{变成求逆元了}$$

先约简: $249 \equiv 50 \pmod{199}$ 。求 $50^{-1} \pmod{199}$:

$$199 - 3 \times 50 = 49$$

$$50 - 1 \times 49 = 1 \quad \rightarrow \quad 1 = 50 - 49 = 50 - (199 - 3 \times 50) = 4 \times 50 - 199$$

所以 $50^{-1} = 4$ 。

答: 4 验算: $249 \times 4 = 996 = 5 \times 199 + 1$ ✓

这道题的信号: "指数差 1"往往意味着要转成求逆元。

第 5 题 验证原根

验证 3 是否是模 41 的本原根。

▼ 看答案

41 是素数， $\varphi(41) = 40$ 。分解： $40 = 2^3 \times 5$ ，素因子是 2 和 5。

只需要验两个指数： $40/2 = 20$ 和 $40/5 = 8$ 。

算 $3^8 \bmod 41$ （逐步平方）：

$$3^2 = 9$$

$$3^4 = 9^2 = 81 \equiv 81 - 41 = 40 \equiv -1 \pmod{41}$$

$$3^8 = (3^4)^2 \equiv (-1)^2 = 1 \pmod{41}$$

$3^8 \equiv 1$ ，违反条件（只要有一个等于 1 就否决）。

答：3 不是模 41 的本原根。

补充： $3^4 \equiv -1$ 、 $3^8 \equiv 1 \implies 3$ 的阶是 8，而 $8 \nmid 40$ ✓ 符合“非原根的阶整除 $\varphi(n)$ ”。

提醒：除数用素因子（2 和 5），不是所有因子。

第 6 题 仿射密码已知明文攻击

已知 $E_{a,b}(7) = 1$ 且 $E_{a,b}(12) = 4$ ，求解密密钥。

▼ 看答案

第一步：列方程，两式相减消 b

$$7a + b \equiv 1 \pmod{26} \quad (1)$$

$$12a + b \equiv 4 \pmod{26} \quad (2)$$

$$(2) - (1): 5a \equiv 3 \pmod{26}$$

第二步：求 a 需要 5^{-1} ：

$$26 - 5 \times 5 = 1 \rightarrow 1 = (-5) \times 5 + 1 \times 26 \Rightarrow 5^{-1} = -5 = 21$$

$$a = 21 \times 3 = 63 \equiv 11 \pmod{26} \quad (\gcd(11, 26) = 1 \checkmark \text{ 合法})$$

第三步：求 b

$$b = (1 - 11 \times 7) \% 26 = -76 \% 26 = -76 + 78 = 2$$

所以加密密钥是 $(11, 2)$ 。

第四步：转成解密密钥（题目问的是这个！）

$$\text{解密密钥} = (a^{-1}, -a^{-1}b)$$

$$\text{求 } 11^{-1} \pmod{26}:$$

$$26 - 2 \times 11 = 4$$

$$11 - 2 \times 4 = 3$$

$$4 - 1 \times 3 = 1 \rightarrow 1 = 4 - 3 = 4 - (11 - 2 \times 4) = 3 \times 4 - 11 = 3(26 - 2 \times 11) - 11 = 3 \times 26 - 7 \times 11$$

$$\Rightarrow 11^{-1} = -7 = 19$$

$$-a^{-1}b = -19 \times 2 = -38 \equiv -38 + 52 = 14$$

答：解密密钥 $(19, 14)$ ，即 $x = (19y + 14) \% 26$

验算： $D(1) = 33 \% 26 = 7 \checkmark$ ； $D(4) = 90 \% 26 = 12 \checkmark$

第 7 题 中国剩余定理

解 $x \equiv 2 \pmod{3}$ ， $x \equiv 3 \pmod{5}$ ， $x \equiv 2 \pmod{7}$ 。

▼ 看答案

$$M = 3 \times 5 \times 7 = 105$$

$$W_1 = 105/3 = 35 \quad W_2 = 105/5 = 21 \quad W_3 = 105/7 = 15$$

$$T_1 = 35^{-1} \bmod 3 = (35 \bmod 3)^{-1} = 2^{-1} = 2 \quad (2 \times 2 = 4 \equiv 1)$$

$$T_2 = 21^{-1} \bmod 5 = (21 \bmod 5)^{-1} = 1^{-1} = 1$$

$$T_3 = 15^{-1} \bmod 7 = (15 \bmod 7)^{-1} = 1^{-1} = 1$$

$$x \equiv 2 \times 35 \times 2 + 3 \times 21 \times 1 + 2 \times 15 \times 1$$

$$= 140 + 63 + 30 = 233$$

$$= 233 - 2 \times 105 = 23 \pmod{105}$$

答：23

验算： $23 \bmod 3 = 2 \checkmark$ ； $23 \bmod 5 = 3 \checkmark$ ； $23 \bmod 7 = 2 \checkmark$

小技巧：算 T_i 时先把 W_i 对 m_i 取模，往往能简化很多（ $35 \bmod 3 = 2$ ，比直接算 35^{-1} 容易）。

第 8 题 RSA 计算

取 $p=3$, $q=5$, $e=3$, 加密 $m=7$, 然后解密验证。

▼ 看答案

① $n = 3 \times 5 = 15$

② $\varphi(n) = (3-1)(5-1) = 2 \times 4 = 8$

③ 检查 $\gcd(e, \varphi(n)) = \gcd(3, 8) = 1 \checkmark$

④ $d = e^{-1} \bmod \varphi(n) = 3^{-1} \bmod 8 = 3$ (因为 $3 \times 3 = 9 \equiv 1 \bmod 8$)

⑤ 公钥 $(e, n) = (3, 15)$, 私钥 $d = 3$

⑥ 加密: $c = m^e \bmod n = 7^3 \bmod 15 = 343 \bmod 15$

$$343 \div 15 = 22 \text{ 余 } 13 \implies c = 13$$

⑦ 解密: $m = c^d \bmod n = 13^3 \bmod 15 = 2197 \bmod 15$

$$2197 \div 15 = 146 \text{ 余 } 7 \implies m = 7 \checkmark \text{ 还原成功}$$

第 9 题 置换的阶

S_5 中元素的最大阶是多少? S_{10} 呢?

▼ 看答案

思路：置换的阶 = 各不相交循环长度的**最小公倍数**。而各循环长度之和必须等于 n 。

S_5 (把 5 拆成几段，算 lcm)：

拆法	5	4+1	3+2	3+1+1	2+2+1
lcm	5	4	6	3	2

S_5 最大阶 = 6 (形如 (1 2 3)(4 5))

S_{10} (试主要拆法)：

拆法	5+3+2	5+4+1	7+3	4+3+2+1
lcm	30	20	21	12

S_{10} 最大阶 = 30 (形如 (1 2 3 4 5)(6 7 8)(9 10))

为什么要凑互素？因为 lcm 只有在互素时才等于乘积。5、3、2 两两互素，乘积 30 一点不浪费；而 6+4 因为 $\gcd(6,4)=2$ ，乘积 24 只能取到 12，白白浪费了。

第 10 题 完美快速洗牌

确定一副 12 张牌的完美快速洗牌的循环分解。

▼ 看答案

第 1 步：确定参数 $2n = 12 \implies$ 模数 $= 2n+1 = 13$ ，规则 $f(x) = (2x) \bmod 13$

第 2 步：从 1 开始迭代

$1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \equiv 3 \rightarrow 6 \rightarrow 12 \rightarrow 24 \equiv 11 \rightarrow 22 \equiv 9 \rightarrow 18 \equiv 5 \rightarrow 10 \rightarrow 20 \equiv 7 \rightarrow 14 \equiv 1$

第 3 步：写成循环

(1 2 4 8 3 6 12 11 9 5 10 7)

这一趟把 12 个位置全串完了，所以是一个 12 阶循环，洗牌次数 = 12。

两个易错点：

- 模数是 13，不是 12。这是 $2n$ 张牌，模数是 $2n+1$ 。
- 中间会出现比 12 大的数 (16、24、22、18、20、14)。这些只是“乘 2 之后的中间量”，取模后必然回到 1~12，不是错误。

对比 8 张牌 (课件例)：模 9，循环分解是 (1 2 4 8 7 5)(3 6)，有两段，所以要取 $\text{lcm}(6,2) = 6$ 次。

自测：如果 10 道题你都会做了

那你已经掌握了这门课所有的手算能力。剩下的知识都是理解性的 (概念、协议设计思路)，不需要算。

B. 常见误解 20 问

用法

这 20 个都是“看着像对、其实是错”的说法。先自己判断对错，再看解释。

1. “算法保密就够了，密钥公开也没关系。”

错

反了。现代密码的原则是：算法公开，密钥保密。

因为算法根本藏不住（装到成千上万台机器上），而且公开还能让全世界帮你找漏洞。而密钥只是一小串数字，好管得多。

记住香农那句话：**数据的安全基于密钥，而不是密码算法。**

2. “只要算法足够复杂，就绝对破不了。”

错

课件明确说：“如果攻击者拥有无限资源，任何密码系统都是可以破译的。”

因为攻击者总能穷举——把所有密钥都试一遍。所以世界上没有“绝对安全”，只有“破它的代价 > 收益”，这叫**计算安全**。

3. “对称密码和公钥密码都是加密，区别不大。”

错

区别很大，而且各有专长：

- **对称密码**：快，但密钥管理复杂 → 用来**加密消息**
- **公钥密码**：慢，但密钥管理简单 → 用来**协商密钥、数字签名**

现实中是**混合使用**：用公钥协商会话密钥，用对称密码加密消息（这就是“数字信封”）。

4. “算出负数就直接写负数就行了。”

错

密码学里约简的结果必须是非负的。所以 -2 要写成 24 （模 26 时）， -13 要写成 87 （模 100 时）。

这条规则你会在求逆元、RSA 计算、坏种子公式里反复用到。**看到负数就条件反射加上模数。**

5. “模运算里可以两边同时除以一个数。”

错

模运算**没有除法**。 $5a \equiv 3 \pmod{26}$ 不能写成 $a = 3/5$ 。

正确做法：**两边乘以 5 的乘法逆元**。 $a \equiv 5^{-1} \times 3 \pmod{26}$ 。

而且想“约掉”一个因数时，**必须先验证它与模数互素**，否则会出错。

6. “求逆元只要算到最大公因子就行了。”

错

辗转相除只给出最大公因子，**回代**才给出逆元。

Euclid 算法分两步：① 辗转相除（左列）→ ② 回代（右列）。第二步不能省。

7. “仿射密码里 a 随便选一个数就行。”

错

a 必须与 26 互素。

反例：如果 $a = 2$ ，那么 $x=0$ 和 $x=13$ 加密后都是 0——多个明文挤到同一个密文，信息丢失，解不回来。

原因：只有互素才有乘法逆元，而没有 a^{-1} 就无法解密。

8. "加密密钥和解密密钥是同一组数。"

错

仿射密码：加密是 (a, b) ，解密是 $(a^{-1}, -a^{-1}b)$ ——两组不同的数。课件里加密是 $(9,3)$ ，解密是 $(3,17)$ 。

RSA：加密用公钥 (e,n) ，签名用私钥 d 。

考试如果问"解密密钥"，填成加密的那组就错了。

9. "移位密码的密钥空间是 26。"

错

是 25。因为 k 取 1~25 ($k=0$ 等于没加密，不算)。

同理，仿射密码的密钥空间是 $12 \times 26 - 1 = 311$ (减的那个 1 也是"什么都不做"的情况)。

10. "Vigenère 里同一个字母变成不同密文，是因为密钥循环使用。"

错

说反了。真正的原因是**密钥里不同位置的字母不一样** (所以移位量不同)。

而"密钥循环"导致的恰恰是**相反**的效果——相隔密钥长度倍数的位置用同一个密钥字母，所以"相同变成相同"。这就是 Vigenère 的周期性死穴。

11. "换位密码会把字母改掉。"

错

换位只改位置，不改内容。所以它的标志是"明文和密文一样长"，而且**字母完全相同**。

判断方法：**数字母**。是同一批 → 换位；不是 → 代替。

(注意陷阱：computer → retupmoc 看着很陌生，但字母完全一样，是换位。)

12. "OTP 是最强的密码，所以应该优先使用。"

错

OTP 确实**理论上唯一不可破**，但它**用不起来**：

- 密钥要和明文一样长 (传 1 万字的消息就要 1 万字的密钥)
- 密钥只能用一次 (用完就扔，下次重新生成和传递)

所以现实中所有系统用的都是"计算安全"，而不是"理论安全"。

13. "序列密码的密钥流可以重复使用，只要密钥保密就行。"

错，而且这是最危险的一条

因为 $C = P \oplus k$ ，只要攻击者知道 P 和 C 中的任何一个，就能算出 k ；反之知道 k 就能算出 P 。

一旦重复使用密钥流，攻击者只要知道一条消息的内容，就能逐位推出所有用同一密钥流加密的消息。

序列密码的铁律：一次一密钥流，用完作废。

14. "ECB 模式最大的问题是速度慢。"

错

ECB 是最快的模式（每块独立、可以并行）。它的真正问题是：

- 易受重放攻击——密文块可以任意拼接替换（课件那个银行转账的例子）
- 泄露数据模式——相同的明文块得到相同的密文块

所以它的定位是"随机存取方便"，而不是"安全"。

15. "CBC 的 IV 需要保密，不能让别人知道。"

错

课件原话："IV 不需保密，可以以明文形式与密文一起传送。"

IV 的作用只是"把每个消息唯一化"（让相同的明文产生不同的密文），它本身不承担保密职责。解密时 IV 也只是被异或掉，泄露 IV 不泄露密钥。

（严格说：IV 不需要保密，但需要不可预测。）

16. "CFB 和 OFB 差不多，可以互换使用。"

错

关键区别在反馈量的来源：

- CFB 反馈的是上一块的密文 → 必须按顺序算，不能预计算
- OFB 反馈的是上一块的输出（只和密钥、IV 有关）→ 可以提前把整条密钥流算好

所以 OFB 相当于把分组密码"当序列密码用"，适合低延迟场景。

17. "序列密码的密钥流只要周期足够长就安全了。"

不完整

周期长只是必要条件。同余发生器和 LFSR 都是纯线性算法，即使周期做到 $p-1$ 或 2^n-1 ，攻击者拿到连续几项就能立方程解出参数。

所以课件说同余发生器"算法不具备随机性"；而 LFSR 主要靠硬件工艺保密 + 多路组合来补安全性，不是靠算法本身。

18. "数字签名就是把消息加密一下，和加密是一回事。"

错

虽然 RSA 里都用到"幂运算"，但方向正好相反：

	加密（保密）	签名（证明身份）
第一步	用对方的公钥	用自己的私钥
第二步	用自己的私钥	用自己的公钥

而且课件强调："保密通信的密钥对不同于数字签名的密钥对。" 实际应用中应该用两对密钥——否则会出"密钥使用不当"的攻击。

19. "SKEY 协议里的哈希链是正着用的。"

错，是反着用的

Alice 手里持有一条链 $x_0 \sim x_{99}$ ，但她登录时从 x_{99} 开始，往 x_{98} 、 x_{97} 的方向走——也就是"倒着用"。

为什么要倒着用？因为散列函数是单向的：正向容易，反向难。

- 倒着用：攻击者截获 x_{99} 后想算出下一个 (x_{98}) → 需要反推哈希 → 做不到 ✓
- 正着用：攻击者截获 x_{99} 后能算出 $x_{100} = h(x_{99})$ → 直接冒充 Alice ✗

课件作业题就是这个："如果采用正向使用是否可行，安全性如何？"——答案是不可行。

20. "位承诺协议只需要一个随机数。"

错

它要同时满足两个方向相反的需求：

- 绑定承诺者（说了不能改）→ 需要有些东西提前公开
- 挡住接收者（现在不能偷看）→ 需要有些东西保持保密

所以：

- 哈希方案用两个随机数： r_1 公开（防篡改）、 r_2 保密（防猜测）
- 对称密码方案： r 公开（防篡改）、 k 保密（用于验证）

为什么"防猜测"这么重要？因为承诺的位只有 0 和 1 两种可能——没有保密参数的话，接收者两次哈希就能试出来。

C. 概念地图

用法

这是整门课的一张"总图"。如果哪块的知识点记混了，回到这里看看它属于哪个分支。

C1. 密码算法地图

密码算法

├── 古典密码（算法保密，密钥保密）

| ├── 代替密码（换内容）

| | ├── 单表代替：移位密码（密钥空间 25）

| | | 仿射密码（311）

| | └── 多表代替：Vigenère (26^m)

| └── 换位密码（换位置）：列换位、矩阵换位

|

└── 现代密码（算法公开，密钥保密）

 ├── 对称密码（快，密钥管理复杂）

 ├── 分组密码：DES（64/56/16）、IDEA（64/128/8）、AES（128/128,192,256/10,12,14）

 └── 序列密码：RC4（字节流）、同余发生器（数据流）、LFSR（位流）

|

|— 公钥密码 (慢, 密钥管理简单)

|

|— RSA —— 因子分解难题

|

|— ElGamal —— 离散对数难题

|

|— ECC —— 椭圆曲线离散对数

|

|— Knapsack —— 子集和问题 (已破)

|

|— 数字摘要: MD5 (128)、SHA1 (160)、SHA256 (256)、SM3

|

|— 数字签名: RSA 签名、ElGamal 签名、DSA、Schnorr、ECDSA

|

|— 国密: SM1 (对称/芯片)、SM2 (公钥)、SM3 (摘要)、SM4 (分组)、SM9 (标识)

C2. 数学基础地图

第 1.4 节的六个"零件"

|— 约简: 取余数, 负数加模数变正 → 全书处处用

|— 同余: 相差 m 的整数倍 = 同一个数 → 全书处处用

|— Euclid / 逆元: 凑出 $ax + by = 1$, a 就是逆元

|— 用在: 仿射密码解密、RSA 算私钥、坏种子公式

|— 群 / 剩余系:

|— Z_n (完全剩余系, n 个) → 配加法是群

|— Z_n^* (简化剩余系, $\varphi(n)$ 个) → 配乘法是群

|— 欧拉函数 $\varphi(n)$: 数个数

|— $\varphi(p) = p - 1$

|— $\varphi(p^e) = (p-1) \cdot p^{e-1}$

└— **欧拉定理**: $x^{\varphi(n)} \equiv 1 \pmod{n} \rightarrow$ 用来降指数

| └— 特例: **费马小定理** $x^{p-1} \equiv 1 \pmod{p}$

└— **阶**: 自乘多少次第一次回到 1

| └— 性质: 阶一定整除 $\varphi(n)$

└— **原根**: 阶 = $\varphi(n)$ 的元素 (转一圈走遍全圈)

| └— 用在: 同余发生器的周期、LFSR 的本原多项式

└— **快速指数**: 三元组法 (奇减一、偶平分)

└— **置换群 / 洗牌**:

| └— 不相交循环分解

| └— 置换的阶 = 各循环长度的 lcm

| └— 完美快速洗牌 $f(x) = (2x) \bmod (2n+1)$

C3. 协议地图

基本的密码协议 (第 3 章)

└— 保密通信: 古典 $\rightarrow$ 对称+TTP $\rightarrow$ 公钥 $\rightarrow$ **混合 (数字信封)**

└— 密钥协商: 公钥 / 对称+KDC / **DH** / EKE / 双线性对

└— 数字签名: 对称+TTP / 公钥 / **hash+公钥** / 带加密 / 签名者参与验证

└— 身份鉴别: PAP $\rightarrow$ hash存储 $\rightarrow$ 加密 $\rightarrow$ **SKEY (哈希链)** / 挑战应答

└— 秘密共享: 简单分割 / 加密封割 / **Shamir 门限** / Asmuth-Bloom / DH 托管

中级的密码协议 (第 4 章)

└— 三次传输: Shamir (不协商密钥)、匿名密钥分配、智力扑克、不经意传输 OT

└— 安全多方计算: **盲化 + 去盲化** (平均薪资、离散对数、姚氏百万富翁)

└— 密钥分发: 传统 $O(k)$ vs **基于 CRT 的 $O(1)$**

└— 位承诺: 单向函数方案 / 对称密码方案

高级的密码协议（第 5 章）

- └— 零知识证明：洞穴协议（重复 n 次）
- └— 盲签名： $m' = m \times b$ （+ 分割选择法防骗）
- └— 同时签约：带仲裁者 / 概率递增 / 密码技术
- └— 匿名投票：协议 I $\rightarrow$ 协议 II $\rightarrow$ **匿名投票（盲签名）**
- └— 匿名通信：代理 / **MIX** / Crowds / 广播
- └— 区块链：POW 记账 + 链式结构 + 五类密码技术

密钥管理（第 7 章）

- └— 密钥生命周期：生成 $\rightarrow$ 存储 $\rightarrow$ 传输 $\rightarrow$ 使用 $\rightarrow$ 更新 $\rightarrow$ 销毁
- └— Kerberos：AS + TGS, TGT + Authenticator，六步协议
- └— 数字证书：X.509、CA、证书链

C4. 一句话记住每个算法的“靠山”

算法 / 技术	靠什么撑住安全性
移位 / 仿射密码	密钥空间太小，其实不安全
Vigenère	密钥空间 26^m ，但被周期性打败
OTP	密钥与明文等长 + 只用一次（理论上不可破）
DES	56 位密钥太短，已被穷举攻破
IDEA / AES	足够长的密钥 + 足够的轮数
RC4	256 字节 S 盒的非线性洗牌
同余发生器 / LFSR	纯线性，其实不安全
MD5 / SHA	单向性 + 雪崩效应 + 抗碰撞性
RSA	因子分解难题
ElGamal / DSA	离散对数难题
ECC	椭圆曲线上的离散对数难题
DH	离散对数难题（但不防中间人）
数字证书	CA 的签名不可伪造
区块链	51% 算力 + 全网核对

最后一句话

这门课看起来东西很多，但核心其实就三句：

1. **算法公开、密钥保密**——这是现代密码的世界观。

2. 安全性都建在"正着算容易、反着算难"的数学难题上——RSA 靠因子分解，ElGamal 靠离散对数，摘要靠单向性。

3. 协议里最怕的不是算法被破，而是用法出错——共用模数、密钥重复、混淆加密和签名的密钥对，都是这类坑。

把这三句记住，再回头翻前面的具体内容，都会顺很多。