第3章　密码学基础

“一轮明月不孤单，
晚起朝落无埋怨。
吴刚吃惊尔登门，
嫦娥煮饭待客欢。”

Alice吟起诗来，Bob倒是比吴刚还吃惊，不过Bob可不是等闲之辈，立马看出端倪来。

“好的，一起吃饭！”Bob一语双关，又加了一句：“你是孔子学院毕业的？”

Alice哈哈大笑，“你答应得这么爽快，显然漏掉了半句话。”见Bob一脸蒙圈，Alice调皮地说：“友情提示，你只看到一条斜线，划个V字试试？”

无论是民间的谜语、藏头诗，还是军队的密电码，无论是娱乐、游戏，还是决定生死存亡的行动，都有隐藏信息的需要。加密与破译的对抗有许多惊心动魄的故事，有些被拍成了令人荡气回肠的电影。例如《U-571》和《模仿游戏》，表现的是同一个历史事件的前后两段，先是盟军巧妙地从轴心国潜艇上猎取恩尼格玛(Enigma，意为谜)密码机，然后是布莱切利庄园一群以“计算机和人工智能之父”艾伦·图灵(Alan Turing)为代表的密码专家，成功破译史上最强密码，最终挽救了千百万生命，拯救了世界。再例如《风语者》，印第安原住民纳瓦霍族人应征入伍，成为通信兵，他们用独特的方言来进行无线电通信，让敌方听到了也完全不知所云，技术上毫无破解之道。

3.1　密码学基本概念

信息加密(information encryption)作为保障消息安全传递的一种方式，其历史相当久远，可能要追溯到公元前2000年。虽然那个时代的技术与当代不可同日而语，但已具

备加密的概念和雏形。例如在畅销书《失落的密符》中有这样一系列的情节：追寻宝藏的教授偶然得到一串字母“YUOEMSTDIINHREKY”，但不知何意，束手无策中被线索指引到一幅壁画前，在不起眼的角落发现了画中的一个幻方(magic square)，于是豁然开朗。

幻方是一种 $n\times n$ 的格子，称为 n 阶幻方，要求将数字 $1\sim n^2$ 不重复、不遗漏地填入各个格子，使得每行、每列、每条对角线的数字之和(称为幻和)都相等。优秀的幻方还能做到四个象限的数字之和也相等，如图 3.1(a)和(b)所示分别为两名大数学家亲自设计的幻方：4 阶丢勒幻方(*Durer's magic square*)和 8 阶富兰克林幻方(*Franklin's magic square*)。n 阶幻方的幻和可以计算为 $\frac{n}{2}\times(n^2+1)$。

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Y	U	O	E
M	S	T	D
I	I	N	H
R	E	K	Y

(a) 丢勒4阶幻方与教授的密文

52	61	4	13	20	29	36	45
14	3	62	51	46	35	30	19
53	60	5	12	21	28	37	44
11	6	59	54	43	38	27	22
55	58	7	10	23	26	39	42
9	8	57	56	41	40	25	24
50	63	2	15	18	31	34	47
16	1	64	49	48	33	32	17

(b) 富兰克林8阶幻方

图 3.1　幻方示例

于是执着寻宝的教授将 16 个密文字母按顺序填到丢勒幻方中，再对照幻方中对应的数字，按从小到大的次序写下字母，一句话就出现了：“YOUR MIND IS THE KEY”(你的思想是关键)，其中 KEY(也有钥匙之意)与 Bob 的话一样也是个双关语。

早期的加密方法只用于字母等文字信息，以手工处理为主，一般采用“置换法”，例如以一定规律打乱字母次序、将字母替换为另一个字母、用数字来表示字母等，这类方法被称为古典加密法。受限于较弱的密码分析能力，这些技术在当时还是具有较强的先进性和安全性的。

然而，随着数据统计能力的提高，各种置换法共同存在的大漏洞逐渐显示出来，就是无论怎么改头换面，文章中各字母出现的频率（即字频）并没有改变。因此，破译者并不需要费劲猜测加密手段，也不需要拼命尝试，只要有足够长的密文，就可以根据密文字母的字频对照图 3.2 的数据，推测出各个字母的原形。例如，字母 e 在英文单词中很常用，字频最高；字母 t 位居次席，且作为首字母的概率遥遥领先，无疑是随处可见的定冠词“the”在起作用。虽然恩尼格密码机将古典加密法推向极致，通过快、中、慢三个转轮实现字母映射关系为每字母一变，复杂性大增，难以简单通过字频来分析，但本质上仍然属于替换法范畴，很难抵抗穷举排序等暴力攻击。

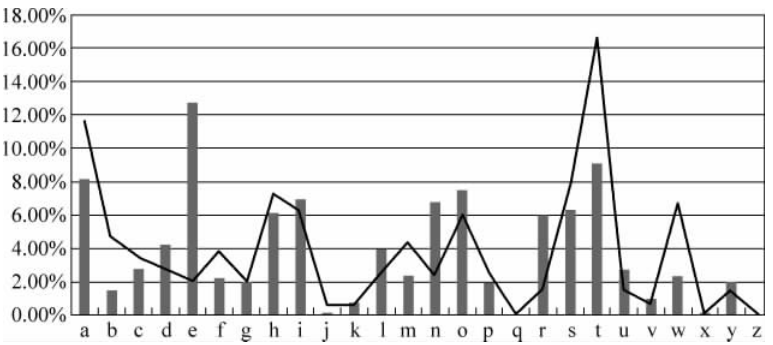

图 3.2　英文字母字频柱状图和首字母字频折线图

1946 年 2 月 14 日，随着世界上第一台计算机艾尼阿克（ENIAC）的诞生，密码分析技术跨入了新阶段，曾经威名远播的 Greece、Caesar、Playfair、Vigenere、Vernam、Hill 等古典加密法永远退出了历史舞台。同时伴随着高等数学研究不断取得新成果，信息加密领域开启了“数学＋电脑”的现代加密法时代，加密技术也升级成为密码科学。

现代密码技术几乎都是针对二进制数据，不论数据是文本、图片、音频、视频还是程序。如图 3.3 所示，信息加密的基本原理是：把明文（plaintext）用加密（encryption）方法结合密钥（key）生成保密的密文（cryptograph 或 ciphertext），只有使用正确的解密（decryption）方法结合解密密钥才能成功还原出明文。换句话说，如果运用的解密方法不符，特别是解密密钥不正确，即使采用高性能计算机，破译的难度依然极大。

信息加密“五要素”模型是原则性的框架，可以依不同技术而有所变化，例如，加密和解密密钥可以是不同数值，可以不使用密钥，也可以不支持解密。需要注意的是，所谓明文并非是“可读”的代名词，而只是一次加密过程的输入或一次解密过程的输出，已经加

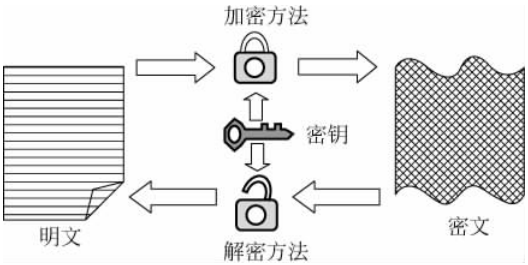

图 3.3　加密和解密原理图

密的密文也可以成为另一次加密的“明文”，即加密可以无限迭代、层层嵌套。多次加密的过程是一种栈式运算，后进行的加密应该先作解密。

现代加密法有三大类技术，如图 3.4 所示，包括：对称密钥加密、非对称密钥加密、单向函数加密。加密技术的合理运用可以产生三方面的作用。

- 保密性：通过改变原始信息的数据构成，使得信息即使遭窃取、截获、泄露，也难以获取原始信息，达到保护信息内容安全的目的。
- 完整性：敏锐发现信息内容发生的任何变化，如信息被篡改（或传输过程发生误码）、伪造，从而保障原始信息的“原封不动”。
- 确权性：鉴别并确定信息的归属方，一方面可以用来证明信息的真正拥有者（所有权），另一方面也可用以判定信息来源，防止抵赖。

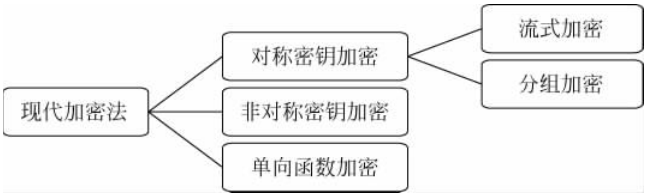

图 3.4　现代加密法分类

信息加密技术是一把“双刃剑”，既有防范自己的信息受到侵害的作用，又有投入成本上升、系统复杂性增大、信息传递延迟等副作用。同时，加密和破解永远是“矛尖还是盾固”的关系，一个在明处，一个在暗处，即便采用最先进的信息加密技术，也达不到绝对的安全，安全始终具有相对性。

（1）计算相对性——普通电脑难以破译的密码，高性能电脑或许就能轻松攻破，因为难度实际上取决于计算能力；还有可能研究出一种数学方法，可以极大地降低计算工

作量。

(2) 时间相对性——目前很安全的密码,随着时间的推移,安全性会衰减,因为在攻击者持续不断地尝试下,被攻破的概率将逐步上升。

(3) 价值相对性——信息的价值越高,受到攻击的可能性就越大;反之,假如破解成本远高于信息的价值,那么信息基本就是"高枕无忧"的。

正因为如此,应避免过度运用加密技术,而是应该根据信息对象属性、应用场景等各种因素来设计合理的保密方案。

3.2 对称密钥加密

对称密钥加密(symmetric key cryptography),又称为私钥加密、单钥加密,因同一个密钥既用于加密又用于解密而得名。

密钥实际上就是一串二进制数据。既然对称加密方法的密钥也要用于解密,那么密钥一定要被妥善保护好、绝不示人,这是其称为私钥的原因。又由于密文的合法接受者也需要这个私钥,所以如何在通信双方间安全地分享密钥就显得非常重要。

对称密钥加密是实现信息保密的主要手段,具有如下技术特点:

- 密钥是关键(Key is key)。现代加密技术的加密算法可以公布,加密代码也可以公开,只要保护好密钥,密文就是安全的。
- 密钥长度决定安全性。密钥越长则加密强度越大,因为穷举密钥几乎是尝试破解的唯一方式,那么密钥每增加 1 位,就可以给破解者的计算工作量增加 1 倍,密钥增加 1 字节,破解工作量就是原来的 256 倍,相当于原本 1 天就可破解,现在则需要将近 1 年。
- 对称加密算法被设计为具有很高的计算效率,可面向大量数据的加密,如文件、数据库、流媒体等,然而密钥的安全生成、安全分发、安全存储、安全使用需要较大的管理工作量。

对称密钥加密可分为流式加密和分组加密两大类,前者适用于流媒体应用,例如在线播放音乐或视频,后者应用范围更广,适用于绝大多数需要数据加密的场合。常用的对称密钥加密技术有 BLOWFISH、TEA、DES、AES、IDEA、SM4、RC4 等,本章选取其中具有代表性的 SM4 和 DES 来展开算法细节。

在比特币核心系统中并没有直接采用对称密钥加密技术，这应该与比特币提倡信息透明化有关，而且存储的信息也比较单一。但是在区块链扩展应用中，当需要对保存的私密信息进行保护时，对称密钥加密技术就有用武之地了。

3.2.1　分组加密技术原理

分组加密(block ciphering)是以一定长度的数据块为单位，对其进行整体变换，从明文块生成密文块，密文块与明文等长，一块数据加密完成后继续加密下一块，密文块也是依次执行解密。就加密算法本身而言，每一块数据的加密或解密运算都可以独立进行。

设分组大小为 nB(字节)。明文数据交付加密前，需要先进行数据填充(填充数据可为任意数值)，如图 3.5 所示，使总长度为 n 的整数倍。最后 4 字节为长度字段，用以表示有效数据长度，以便解密后完全恢复原始数据。

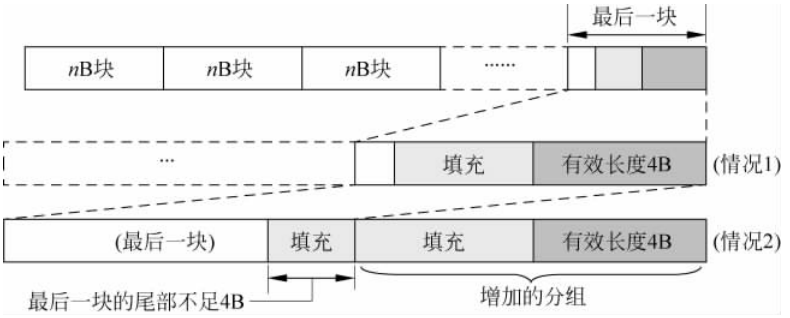

图 3.5　分组加密前的数据块准备

显然，对 nb(位)的分组，有 2^n个不同的明文分组和 2^n个不同的密文分组，相互间共有 $2^n!$ 个非奇异的映射(变换)关系。

不失一般性，设 $n=4$，如图 3.6 所示，4b 明文分组 $M=(M_3M_2M_1M_0)$共有 $2^4=16$ 种编码组合，编码器的 16 个输出分别为 $X_0 \sim X_f$，共有 $2^4!=20,922,789,888,000$ 种不同的映射关系。设对于一种特定的映射关系转换为 $Y_0 \sim Y_f$，可解码为 4b，输出 $C=(C_3C_2C_1C_0)$，C 即为明文 M 加密后的密文。例如，4b 明文为 1001(十六进制 9)，编码结果应为 X_9，在图 3.6 所示的映射关系下被转换为 Y_7，则可解码并输出密文 0111。在相同的映射关系下，若以 C 为密文输入、M 为明文输出，则为解密操作。

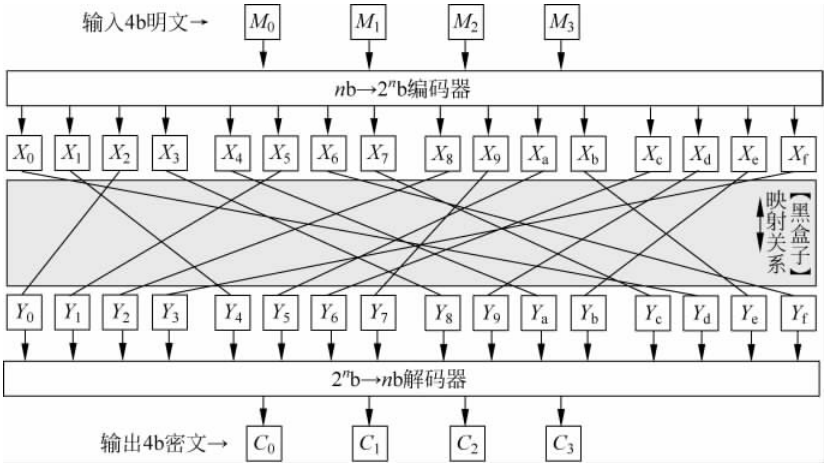

图 3.6 4b 分组加密变换示意图

“不对啊，4 位明文不是应该就只有 16 种编码组合，怎么会弄出 16!个映射关系呢?”Bob 有点不在状态，脑子一时转不过弯来。

乐于助人的 Alice 一如既往地耐心解释：“映射是从编码器的 16 个输出到解码器的 16 个输入之间的不同关系。对于例子中的映射关系，X_9 映射为 Y_7、X_2 映射为 Y_0，如果换一种映射关系，例如只把 Y_0 和 Y_1 上两根线调换位置，其他不变，那么 X_9 仍然映射为 Y_7，X_2 就映射为 Y_1 了。所以编码组合与映射关系是不同的概念。”

当映射关系改变时，同一个明文可输出不同的密文，这就相当于采用了不同的密钥而产生不同的加密结果。换句话说，映射关系就是密钥，每一种变换就是一个密钥。因此，映射关系需要像个“黑盒子”一样严密保护起来。

这个加密模型穷尽了所有的编码和映射关系，在技术上已经无法更好了，因此被称为是“理想分组加密”方法，保密性达到最佳。然而，理想归理想，该模型存在一个不容忽视的问题：密钥过长。如图 3.6 所示的映射关系可表示为如图 3.7 所示的编码对应表，每一种不同映射产生一张表，nb 明文总共可得 2^n! 张表。不失一般性，将明文编码统一为升序排列，密文一行所代表的就是密钥，则密钥长度为 $n\times 2^n$b。对于常用的 64b 分组，其密钥长度将达到惊人的 10^{21} b。过长的密钥对数据加密的计算和管理均会带来不利影响。

明文	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
密文	1101	0100	0000	1000	1010	0001	1111	1100	0010	0111	0101	1110	0110	1001	1011	0011

图 3.7　分组加密编码映射关系示意图

所以实际的分组加密算法设计采用的是理想分组加密的近似体制，相当于取不同映射关系的一个子集，以减少映射关系数为代价，换取较短的密钥长度，虽然牺牲了一部分保密性，但使加密方法更具有实用性。

根据信息论创始人香农（Claude Shannon）的加密理论，使用信息编码的混淆（confusion）和扩散（diffusion）方法，可以抵抗基于统计方法的密码分析（破解）。混淆可以尽可能使密文和密钥间的统计关系复杂化，防止密钥被发现；扩散则使明文的统计特征消散在密文中，可以让每个明文编码单元尽可能多地影响多个密文编码单元。

费斯托（Feistel）模型就是基于香农理论设计的对称密钥加密统一架构。各种对称密钥加密算法都是参照这一模型设计的。在 Feistel 模型中，数据运用代换（substitution）和置换（swap）操作进行处理，分别就是混淆和扩散的实际应用。代换采用函数运算方法，置换通过数据的交叉换位实现。

设：明文和密文为 $2w$b，密钥为 K。图 3.8 所示即为 Feistel 加密的算法结构。该模

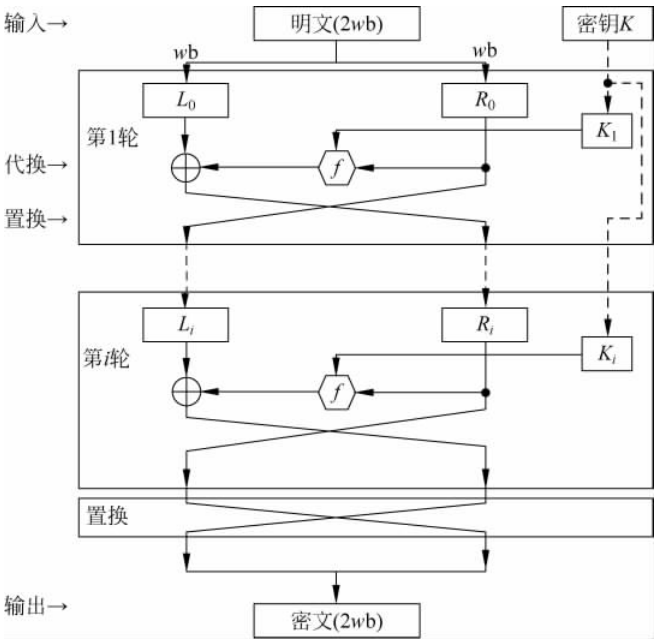

图 3.8　Feistel 加密模型模块组成结构示意图

型由 i 轮加密运算组成，每轮具有相似的运算方法，分别使用由密钥 K 生成的该轮次的子密钥 K_i。每轮 2wb 输入被分为 wb 的左半部分和 wb 的右半部分，进行代换运算，在输出前左、右两部分进行置换运算。算法结束前再进行一次左右置换运算，最终合并成 2wb 的密文。

解密过程与加密过程完全一致(注意不是逆向进行)，但解密时逆序使用子密钥。这一论断的正确性可被证明，如下：

不失一般性，设加密算法执行 16 轮，再设：加密时，明文为 $LE_0|RE_0$，输出密文为 $RE_{16}|LE_{16}$；解密时，密文为 $LD_0|RD_0$，输出密文为 $RD_{16}|LD_{16}$。

显然有：

$$LD_0 = RE_{16}, \quad RD_0 = LE_{16}$$

对于加密过程有：

$$LE_{16} = RE_{15}, \quad RE_{16} = LE_{15} \oplus f(RE_{15}, K_{16})$$

其中，$\oplus$ 表示按位异或运算(位相同得 0，相异得 1)，其“优秀”性质是一个数被另一个数异或两次就会还原。

按照解密过程第 1 轮算法有：

$$LD_0 = RE_{16}, \quad RD_0 = LE_{16}, \quad LD_1 = RD_0 = LE_{16} = RE_{15}$$

$$\begin{aligned} RD_1 &= LD_0 \oplus f(RD_0, K_{16}) = RE_{16} \oplus f(RE_{15}, K_{16}) \\ &= [LE_{15} \oplus f(RE_{15}, K_{16})] \oplus f(RE_{15}, K_{16}) = LE_{15} \end{aligned}$$

因此，解密第 1 轮输出 $LE_{15}|RE_{15}$，正是加密第 16 轮输入左右互换的值。

依此类推，解密第 16 轮：

$$LD_{16} = RE_0, \quad RD_{16} = LE_0$$

最终换位输出为 $LE_0|RE_0$，正是原始明文(证毕)。

Feistel 加密模型采用的 i 轮次运算又称为迭代(iteration)操作。迭代轮次数越多，密文信息的混淆和扩散越彻底，保密性就越强，但会有加解密速度越慢的“后遗症”，作为妥协，通常选择 16 轮迭代。此外，分组长度和密钥长度越长，安全性能也越好，但同样存在拖慢运算速度的弊端，常见的分组长度选择 64b 或 128b，密钥长度则会根据算法指标、密级要求来确定。

3.2.2 SM4 算法

SM4 算法(原名 SMS4)是基于 Feistel 模型的对称密钥分组加密算法，作为中国的第

一个商用密码国家标准，于 2006 年 1 月发布。

SM4 采用的数据分组长度和密钥长度均为 128b，数据处理单位为 8b 和 32b，进行 32 轮非线性迭代运算。

如图 3.9 所示，SM4 将 128b 的明文分割为 4 个 32b 分组块，采用 8 次循环，每次使用轮函数 f 进行 4 轮迭代（故总共为 32 轮迭代），每轮使用一个子密钥（轮密钥）。子密钥由 128b 的密钥经扩展函数 g 生成。最后，反向排列 32b 的 $X_{32}\sim X_{35}$，合并为 128b 的密文。

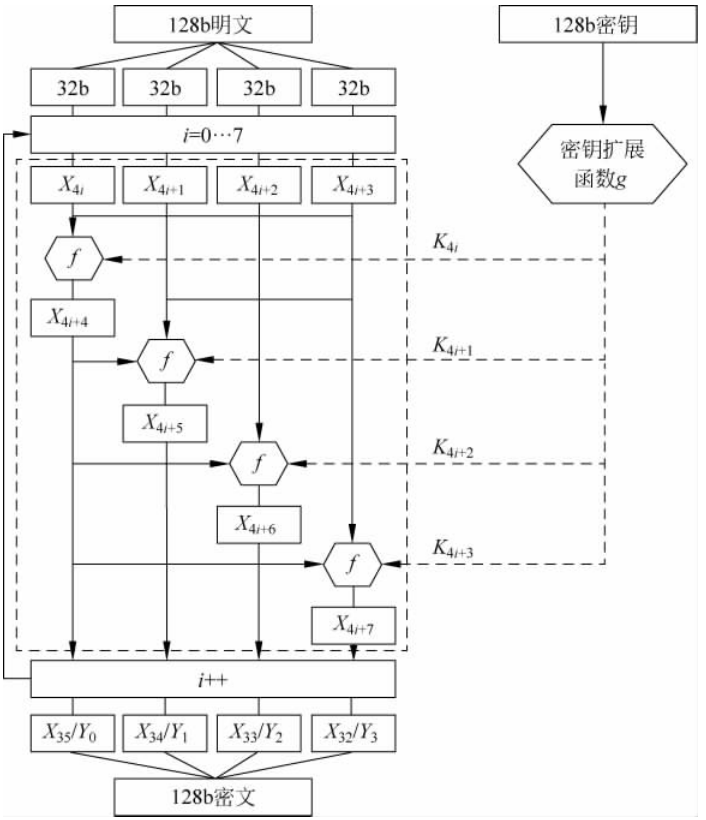

图 3.9 SM4 加密算法逻辑图

SM4 的加密运算过程可用如图 3.10 所示的等价变换图表示。

设 X_j 为 32b 寄存器，k_i 为轮密钥。SM4 每轮的迭代运算由按位异或（⊕）、循环左移（$\lll$）和变换函数 S 所组成，轮函数 f 定义如下：

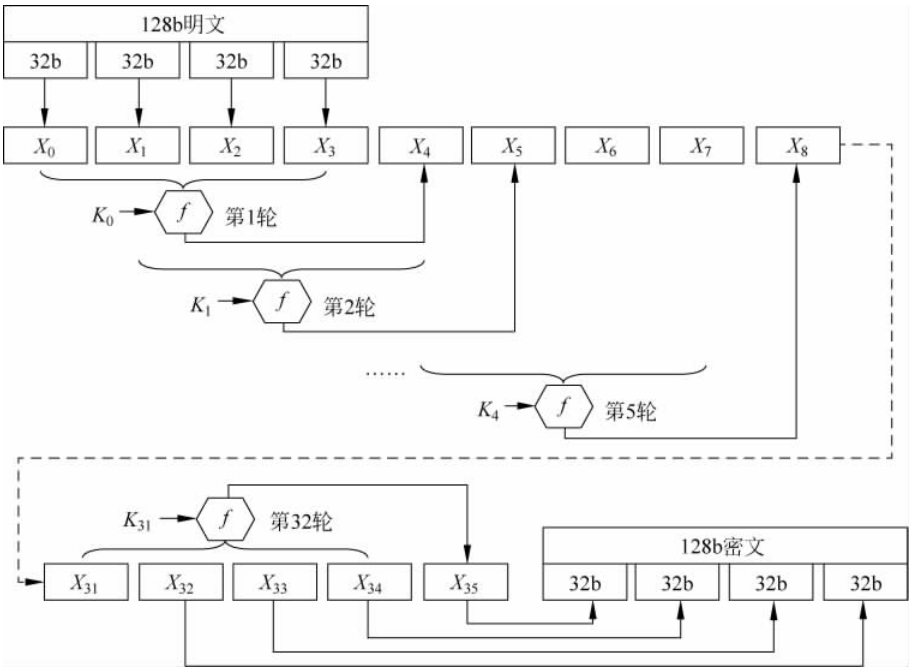

图 3.10 SM4 加密算法等价变换示意图

$$X_{i+4} = f(X_i, X_{i+1}, X_{i+2}, X_{i+3}, k_i)$$
$$= X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus k_i), i = 0, 1, \cdots, 31$$
$$T(x) = L(S(x))$$
$$L(x) = x \oplus (x <<< 2) \oplus (x <<< 10) \oplus (x <<< 18) \oplus (x <<< 24)$$

如图 3.11 所示，32b 的 S 函数由 4 个并列的 8b 单元的 S-box 变换所构成。S-box 采用查表选择法，使用如表 3.1 所示的置换选择表，属于一种非线性变换方式。表中的 256 个数值从 0x00 到 0xff，具有唯一性。设 S-box 的 8b 输入为十六进制 0xRC，S-box 以 R 为行号、C 为列号查表，所得的数据作为 S-box 的输出结果。例如，输入 0x2a 应查行号为 2、列号为 a 的数据，经 S-box 输出为 0x0b。

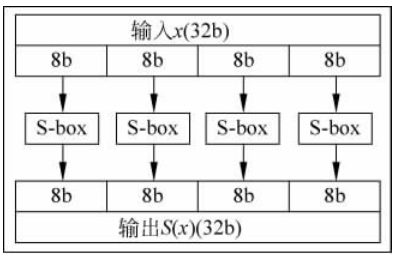

图 3.11 SM4 算法 S 函数原理图

表 3.1　SM4 算法 S-box 置换选择表

R \ C	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	d6	90	e9	fe	cc	e1	3d	b7	16	b6	14	c2	28	fb	2c	05
1	2b	67	9a	76	2a	be	04	c3	aa	44	13	26	49	86	06	99
2	9c	42	50	f4	91	ef	98	7a	33	54	0b	43	ed	cf	ac	62
3	e4	b3	1c	a9	c9	08	e8	95	80	df	94	fa	75	8f	3f	a6
4	47	07	a7	fc	f3	73	17	ba	83	59	3c	19	e6	85	4f	a8
5	68	6b	81	b2	71	64	da	8b	f8	eb	0f	4b	70	56	9d	35
6	1e	24	0e	5e	63	58	d1	a2	25	22	7c	3b	01	21	78	87
7	d4	00	46	57	9f	d3	27	52	4c	36	02	e7	a0	c4	c8	9e
8	ea	bf	8a	d2	40	c7	38	b5	a3	f7	f2	ce	f9	61	15	a1
9	e0	ae	5d	a4	9b	34	1a	55	ad	93	32	30	f5	8c	b1	e3
a	1d	f6	e2	2e	82	66	ca	60	c0	29	23	ab	0d	53	4e	6f
b	d5	db	37	45	de	fd	8e	2f	03	ff	6a	72	6d	6c	5b	51
c	8d	1b	af	92	bb	dd	bc	7f	11	d9	5c	41	1f	10	5a	d8
d	0a	c1	31	88	a5	cd	7b	bd	2d	74	d0	12	b8	e5	b4	b0
e	89	69	97	4a	0c	96	77	7e	65	b9	f1	09	c5	6e	c6	84
f	18	f0	7d	ec	3a	dc	4d	20	79	ee	5f	3e	d7	cb	39	48

SM4 加密算法每轮使用不同的 32b 子密钥，每个子密钥都是由 128b 原始密钥 K 经变换生成。设 4 个 32b 常数为 $FK_i(i=0,1,2,3)$：

$$FK_0 = 0\text{xA3B1BAC6}, \quad FK_1 = 0\text{x56AA3350},$$

$$FK_2 = 0\text{x677D9197}, \quad FK_3 = 0\text{xB27022DC}$$

又设 32b 参数为 $CK_i(i=0,1,2,\cdots,31)$，定义：$CK_i = < ck[i][0]|ck[i][1]|ck[i][2]|ck[i][3] >$，即 CK_i 是由 4 个一组的 8b 数值 $ck[i][j](j=0,1,2,3)$ 拼接而成，而 $ck[i][j]$ 可根据其下标进行取值：$ck[i][j] = (4i+j)\times 7 \pmod{256}$。由此可得 32 个 CK_i 参数为：

```
00070e15  1c232a31  383f464d  545b6269  70777e85  8c939aa1  a8afb6bd  c4cbd2d9
e0e7eef5  fc030a11  181f262d  343b4249  50575e65  6c737a81  888f969d  a4abb2b9
c0c7ced5  dce3eaf1  f8ff060d  141b2229  30373e45  4c535a61  686f767d  848b9299
a0a7aeb5  bcc3cad1  d8dfe6ed  f4fb0209  10171e25  2c333a41  484f565d  646b7279
```

将密钥 K 分割为 4 个 32b 数值，即 $K=(K_0,K_1,K_2,K_3)$。设每轮使用的子密钥为 $k_i(i=0,1,2,\cdots,31)$，Z_j 为中间变量，密钥变换算法如下：

$$Z_j = K_j \oplus FK_j, \quad j = 0,1,2,3$$

$$k_i = Z_{i+4} = Z_i \oplus T'(Z_{i+1} \oplus Z_{i+2} \oplus Z_{i+3} \oplus CK_i)$$

$$T'(x) = L'(S(x)), \quad L'(x) = x \oplus (x \ll 13) \oplus (x \ll 23)$$

可见，SM4 子密钥的生成同样采用了加密运算中的 S 函数变换，迭代运算方法也很类似，便于算法的实现。

SM4 算法的解密过程与加密完全相同，但逆序使用子密钥。

3.2.3 DES 算法

数据加密标准(Data Encryption Standard，DES)是信息加密领域最著名、应用最广泛的对称密钥加密技术，于 1977 年作为美国联邦标准发布，被全球各国广泛采用。DES 采用 DEA(Data Encryption Algorithm)算法，但习惯上称之为 DES 算法。

DES 符合 Feistel 模型，针对 64b 数据块(分组)进行加密和解密操作。DES 采用的密钥为 64b 的任意数，每字节的最高位作为奇偶校验位，因此实际密钥长度为 56b。

输入明文被划分成 64b 分组单元，DES 对每个分组进行 16 轮迭代运算，每轮使用一个 48b 的子密钥，而子密钥由 56b 的原始密钥变换而来。

DES 算法首先对 64b(设编号为 1～64)的输入分组作初始置换(Initial Permutation，IP)，置换规则如表 3.2 所示：第 6 位置换到第 24 位的位置，第 57 位置换到第 33 位的位置。即把位排列次序“打乱”，最后输出自然地分为 32b 的 L_0、R_0两部分。

表 3.2 DES 算法初始置换表

L_0(32b)								R_0(32b)							
58	50	12	34	26	18	10	2	60	52	44	36	28	20	12	4	57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8	61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

随后，DES 进入 16 轮的迭代运算 f，包括扩展换位运算、选择压缩运算、置换运算等步骤，每轮使用一个子密钥 K_i。运算过程如表 3.3 所示。

表 3.3 DES 算法迭代运算流程表

	L_{i-1}(32b)	R_{i-1}(32b)
16 轮 迭代运算 f $i=1,\cdots,16$	↓	扩展换位运算 $E\rightarrow$48b
		与 K_{i-1}按位异或运算(48b)
		选择压缩运算 $S\rightarrow$32b
		置换运算 $P\rightarrow$32b
		与 L_{i-1}按位异或运算(32b)
	L_i(32b)$\leftarrow R_{i-1}$	R_i(32b)

经过 16 轮次迭代运算 f 后，得到 L_{16} 和 R_{16}，以此作为输入，进行如表 3.4 所示的逆置换(IP^{-1})操作，即可得到输出的 64b 密文。IP^{-1} 正好是 IP 的逆变换，例如，第 1 位经过 IP 后，在第 40 位的位置；而通过 IP^{-1} 后，从第 40 位换回到第 1 位的位置。

表 3.4 DES 算法逆置换表

L_0(32b)								R_0(32b)							
40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25

扩展换位运算 E 可实现 32b 到 48b 变换，如表 3.5 所示，位排列次序发生变化，部分位被复制。

表 3.5 DES 算法扩展换位运算 *E*

R_i(32b)→48b															
32	1	2	3	4	5	4	5	6	7	8	9	8	9	10	11
12	13	12	13	14	15	16	17	16	17	18	19	20	21	20	21
22	23	24	25	24	25	26	27	28	29	28	29	30	31	32	1

置换运算 P 实现的是 32b 的换位操作，如表 3.6 所示，对位排列次序进行了调整。

表 3.6 DES 算法置换运算 *P*

32b															
16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

选择压缩运算 S 如图 3.12 所示。输入的 48b 数据被均分为 8 份，各为 6b，分别由 S 所包含的 S_1～S_8 8 个选择函数(S-box)进行处理。每个选择函数的功能是把 6b 数据变

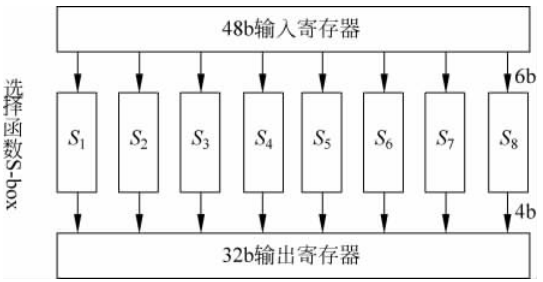

图 3.12 DES 算法选择压缩运算 *S*

换为 4b 数据，最终把 48b 输入值压缩转换为 32b。

选择函数 S_1～S_8采用查表变换方法。如表 3.7 所示，每个 S-box 均占有 4 行(0～3 行)、每行 16 列(0～15 列)的数值矩阵。设输入 6b 数据为 $D_1D_2D_3D_4D_5D_6$，令列号＝$D_2D_3D_4D_5$(取值 0,1,2,…,15)，行号＝D_1D_6(取值 0,1,2,3)，查表得到对应的数以 4b 表示，即得到 S-box 输出。例如，S_8 输入 110110，则行、列号为(2,11)，查表得 6，输出为 0110。

表 3.7 DES 算法选择压缩运算 S-box 表

S_1：	S_5：
14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,	2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,
0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,	14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,
4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,	4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,
15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13	11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3
S_2：	S_6：
15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,	12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,
3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,	10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8,
0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,	9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,
13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9	4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13
S_3：	S_7：
10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,	4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,
13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,	13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,
13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,	1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,
1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12	6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12
S_4：	S_8：
7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,	13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,
13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,	1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,
10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,	7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,
3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14	2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11

16 轮迭代运算 f 中，每轮使用的子密钥 K_0～K_{15}来自 64b 初始会话密钥 K(bit_0～bit_{63})。先实施密钥缩小选择换位运算 1(如表 3.8 所示)，换位的同时顺便去除了校验

位(每字节的最高位,如 bit_7),得到各 28b 的两部分 P 和 Q,作为每次计算子密钥的输入。

表 3.8 DES 算法密钥变换缩小换位运算 1

$K(64b)\rightarrow P(28b)+Q(28b)$
P: 56,48,40,32,24,16,8,0,57,49,41,33,25,17,9,1,58,50,42,34,26,18,10,2,59,51,43,35
Q: 62,54,46,38,30,22,14,6,61,53,45,37,29,21,13,5,60,52,44,36,28,20,12,4,27,19,11,3

共 16 次密钥变换($i=0,1,2,\cdots,15$)的计算方式一致:对 P 和 Q 分别进行 $M[i]$位循环左移,$M[i]=\{1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1\}$,得 P 和 Q,合并得到 56b 后,再实施缩小选择换位运算 2(如表 3.9 所示),即获得用于第 i 轮迭代运算的 48b 子密钥 K_i。

表 3.9 DES 算法密钥变换缩小换位运算 2

$56b(P+Q)\rightarrow 48b(K_i)$																							
13	16	10	23	0	4	2	27	14	5	20	9	22	18	11	3	25	7	15	6	26	19	12	1
40	51	30	36	46	54	29	39	50	44	32	47	43	48	38	55	33	52	45	41	49	35	28	31

DES 解密运算方法和加密完全一致,只是逆向使用子密钥。

DES 在诞生之初,有人曾估计要使用耗资 2000 万美元的专用计算机连续运行 12 小时才可能将其破解,当时它被认为很强大。然而,这一经典的算法如今面临安全性严重不足的问题,一方面是破解方法不断被探索,另一方面是 56b 密钥长度太短,很难抵抗性能越来越强大的计算机进行的暴力攻击。

改善 DES 保密性的一种可行方式为三重 DES(triple DES),即对一个明文分组用 3 个相同或不同的密钥实施 3 次 DES 加密(或解密)运算,可获得大约相当于 112b 长度密钥的强度。设:加密操作为 E,解密操作为 D,密钥为 K_1、K_2、K_3,则可采用以下 4 种模型之一实现加密:$E(K_1)E(K_2)E(K_3)$;$E(K_1)D(K_2)E(K_3)$;$E(K_1)E(K_2)E(K_1)$;$E(K_1)D(K_2)E(K_1)$。

DES 加密实际应用时,根据在加密过程中对明文分组不同的处理手段,分为 4 种工作方式(如图 3.13 所示),有的支持并行计算,有的可加强密文分组之间的相关性。

DES 算法的改进是高级加密标准(Advanced Encryption Standard,AES)。

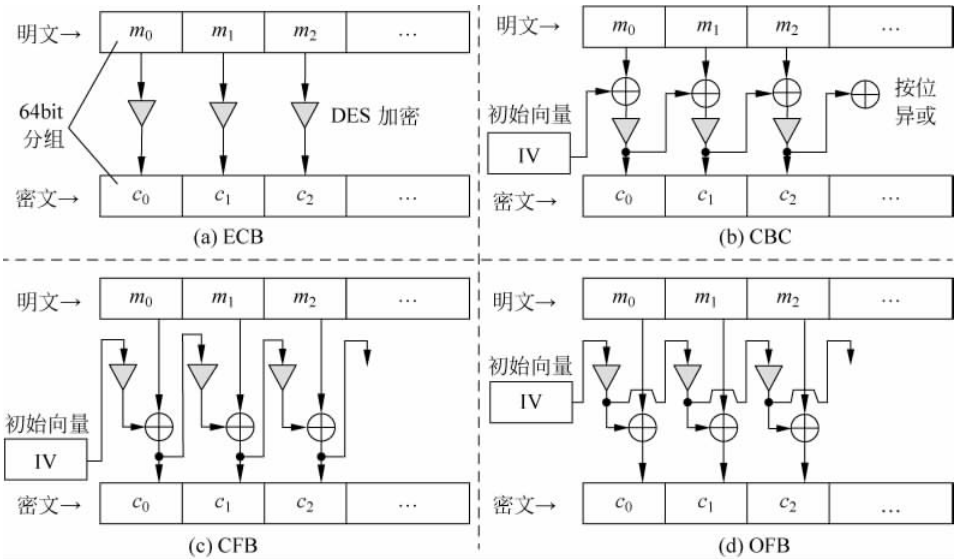

图 3.13 DES 算法工作方式原理图

3.3 非对称密钥加密

加密用哪个密钥、解密也必须用哪个，似乎就像锁定保险箱用这把钥匙、打开保险箱就一定用这把那样必然。然而这种“自然而然”的状况在 1976 年被 Diffie 和 Hellman 打破了，他们在《密码学新方向》一文中提出的技术颠覆了“一把钥匙开一把锁”的思维定式，开创了密码学的新领域。

3.3.1 非对称密钥加密技术原理

非对称密钥加密（asymmetric key cryptography）也称公开密钥加密（public key cryptography）或公钥加密、双密钥加密，其方法是使用一对密钥来加密和解密，其中一个是只有密钥拥有者自己掌握的、保密的私钥（private key），另一个是通信过程中由其他方使用的、可以公开的公钥（public key）。

公钥体制的优越性在于分离出两个相关的密钥，其中的公钥不需要保密，而私钥绝对不在网络上传输，因此就不存在密钥泄露问题。公钥和私钥的使用规则为：

- 用公钥加密的数据用且只能用对应的私钥解密。

• 用私钥加密的数据用且只能用对应的公钥解密。

假设 Alice 有一对密钥 priKeyA 和 pubKeyA，Bob 有 priKeyB 和 pubKeyB，他们需要在网络上传输信息。利用非对称密钥加密技术，Alice 和 Bob 有三种可选方法，如图 3.14(a)、(b)和(c)所示，获得的效果完全不同。

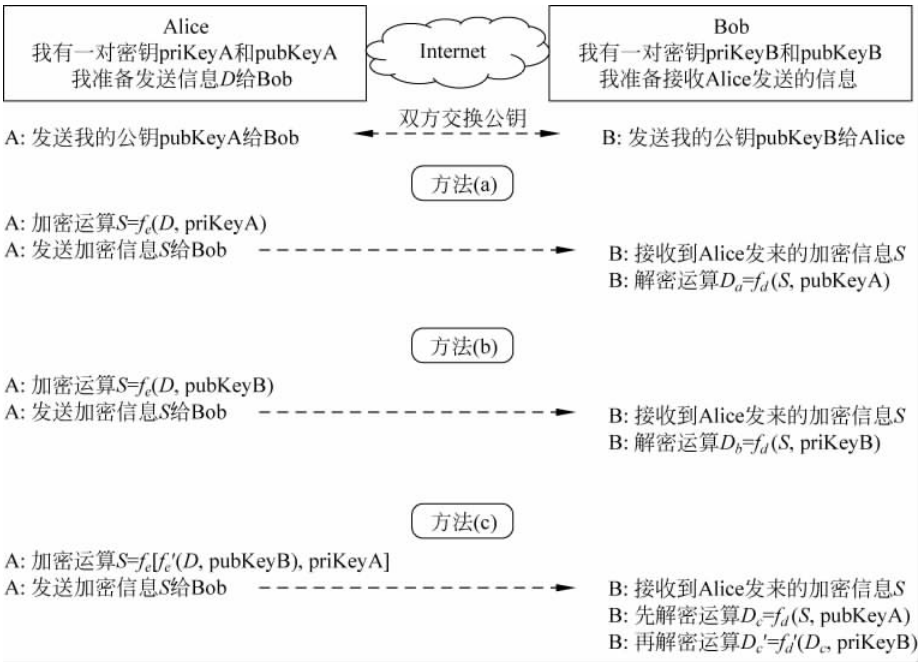

图 3.14　公钥加密不同方法比较示意图

(1) Alice 用自己的私钥 priKeyA 加密，可以确保信息是由其发出的，其他人没有其私钥就无法假冒，同时 Alice 也不能否认其发送的信息，该过程具有确权性；但用于解密的 Alice 的公钥 pubKeyA 是公开的，说明除了 Bob 以外的其他人也能获得公钥并能够解密，因此方法(a)不具有保密性。

(2) Alice 用 Bob 的公钥 pubKeyB 加密，使得只有握有私钥 priKeyB 的 Bob 才能解密，其他人则无法轻易解密，达到了保密的效果；但由于 Bob 的公钥是公开的，任何人都能进行加密发送，并不能指证该密文是 Alice 加密发出的，因此方法(b)不具有确权性。

(3) 方法(a)和(b)从安全效果上基本上是“互补”的关系，方法(c)就是对两种方法的综合，既能确认发送者，又能保护信息的私密性。

对于公钥加密的几种方法，擅于找破绽的 Bob 还有话要说："追根溯源，不管哪种方法，首先要把自己的公钥传播出去，这是一切操作的基础对不对？"

"是的。"Alice 不知道 Bob 的葫芦里卖的什么药。

"假如有个'中间人'攻击者拦截了你我发出的公钥，分别替换为他自己构造的密钥，会发生什么呢？"

"在方法(a)中我发送的信息你会拒绝认可，反而认可攻击者假冒我的身份发出的信息。"Alice 越说越心惊，"方法(b)中攻击者可以解密我发给你的保密信息，然后可以篡改后再'加密'发送给你。"

Alice 和 Bob 探讨的就是公钥的安全发布问题，是公钥体系安全运行的前提条件。公钥不仅可以公开，而且是越公开越好，假如让自己的公钥变成"众所周知"，那么"中间人攻击"就没有空子可钻了。实际的网络系统中应建立严密、可靠的公钥传播机制，才能让需要者获取到真实可信的公钥。

公钥加密的重要作用是信息验证。如图 3.15 所示，Bob 想公开自己的电子邮件地址，但又不希望被人恶意篡改，于是将名片信息用私钥加密后与名片一并发布，其他打算联络 Bob 却将信将疑的人就可以用 Bob 的公钥来验证，以确认这条信息真的是 Bob 发出的以及电子邮件地址真的属于 Bob。

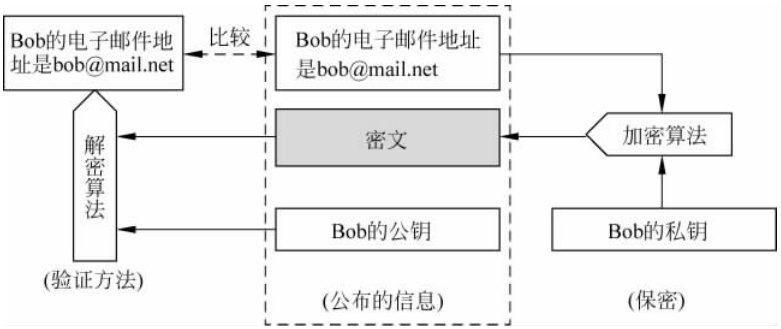

图 3.15 公钥加密应用示例

公钥加密的计算效率通常很低，甚至只有对称密钥加密方法的千分之一，因此不适合对大量的数据进行加密，一般用来加密会话密钥等短小数据。

常用的公钥加密算法有 RSA、ElGamal 等，目前技术含量最高、安全性最强的是 ECC 算法，该算法在比特币系统中得到充分运用，实现了虚拟货币资产的匿名持有和支付验证。

3.3.2 RSA 算法

RSA 算法于 1977 年被提出，以三位发明者 Ron Rivest、Adi Shamir 和 Leonard Adleman 姓氏的首字母来命名，是最早也是最著名的公开密钥加密算法，应用相当广泛。RSA 算法的数学基础是数论中的欧拉(Euler)定理，安全性建立在大整数分解质数(素数)因子的困难性之上。

RSA 算法公钥、私钥密钥对的生成步骤如下：

(1) 选择不同的大质数 p 和 q，计算 $n=p\times q$ 和 $\varphi(n)=(p-1)\times(q-1)$。

(2) 选择大整数 e，与 $\varphi(n)$互质，且 $1<e<\varphi(n)$。

(3) 计算整数 d，使 $d\times e=1 \bmod \varphi(n)$。

(4) 舍弃 p 和 q，得：公钥 pubKey=$\{e, n\}$，私钥 priKey=$\{d, n\}$。

考查私钥的安全性：虽然攻击者可以从公开的公钥得到 e 和 n，但以现有的数学理论成果，要分解一个整数为两个质数因子一般只能采用穷举法，而整数非常大时，计算机除法运算十分耗时，很难在有限时间内完成。1991 年 RSA 实验室曾发起因子分解挑战赛，公布了 54 个十进制位数为 100～617 位的大整数，不论谁无论用何种方法，分解出一个即得重奖。结果在 20 年时间里，只有最小的 18 个数(二进制 768 位长度以下)被成功分解，可见其困难程度。虽然近年来陆续有一些数学研究成果，可以在一定程度上摆脱暴力试除，但二进制 1024 位乃至 2048 位以上大整数仍然保持相当高的安全性。既然攻击者难以分解大数 n 以获取至关重要的 p 和 q，就无法获得 d，因此私钥是安全的。

RSA 密钥生成算法中，e 与 $\varphi(n)$互质的意思是两个数的最大公因数(Greatest Common Divisor，GCD)为 1。可以运用辗转相除法，又名欧几里得算法(Euclidean)，在判别 e 是否与 $\varphi(n)$互质时，不需要先把两个数作质因数分解，即可直接求出最大公因数。

求解 GCD 的算法用代码表示如下(不失一般性，设 $p>q$，递归函数返回值就是 p 和 q 的最大公因数)：

```
uint gcd(p, q)
{
    if ( (p mod q) <> 0 ) return gcd( q, (p mod q) );
    else return q;
}
```

此外，私钥中的 d 实际上是求 e 的 mod $\varphi(n)$乘法逆元，可使用扩展欧几里得算法。设求解 k^{-1} mod n，算法 extended_euclid(k, n)程序流程如下：

```
uint extended_euclid(k, n)
{
    (x1, x2, x3) = (1, 0, n); (y1, y2, y3)=(0, 1, k);
    while 1 {
        if (y3 == 0) return failure;          //不存在乘法逆元,失败返回
        if (y3 == 1) return(y2);              //成功返回,y2 即为 k 的乘法逆元
        else {
            q = x3 / y3;                      //q 为除法的商,为整数
            (t1, t2, t3) = (x1-q*y1, x2-q*y2, x3-q*y3);
            (x1, x2, x3) = (y1, y2, y3); (y1, y2, y3) = (t1, t2, t3);
        }
    }
}
```

使用公钥 pubKey=$\{e, n\}$的加密过程如下。对于明文 M，若 $M<n$，将 M 作为一个大整数来计算，若 $M\geqslant n$，则进行分段计算：

$$C = M^e \bmod n$$

使用私钥 priKey=$\{d, n\}$的解密过程如下：

$$M = C^d \bmod n$$

解密与加密为互逆的运算，可简单证明如下(根据欧拉定理)：

$$M' = (M^e)^d \bmod n = M^{e\times d} \bmod n = M^1 \bmod n = M$$

用私钥加密、公钥解密的计算公式同理可得。从计算方式来看，加密、解密都是指数运算，而且底和幂都是大整数，对计算机而言单次运算量很大，这正是公钥加密方法不太适用于大量信息的原因。

实际应用中应选择十进制 100 位以上的质数，n 的长度至少要达到 1024b。为了抵抗整数分解算法，对 p 和 q 另有如下要求：

(1) $|p-q|$很大，通常 p 和 q 的长度相近。

(2) $p-1$ 和 $q-1$ 分别含有大素因子 p_1 和 q_1。

(3) p_1-1 和 q_1-1 分别含有大素因子 p_2 和 q_2。

(4) $p+1$ 和 $q+1$ 分别含有大素因子 p_3 和 q_3。

3.3.3 ElGamal 算法

ElGamal 算法是一种公开密钥加密技术，其安全性原理依赖于计算有限域上离散对数这一难题。ElGamal 密钥对的产生流程如下：

首先选择一个质数 p 和两个随机数 g 和 x（g，$x<p$），计算：

$$y = g^x \bmod p$$

则公钥为$\{y,g,p\}$，私钥为 x。其中 g 和 p 可由一组用户共享。

ElGamal 进行公钥加密时，设明文为 M，需选择一个随机数 k，使 k 与$(p-1)$互质（可采用欧几里得辗转相除法），并计算：

$$a = g^k \bmod p$$

$$b = y^k M \bmod p$$

所得(a,b)即为密文，长度是明文的两倍。用私钥解密时计算：

$$M = \frac{b}{a^x} \bmod p$$

求解时可采用扩展欧几里得算法，先求 $a^x \pmod p$乘法逆元，再与 b 相乘，以降低计算难度。

在运用 ElGamal 时，质数 p 必须足够大，且 $p-1$ 应至少包含一个大质数因子，并保证 g 对于 $p-1$ 的大质数因子不可约。ElGamal 的一个不足之处是其密文长度为明文的两倍，增加了存储空间和传输带宽的占用。

3.3.4 ECC 算法

椭圆曲线加密算法（Ellipse Curve Cryptography，ECC）是基于椭圆曲线理论的公钥加密技术。在数学上，对椭圆曲线的性质和功能的研究已逾 150 年，但是在加密技术上的应用是在 1985 年由 Neal Koblitz 和 Victor Miller 首次提出。与其他建立在大质数因子分解困难性基础上的加密方法不同，ECC 利用椭圆曲线方程式的数学性质产生密钥，正向计算比较容易，反过来却非常困难。

与 RSA 方法相比，ECC 可以使用 164b 密钥产生一个安全级相当于 RSA 方法的 1024b 密钥提供的保密强度，而且计算量较小、处理速度更快、存储空间和传输带宽占用较少，具有较大的技术优势。

与一般加密方法采用的对数据进行函数运算的方式大相径庭，椭圆曲线加密是有限离散域的、曲线上坐标点的转换，其技术原理必须从抽象的射影平面开始去理解。

1. 射影平面

平面上的两条直线只有相交、平行两种情况。相交线只有一个交点，若有两个交点则为重合；平行线没有交点。

定义平行线相交于无穷远点 P_∞，如图 3.16 所示，这样，平面上任何两条直线都统一为有唯一的交点。P_∞ 具有以下性质：

(1) 一条直线只有一个无穷远点，一对平行线有公共的无穷远点(即交点)。

(2) 任何两条不平行的直线有不同的无穷远点(否则会造成两个交点)。

(3) 平面上全体无穷远点构成一条无穷远直线。

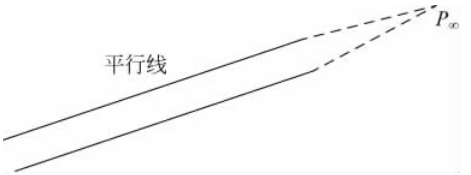

图 3.16　射影平面的平行线示意图

平面上全体无穷远点(无穷远直线)与全体平常点构成射影平面(projective plane)。

对普通平面上点(x, y)，令 $x=X/Z$，$y=Y/Z$，$Z\neq 0$，则投影为射影平面上的点$(X:Y:Z)$。例如点(1,3)可投影为$(Z:3Z:Z)$，即可为$(1:3:1)$、$(2.3:6.9:2.3)$等多种赋值。

对普通平面上的直线 $ax+by+c=0$，作同理变换，得到对应于射影平面上的直线为 $aX+bY+cZ=0$。

对平行线 $aX+bY+c_1Z=0$ 和 $aX+bY+c_2Z=0$，易解得 $Z=0$，说明射影平面上平行线交点即无穷远点 P_∞ 的坐标为$(X:Y:0)$。

2. 椭圆曲线

一条椭圆曲线(ellipse curve)是在射影平面上满足威尔斯特拉斯方程(Weierstrass)的所有点的集合。射影平面上统一的椭圆曲线方程为

$$Y^2Z+a_1XYZ+a_3YZ^2=X^3+a_2X^2Z+a_4XZ^2+a_6Z^3$$

椭圆曲线方程是一个齐次方程，且要求椭圆曲线上的每个点都必须是非奇异的（光滑的）、可导的，即方程的偏导数 $F_X(X,Y,Z)$、$F_Y(X,Y,Z)$和 $F_Z(X,Y,Z)$不能同时为 0。

令 $Z=0$，代入椭圆曲线方程得 $X=0$，说明椭圆曲线上有一个无穷远点 O_∞，其坐标为$(0:Y:0)$。无穷远点 O_∞和普通平面上的平常点（即曲线）一起，共同组成射影平面上的椭圆曲线。

运用射影平面与普通平面的点的转换关系，椭圆曲线方程可转换为普通平面方程：

$$y^2+a_1xy+a_3y=x^3+a_2x^2+a_4x+a_6$$

对椭圆曲线的平常点(x, y)求导，并计算过该点的切线的斜率 k，有：

$$F_x(x,y)=a_1y-3x^2-2a_2x-a_4$$

$$F_y(x,y)=2y+a_1x+a_3$$

$$k=f'(x)=-\frac{F_x(x,y)}{F_y(x,y)}=\frac{3x^2+2a_2x+a_4-a_1y}{2y+a_1x+a_3}$$

椭圆曲线的形状并非如其名呈椭圆状，例如，方程 $Y^2Z=X^3+XZ^2+Z^3$ 可转换为普通方程 $y^2=x^3+x+1$，曲线如图 3.17(a)所示；方程 $Y^2Z=X^3-XZ^2$ 可转换为普通方程 $y^2=x^3-x$，曲线如图 3.17(b)所示。

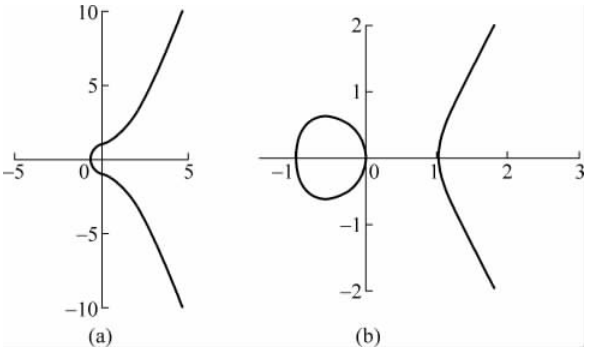

图 3.17　椭圆曲线示例

而且并非所有形式类似的方程都是椭圆曲线方程，例如，如图 3.18(a)和(b)所示的方程 $Y^2Z=X^3+X^2$ 和 $Y^2Z=X^3$ 就不属于椭圆曲线，因为 0 点为奇异点（不可导）。

3. 椭圆曲线加法

在椭圆曲线中引入阿贝尔（Abel）加法群（又称交换群）的概念，可进一步实现对椭圆

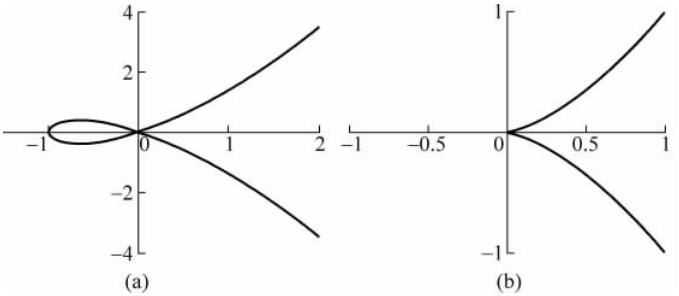

图 3.18　非椭圆曲线示例

曲线上的点的运算。

任意取椭圆曲线上两点 P、Q(若 P、Q 两点重合,则作 P 点的切线),过两点作直线交于椭圆曲线的另一点 R',过 R'做 y 轴的平行线交椭圆曲线于 R,定义 $P+Q=R$。可见,加法的和也在椭圆曲线上,并同样遵循加法的交换律、结合律。

例如,图 3.17(b)所示的椭圆曲线方程为 $Y^2Z=X^3-XZ^2$,普通方程为 $y^2=x^3-x$,加法运算过程如图 3.19 所示,其中图 3.19(a)和图 3.19(b)分别为 P、Q 重合与不重合两种情况。

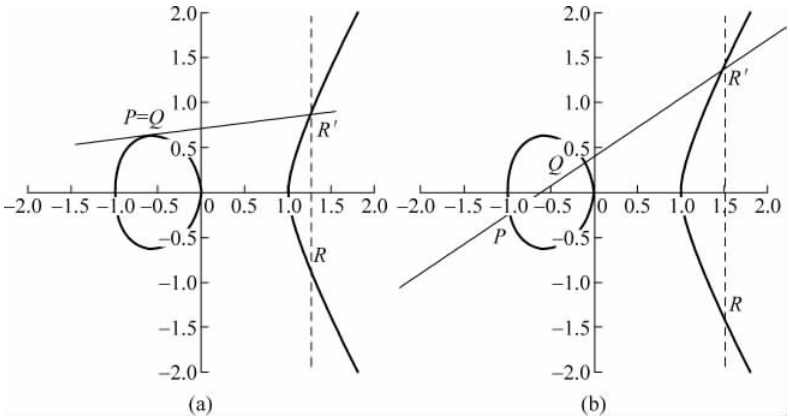

图 3.19　椭圆曲线加法原理图

如图 3.20(a)所示,椭圆曲线无穷远点 O_∞ 与椭圆曲线上一点 P 的连线交椭圆曲线于另一点 P',过 P'作 y 轴的平行线必交椭圆曲线于 P(两条线重合),根据加法定义,有 $O_\infty+P=P$。可见,无穷远点 O_∞ 与普通加法中零相当,因此把 O_∞ 称为零元。同时易知

$P+P'=O_\infty$，于是 P'被称为 P 的负元，记作 $-P$。如图 3.20(b)所示，还可推出：如果椭圆曲线上的 3 个点 A、B、C 处于同一直线上，则其和等于零元，即 $A+B+C=O_\infty$。

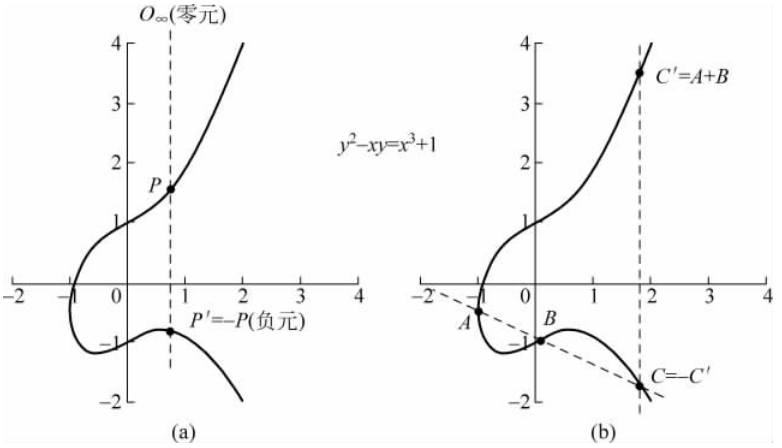

图 3.20　椭圆曲线零元与负元

再进一步，如图 3.21 所示，若有 k 个相同的点 P 相加，记作 kP，有：

$$P+P+P=2P+P=3P$$

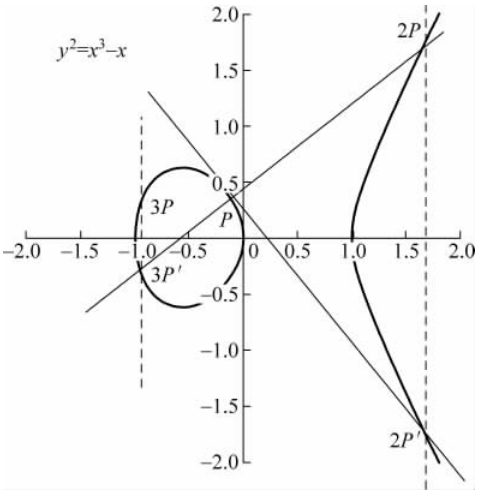

图 3.21　椭圆曲线同点加法示意图

在图 3.19 所示曲线上，若已知点 P、Q 的坐标分别为(x_1,y_1)、(x_2,y_2)，令 $R=P+Q$，设：$-R$(即 R')的坐标为(x_3,y_3)，R 的坐标为(x_4,y_4)，显然有 $x_3=x_4$。

因为 P、Q、$-R$ 三点共线，所以设共线方程为 $y=kx+b$，分两种情况：

(1) 若 $P\neq Q$(P、Q 两点不重合)，则直线斜率为：

$$k=\frac{y_1-y_2}{x_1-x_2}$$

(2) 若 $P=Q$(P、Q 两点重合)，则直线为椭圆曲线的切线，代入斜率公式可得：

$$k=\frac{3x_1^2+2a_2x_1+a_4-a_1y_1}{2y_1+a_1x_1+a_3}$$

因此，P、Q、$-R$ 三点的坐标值(x_1,y_1)、(x_2,y_2)、(x_3,y_3)就是椭圆曲线统一方程与共线方程组成的方程组的解(即三个交点)。将共线方程代入后得：

$$(kx+b)^2+a_1x(kx+b)+a_3(kx+b)=x^3+a_2x^2+a_4x+a_6$$

将其整理(按次数归并)并化为一般方程为：

$$x^3+(a_2-ka_1-k^2)x^2+(a_4-a_1b-2kb-ka_3)x+a_6-a_3b=0$$

根据三次方程根与系数的关系可知：当三次项系数为 1 时，$-x_1x_2x_3$ 等于常数项，$x_1x_2+x_2x_3+x_3x_1$ 等于一次项系数，$-(x_1+x_2+x_3)$等于二次项系数，列方程组可解出 x_1、x_2、x_3，再根据共线方程可求出 y_1、y_2、y_3。

由于 $-(x_1+x_2+x_3)=a_2-ka_1-k^2$，即 $x_4=x_3=k^2+ka_1-a_2-x_1-x_2$，又由于 $k=\frac{y_1-y_3}{x_1-x_3}$，即可求出：$y_3=y_1-k(x_1-x_3)$。

由于 R 就是 R'作 y 轴平行线与曲线的交点，因此将 $x=x_4$ 代入椭圆曲线统一方程，并化为一般方程，得：

$$y^2+(a_1x_4+a_3)y-(x_4^3+a_2x_4^2+a_4x_4+a_6)=0$$

根据二次方程根与系数的关系$\left(x_1+x_2=-\frac{b}{a};\ x_1x_2=\frac{c}{a}\right)$，有 $-(a_1x_4+a_3)=y_3+y_4$，则可求出 $y_4=-y_3-(a_1x_4+a_3)$。所得 $R(x_4,y_4)$即为 P、Q 的和。

4. 有限域椭圆曲线

由于信息的明文、密文都是整数型数值，因此信息加密(即整数间的变换)应当是在有限域上进行的，域的最大值、最小值由信息长度决定(并非无穷大)，而且信息是离散型的整数，所以必须对实数域上的椭圆曲线进行改进，以适合有限数量的整数运算的需要。另外，椭圆曲线的选择很重要，并不是所有椭圆曲线都适合加密。

定义有限域 F_p：

(1) F_p 中有 p(p 为质数)个元素 0,1,2,…,$p-2$,$p-1$。

(2) F_p 的加法是 $a+b\equiv c(\mathrm{mod}\ p)$。

(3) F_p 的乘法是 $a\times b\equiv c(\mathrm{mod}\ p)$。

(4) F_p 的除法是 $a\div b\equiv c(\mathrm{mod}\ p)$。

(5) F_p 的单位元是 1,零元是 0。

(6) F_p 域内运算满足交换律、结合律、分配律。

以椭圆曲线 $y^2=x^3+ax+b$ 为例,将其定义在 F_p 上,即对应 $y^2=x^3+ax+b(\mathrm{mod}\ p)$上的所有点$(x,\ y)$再加上无穷远点 O_∞。无穷远点 O_∞ 是零元,$O_\infty+O_\infty=O_\infty$,$O_\infty+P=P$。$P(x,\ y)$的负元是 $-P=P'(x,\ -y)$,有 $P+(-P)=O_\infty$。

选择质数 p,应有 $x,y\in[0,p-1]$。将这条椭圆曲线记为 $E_p(a,\ b)$。选择两个小于 p 的非负整数 a、b,满足约束条件：$4a^3+27b^2\neq 0(\mathrm{mod}\ p)$。

以一条简单的椭圆曲线为例。设 $p=23$,$a=b=1$,椭圆曲线可记为 $E_{23}(1,1)$,曲线如图 3.22 所示。可见离散域上的椭圆曲线已经变成一些不连续的点,其坐标均为整数。

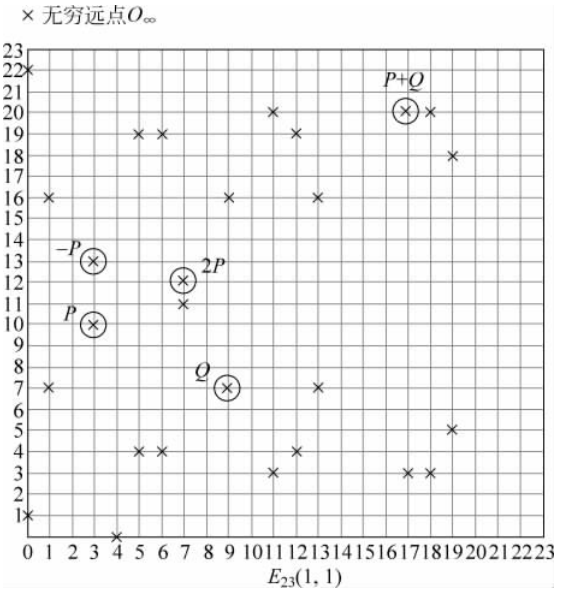

图 3.22　有限域椭圆曲线示例

如果已知曲线上两点 $P(3,10)$、$Q(9,7)$，则 P 的负元 $-P=(3,-10)$，即 $(3,13)$，斜率 $k=(7-10)/(9-3)=-\frac{1}{2}$，因为 $2\times12=1(\text{mod }23)$，所以 2 的乘法逆元为 12，即 1/2，同时 $-12(\text{mod }23)=11(\text{mod }23)$，所以有 $k=11$。

根据 P 和 Q 的和 $R(x,y)$ 的计算公式，有：

$$\begin{cases} x = k^2 - x_1 - x_2 = 11^2 - 3 - 9 = 109 = 17(\text{mod }23) \\ y = k(x_1 - x) - y_1 = 11\times(3-(-6)) - 10 = 89 = 20(\text{mod }23) \end{cases}$$

因此 $P+Q$ 的坐标为 $(17,20)$。

另外，过 $P(3,10)$ 的切线斜率 k' 根据公式可计算为：

$$k' = \frac{3\times3^2+1}{2\times10} = \frac{1}{4} = 6(\text{mod }23)$$

则可同理计算 $R(x,y)$ 的坐标为：

$$\begin{cases} x = 6^2 - 3 - 3 = 30 = 7(\text{mod }23) \\ y = 6\times(3-7) - 10 = -34 = 12(\text{mod }23) \end{cases}$$

因此 $2P$ 的坐标为 $(7,12)$，依此类推可得 $3P$、$4P$ 及任意 nP。

如果椭圆曲线上一点 P，存在最小的正整数 n，使得数乘 $nP=O_\infty$（显然 $(n-1)P=-P$），则将 n 称为 P 的阶，若 n 不存在，则 P 是无限阶的。事实上，在有限域上定义的椭圆曲线上所有的点的阶 n 都是存在的。

5. 椭圆曲线加密

在椭圆曲线 $E_p(a,b)$ 上选择一个点 G 为基点（Base Point），n 为阶（$nG=O_\infty$），并选择一个整数 $k(k<n)$。计算 $K=kG$，K 也是椭圆曲线上的点。则点 k 为私钥，K 为公钥（椭圆曲线 $E_p(a,b)$ 和 G 也是公钥的组成部分）。

不难发现，给定 k 和 G，根据加法法则，计算 K 很容易；但反过来，即使已知 K 和 G，求 k 却是非常困难的（k 是大数），唯一途径是暴力破解，但耗时漫长。这就是椭圆曲线加密算法安全性的数学依据。

椭圆曲线加密的基本原理是对曲线上的点实施变换，所以首先需要将待加密的明文数据进行编码，转化为椭圆曲线上的一个点（坐标形式），才能符合加密运算的条件。解密后的数据同样是曲线上的点，则需要反过来进行译码，恢复为明文数据的表达方式。

根据数论定义，令 n 为正整数，若整数 a 满足与 n 互质（即有 $\text{GCD}(a,n)=1$），且使得

$x^2=a(\mathrm{mod}\ n)$有解，则称 a 为模 n 的平方剩余，记为 QR_n，否则 a 称为模 n 的平方非剩余，记为 QNR_n。

设：明文 $m\in(0,\ M)$，M 为与 m 相同二进制长度的最大整数。又设函数 $f(x)$：$y^2=x^3+ax+b$(标准椭圆曲线方程)，有限域 $E_p(a,\ b)$元素个数为 p(p 为质数)。

选择整数 k，满足 $Mk<p$，令 $x_i=mk+i$ $(i=1,2,\cdots,k-1)$，依次计算 $f(x_i)=x^3+ax+b(\mathrm{mod}\ p)$。若找到 $f(x_i)$是模 p 的平方剩余(QR_p)，则用 x_m 表示此时的 x_i，y_m 表示 $f(x_i)$的平方根，这样明文 m 即编码成为椭圆曲线上的点(x_m,y_m)。

译码计算非常简单，由于 $i\in(0,\ k)$，只需取解密后的 x'_m坐标值进行除法运算并取整$[x'_m/k]$，即恢复明文 m。

因为模 p 的平方剩余和平方非剩余各占一半，所以 k 次内找到 y_i^2 的概率不小于 $1-(1/2)^k$，编码成功率很高。另外也可设计其他不同的编码、译码方法。

需要进行公钥加密、私钥解密来传输保密数据时，密钥生成方法和加密、解密的操作过程如下：

(1) 接收方选定一条椭圆曲线 $E_p(a,\ b)$，并取椭圆曲线上一点作为基点 G；选择一个随机数为私钥 k，并生成公钥 $K=kG$。接收方将椭圆曲线 $E_p(a,\ b)$和点 K、G(即公钥)传给发送方。

(2) 发送方收到公钥后，先将待传输的明文编码到 $E_p(a,\ b)$上的一点 M，并产生一个随机数 $r(r<n)$。计算 $C_1=M+rK$ 和 $C_2=rG$。发送密文 C_1 和 C_2。

(3) 接收方收到密文后，进行解密计算 $M'=C_1-kC_2$，M'经过译码即为明文。解密运算的原理证明如下：

$$C_1-kC_2=M+rK-k(rG)=M+r(K-kG)=M$$

如需要用私钥加密原消息作签名，然后用公钥来验证签名，则密钥对由发送方生成，密钥生成方法不变，公钥被传给接收方，之后进行的加密、验证过程如下：

(1) 发送方生成随机数 r，计算 $S_1=rG$。对原消息 M 作单向函数运算 $h=\mathrm{Hash}(M)$，计算 $S_2=(h+kM)/r$。

(2) 接收方收到原消息 M 和密文 S_1、S_2 后，计算验证 $hG/S_2+MK/S_2=S_1$ 是否成立。原理证明如下：

$$\frac{hG+MK}{S_2}=\frac{(hG+MK)\times r}{h+kM}=\frac{(hG+MkG)\times r}{h+kM}=\frac{(h+kM)\times rG}{h+kM}=rG=S_1$$

椭圆曲线方程的选择对于加密算法而言至关重要，是加密强度的基础。选择不当可

能造成加密强度不足，甚至可能存在安全漏洞（假如是故意为之，则为安全后门）。

通常将 F_p 上的一条椭圆曲线描述为 $T=(p,a,b,G,n,h)$。其中：p、a、b 用来确定一条椭圆曲线，G 为基点，n 为点 G 的阶，h 是椭圆曲线上所有点的个数 m 与 n 相除的商的整数部分。这 6 个参量取值的选择，直接影响了加密的安全性。参量值一般要求满足以下几个条件：

- p 越大安全性越好，但会导致计算速度变慢，200b 左右可满足一般安全要求；
- n 应为质数；
- $h \leqslant 4$；
- $p \neq n \times h$；
- $pt \neq 1 \pmod{n}$，$1 \leqslant t < 20$；
- $4a^3 + 27b^2 \neq 0 \pmod{p}$。

比特币系统中即采用椭圆曲线加密法为签名算法，并选用了 Secp256k1 曲线，参照 SEC 标准（Standards for Efficient Cryptography）。F_p 上的椭圆曲线参量 $T=(p,a,b,G,n,h)$ 定义如下（方程式为 $y^2=x^3+7$，曲线如图 3.23 所示）：

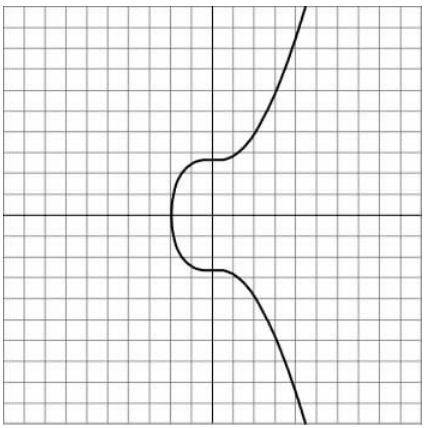

图 3.23 比特币系统采用的椭圆曲线示意图

- 有限域采用 256b 的质数 p：

$p=$ FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F

$= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$

- 椭圆曲线 E 为 $y^2=x^3+ax+b$，其中：$a=0$，$b=7$。
- 基点 G 选择分为两种情况：

(1) 压缩格式下 G = 02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798

(2) 非压缩格式下 G = 04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8

- 基点 G 的阶 n 为：

n = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141

- $h=1$。

3.4　单向函数加密

密码学中有一种特殊类型的加密技术，但称其为加密实在有点勉为其难，第一因为其不支持解密，第二通常没有密钥。

Alice 带来一个筒形背包，里面装的圆柱形的木块的直径与背包一致，恰好装得满满当当。Alice 边把木块取出来边对 Bob 说："这些木块圆柱直径相等，高度不等，问怎么求背包高度？"

"算这个初等数学题我还是游刃有余的。"Bob 不假思索地刷刷写下一个算式：

$$h = \sum_{i=1}^{k} h_i \times n_i$$

"这里 h_i 是各种木块的高度，n_i 是背包中这种木块的块数，h 就是你背包的高度。"

"好。换一个问题。"Alice 当然不会考 Bob 如此简单的问题，后面肯定会有个大坑在等着，"假如我告诉你各种木块的高度，还告诉你背包的高度，问各需要多少块不同木块正好装满背包？"

Bob 认真想了想，无奈地回答："这是多元一次方程，除非有更多的条件构成方程组，否则只能凑数求解。"

Alice 所提的正是有名的背包问题(knapsack problem)。设高度值 h_i 为已知系数，各种高度对应的块数的集合$\{n_i\}$为数据，则从数据计算总高度 h 是非常容易的，而从总高度反推数据则非常困难。

相似的还有离散对数问题：令质数 p 满足 $p-1$ 含有另一大质因子 q 及一整数 $g(1<g<p-1)$。给定整数 x，求 $y=g^x \bmod p$，只需要有限次的乘法运算；但如果给定 y、g 和 p，要求解 x，则除了暴力尝试别无他法，在计算上不可行。

3.4.1 单向函数技术原理

单向函数(one-way function)运用了背包问题的原理，如图 3.24 所示，将原消息作为函数的输入(相当于$\{n_i\}$)，函数的输出(相当于 h)为消息摘要(message digest)。单向函数是将原消息进行处理后生成固定长度(一般比原消息短很多)的数据，无法还原，这是其名为单向的原因所在。单向函数又称为哈希函数(hash function)，消息摘要就称为原消息的哈希(hash)。

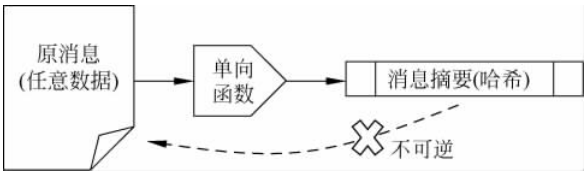

图 3.24 单向函数原理图

哈希值能够在统计上唯一地表征输入值，即把消息进行数学意义上的内容摘要，但从摘要中无法得到比摘要本身更多的关于原有消息的信息，具有安全性。哈希算法需要满足以下关键特性：

(1) 单向性。从输入值能够简单、迅速地得到哈希值，而反过来在计算上不可行。

(2) 抗冲突性(collision resistant)。给定 M，计算上难以找到 M'，满足 $H(M)=H(M')$，此谓弱抗冲突性；如果计算上也难以寻找一对任意的 M 和 M'，满足 $H(M)=H(M')$，此谓强抗冲突性。抗冲突性在某种程度上可以认为就是哈希的唯一性。

(3) 分布均匀性。存在映射分布均匀性和差分分布均匀性。哈希值中，0 和 1 的数量应该大致相等；输入中每 1b 的变化，哈希值中应有一半以上的位改变，这被称为雪崩效应(avalanche effect)；反之，要实现使哈希值中出现 1b 的变化，则输入中至少有一半

以上的位必须发生变化。分布均匀性本质上是必须使输入中每一位的信息，尽量均匀地反映到输出的每一位上去，同时输出中的每一位，都是输入中尽可能多的位一起作用的结果。

单纯从理论上看，原消息的编码空间巨大，例如照片、电影，动辄几兆、几百兆字节，而哈希仅有数十个字节长度，编码空间相对很小，哈希冲突的概率应当很高才对。然而，考查实际的数据对象，以英文文章为例，每个字节的256个不同编码中可读ASCII字符只占不到一半，文章大部分字符都是小写字母，有效编码空间大幅缩小，再考虑到文章不是字母的胡乱组合，而是字母构成数量有限的单词、单词组成有意义的语句，这就进一步压缩了编码空间。因此，修改一篇文章，至少要通顺，表达的意思上达到篡改的目的，还要使之输出相同的哈希，无疑是天方夜谭。

单向函数因为具备这些独特的性质，可在信息安全领域发挥巨大作用。哈希携带了原消息的内容特征，不同的消息有不同的哈希，因此也称其为数字指纹(digital fingerprint)，就好比人类的指纹可以用来“画押”，与亲笔签名有同等效力，可以用来代表一个人。犯罪现场的指纹鉴定也是同理。

例如，有一份带上哈希的电子合同，如图3.25所示，倘若合同被篡改，哪怕只是改了一个小数点，用同样的单向函数得到的哈希将发生巨大的变化，而且几乎不可能伪造一份具有相同哈希的合同，所以数字指纹可以用来验证合同真伪。然而，如果合同篡改者同时用虚假合同生成了新的哈希，替换了原有的数字指纹，则反而会带来“虚假的真实”的弊端。因此，数字指纹在实际运用时还需要结合其他密码学技术，例如配合公钥加密，才能切实保障其安全性。

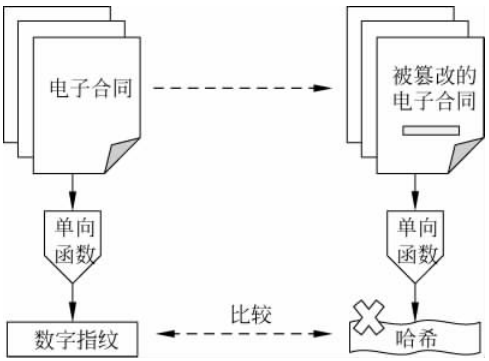

图3.25　数字指纹应用示例

再例如，网络用户在客户端输入用户名、密码（口令）登录几乎是每天要做的事。如果直接把用户输入的密码传送到服务端进行验证，就存在两方面的安全隐患：一是传输过程中可能被窃听，明文密码就暴露了；二是服务端数据库存储的明文密码也有暴露风险（例如被“脱库”或被“内鬼”窃取账号）。为防范登录密码安全风险，网络系统通常采用对密码作哈希的方法，客户端向服务器传输的是密码的哈希值，而服务端存储的也是哈希值，不影响登录验证，也不会暴露用户密码，这是单向函数加密技术的成功应用范例。

但是网络登录密码单纯采用哈希加密手段仍然存在安全风险，例如攻击者截取到哈希值后，至少有两种手法可实现攻击：一种是“重放攻击”，即直接向服务端传送密码哈希值要求登录，服务端会照样放行；另一种是“字典攻击”，即利用事先做好的密码串-哈希值对应表（因为大部分人会用便于记忆的单词和词组来做密码），查表“恢复”密码明文。抵抗这一风险的做法是采用“哈希加盐”的一次性登录（Single-time Sign-On，SSO）方案，如图 3.26 所示，每次登录使用随机数作为“盐”（salt），与密码哈希拼接后再做哈希，从而实现每次登录都传输不同的密文（随机数可事先传输或随密文传输，不影响安全性）。

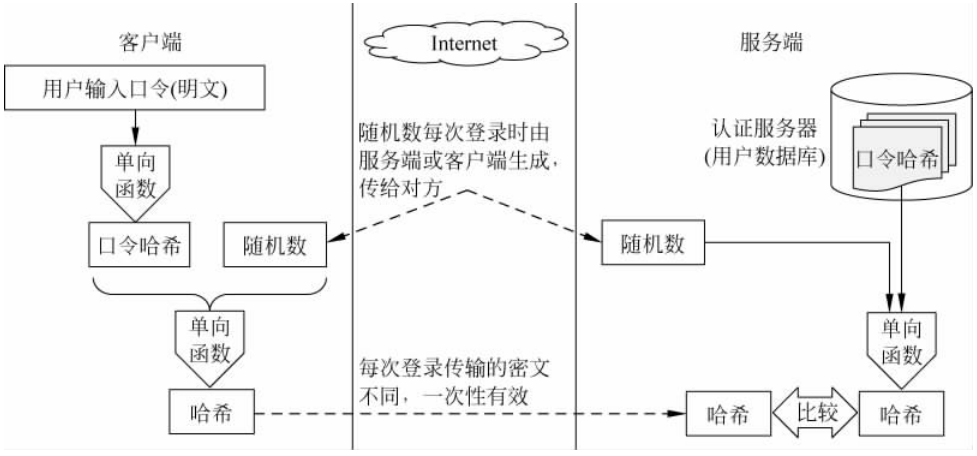

图 3.26　一次性口令技术原理

单向函数的另一项重要用途是信息标识。如果需要识别大数据量的信息对象，例如不同的电影，“全量比较”费力不讨好，则可生成信息对象的数字指纹，例如 32B 的定长哈希，作为类似身份证号码的 ID，检索时只需出示 ID，存储、传输、比对、查找都很便捷。例如网上 P2P 下载应用就是采用 DHT（分布式哈希表）方法，从各个用户终端上寻找大文件的不同片段，实现并发式下载。

比特币系统设计中大量运用了成熟的单向函数加密技术，将单向函数各种技术特点和功能发挥得淋漓尽致：

- 数据缠结——利用哈希对原消息的变化极其敏感的特性，构造数据之间的缠结关系，防范信息篡改和伪造，包括：形成区块之间的链和生成交易默克尔树。
- 信息标识——利用哈希短小（压缩性）、等长、抗冲突等特性，以之为区块和交易的标识，非常便于存储和检索。
- 地址编码——利用哈希单向转换的特性，生成比特币用户地址，既提高了安全性、规范性、隐蔽性，又实现了可验证。
- 交易签名——利用哈希作为原消息数字指纹的特性，对资金归属和使用规则进行签名锁定，并可验证、解锁，实现智能化交易。

常用的单向函数算法有 CRC、MD、RIPEMD、SHA 等。

3.4.2　CRC 算法

循环冗余码（Cyclic Redundancy Check，CRC）通常不被当作单向函数来看待，常用在网络通信中的数据链路层，例如作为传输检错的帧校验序列（Frame Check Sequence，FCS）编码。但 CRC 码确实就是一种单向函数，而且适合流式数据，可以边传输边计算哈希，具备独特的能力。

CRC 码是一种线性分组码，一般为 16b 或 32b 数值，编码和解码方法简单，能够达到 0.0047%以下漏检率，可检测出所有奇数个随机错误以及长度小于或等于生成多项式阶数的突发错误。

设：生成多项式 $P(x)$ 为 n 阶，传输的比特流为 M，则生成 nb 的 CRC 码 $R(x)$。生成多项式是系数为 1 或 0 的一元多次多项式，其最高指数就是阶。例如，$P(x)=x^7+x^5+x+1$，阶为 7，对应 8b 二进制数为 10100011。

CRC 码计算公式为：$R(x)=(M\times 2^n) \bmod P(x)$。即原消息比特流 M 扩展到 n 位后作为被除数，并以生成多项式 $P(x)$ 转换而来的二进制数据为除数，做按位除法，取余数 $R(x)$ 扩展到 n 位即为 CRC 码。

CRC 码紧接在 M 比特流的后面进行传输，信息接收方采用同样的生成多项式进行按位除法运算以验证：$R'(x)=(M\times 2^n+R(x)) \bmod P(x)$，若 $R'(x)=0$，说明 M 无差错，否则说明传输有误（存在误码或被篡改）。

如图 3.27 所示为计算余数的按位除法的竖式运算法示例，其中每一步采用的不是减法，而是按位异或运算。

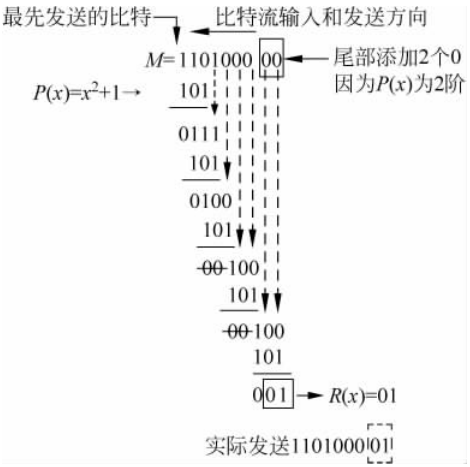

图 3.27 CRC 按位除法示例

CRC 码生成多项式的选择非常重要，是保障检错能力的基础。国际标准中较常用的生成多项式有：

- CRC-16=$x^{16}+x^{15}+x^2+1$；
- CRC-CCITT=$x^{16}+x^{12}+x^5+1$；
- CRC-32=$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$。

3.4.3 MD 算法

消息摘要(Message Digest，MD)是一种经典的单向函数加密算法，由 R. Rivest 于 1989—1992 年间开发并升级，分为 MD2、MD4 和 MD5，其中 MD5 算法使用最为广泛。

MD5 以 512b(64B)分组为单位来处理输入信息，输出 128b(16B)哈希值。

首先对信息进行填充，在信息的尾部填充一个 1 和连续的 0，直到满足位长度对 512 求余的结果等于 448(即 $n\times512+448$)，其后附加 64b 原信息长度值(以 byte 为单位)。

MD5 对 4 个 32b 的链接变量(chaining variable)分别设置初始值为：V_a=0x01234567，V_b=0x89abcdef，V_c=0xfedcba98，V_d=0x76543210。执行算法流程如下：

(1) 令 $a=V_a$，$b=V_b$，$c=V_c$，$d=V_d$。

(2) 将输入的 512b 分组划分成 16 个 32b 子分组 m_j，$j=0,1,2,\cdots,15$。

(3) 进行 4 轮运算，每轮由 16 步组成，每步为 $K(A,B,C,D,m_j,s,t_i)$函数运算(如图 3.28 所示)，K 函数表示为：$A=B+((A+k(B,C,D)+m_j+t_i)\gg s)$，$K\in\{F, G, P, Q\}$，$k\in\{f, g, p, q\}$，$\gg s$ 表示循环右移 s 位，s 是常数(图 3.28 中对应数值)。在第 i 步中($i=1,2,\cdots,64$)，以 i 为弧度值，可计算常数为：$t_i=\text{int}(2^{32}\times|\sin(i)|)$。分别用于 4 轮运算的 4 个非线性函数如下(每轮一个，均为按位逻辑运算：& 与，| 或，～非，⊕异或)：

$$f(x, y, z) = (x \,\&\, y) \mid ((\sim x) \,\&\, z)$$

$$g(x, y, z) = (x \,\&\, z) \mid (y \,\&\, (\sim z))$$

$$p(x, y, z) = x \oplus y \oplus z$$

$$q(x, y, z) = y \oplus (x \mid (\sim z))$$

(4) 计算：$V_a=V_a+a$，$V_b=V_b+b$，$V_c=V_c+c$，$V_d=V_d+d$。

(5) 若存在更多 512b 分组，回到(1)继续运算，否则算法结束。

第一轮	第二轮
F(a,b,c,d,m0,7,0xd76aa478)	G(a,b,c,d,m1,5,0xf61e2562)
F(d,a,b,c,m1,12,0xe8c7b756)	G(d,a,b,c,m6,9,0xc040b340)
F(c,d,a,b,m2,17,0x242070db)	G(c,d,a,b,m11,14,0x265e5a51)
F(b,c,d,a,m3,22,0xc1bdceee)	G(b,c,d,a,m0,20,0xe9b6c7aa)
F(a,b,c,d,m4,7,0xf57c0faf)	G(a,b,c,d,m5,5,0xd62f105d)
F(d,a,b,c,m5,12,0x4787c62a)	G(d,a,b,c,m10,9,0x02441453)
F(c,d,a,b,m6,17,0xa8304613)	G(c,d,a,b,m15,14,0xd8a1e681)
F(b,c,d,a,m7,22,0xfd469501)	G(b,c,d,a,m4,20,0xe7d3fbc8)
F(a,b,c,d,m8,7,0x698098d8)	G(a,b,c,d,m9,5,0x21e1cde6)
F(d,a,b,c,m9,12,0x8b44f7af)	G(d,a,b,c,m14,9,0xc33707d6)
F(c,d,a,b,m10,17,0xffff5bb1)	G(c,d,a,b,m3,14,0xf4d50d87)
F(b,c,d,a,m11,22,0x895cd7be)	G(b,c,d,a,m8,20,0x455a14ed)
F(a,b,c,d,m12,7,0x6b901122)	G(a,b,c,d,m13,5,0xa9e3e905)
F(d,a,b,c,m13,12,0xfd987193)	G(d,a,b,c,m2,9,0xfcefa3f8)
F(c,d,a,b,m14,17,0xa679438e)	G(c,d,a,b,m7,14,0x676f02d9)
F(b,c,d,a,m15,22,0x49b40821)	G(b,c,d,a,m12,20,0x8d2a4c8a)

图 3.28　MD5 4 轮函数运算表

第三轮	第四轮
P(a,b,c,d,m5,4,0xfffa3942)	Q(a,b,c,d,m0,6,0xf4292244)
P(d,a,b,c,m8,11,0x8771f681)	Q(d,a,b,c,m7,10,0x432aff97)
P(c,d,a,b,m11,16,0x6d9d6122)	Q(c,d,a,b,m14,15,0xab9423a7)
P(b,c,d,a,m14,23,0xfde5380c)	Q(b,c,d,a,m5,21,0xfc93a039)
P(a,b,c,d,m1,4,0xa4beea44)	Q(a,b,c,d,m12,6,0x655b59c3)
P(d,a,b,c,m4,11,0x4bdecfa9)	Q(d,a,b,c,m3,10,0x8f0ccc92)
P(c,d,a,b,m7,16,0xf6bb4b60)	Q(c,d,a,b,m10,15,0xffeff47d)
P(b,c,d,a,m10,23,0xbebfbc70)	Q(b,c,d,a,m1,21,0x85845dd1)
P(a,b,c,d,m13,4,0x289b7ec6)	Q(a,b,c,d,m8,6,0x6fa87e4f)
P(d,a,b,c,m0,11,0xeaa127fa)	Q(d,a,b,c,m15,10,0xfe2ce6e0)
P(c,d,a,b,m3,16,0xd4ef3085)	Q(c,d,a,b,m6,15,0xa3014314)
P(b,c,d,a,m6,23,0x04881d05)	Q(b,c,d,a,m13,21,0x4e0811a1)
P(a,b,c,d,m9,4,0xd9d4d039)	Q(a,b,c,d,m4,6,0xf7537e82)
P(d,a,b,c,m12,11,0xe6db99e5)	Q(d,a,b,c,m11,10,0xbd3af235)
P(c,d,a,b,m15,16,0x1fa27cf8)	Q(c,d,a,b,m2,15,0x2ad7d2bb)
P(b,c,d,a,m2,23,0xc4ac5665)	Q(b,c,d,a,m9,21,0xeb86d391)

图 3.28 （续）

对所有原消息分组运算完毕后，最后输出 V_a、V_b、V_c、V_d 的级联 $\{V_a|V_b|V_c|V_d\}$，即为128b(16B)的 MD5 值。

例如，第 2 轮的第 1 步(总第 17 步)为 $G(a,b,c,d,m_1,s,t_{17})$，表示：

$$a = b + ((a + g(b,c,d) + m_1 + t_{17}) >> s)$$

即为：

$$a = b + ((a + ((b\&d) \mid (c\&(\sim d))) + m_1 + t_{17}) >> s)$$

也即：

$$a = b + ((a + ((b\&d) \mid (c\&(\sim d))) + m_1 + 0\text{xf61e2562}) >> 5)$$

采用 MD5 加密字符串(例如加密口令)的样例如下(第一个例子为空字符串，即长度为 0 的字符串)：

```
md5 ("") = D41D8CD98F00B204E9800998ECF8427E
md5 ("a") = 0CC175B9C0F1B6A831C399E269772661
md5 ("abc") = 900150983CD24FB0D6963F7D28E17F72
md5 ("message digest") = F96B697D7CB7938D525A2F31AAF161D0
md5 ("abc…z") = C3FCD3D76192E4007DFB496CCA67E13B
```

可见 MD5 单向函数生成的哈希具备非常随机的外观，原字符串哪怕仅发生微小的变化，哈希值的很多位都会随之而变，杜绝了推测攻击的可能性。

3.4.4 RIPEMD 算法

RIPEMD(RACE Integrity Primitives Evaluation Message Digest)是一种单向函数算法，由 Hans Dobbertin、Antoon Bosselaers 和 Bart Preneel 等 3 人在 MD4 的基础上于 1996 年提出，与 MD5 一样都是着眼于改善 MD4 存在的安全缺陷。RIPEMD 算法共有 4 个标准 RIPEMD-128、RIPEMD-160、RIPEMD-256 和 RIPEMD-320，其对应输出长度分别为 16B、20B、32B 和 40B。让人难以置信的是 256 和 320 这两种标准只是在 128 和 160 的基础上修改了初始参数和 S-box 来达到输出为 256b 和 320b 的目的，所以，256 的强度和 128 相当，而 320 的强度和 160 位相当。其中 RIPEMD-160 在比特币系统的地址生成算法中得到运用，在此之前 RIPEMD 基本上处于默默无闻的状态。

RIPEMD-160 算法以 32b 字为计算单元，输入 16 个 32b 字 $X(i)$组成的分组，输出 5 个 32b 字(即 20B)的级联。输入消息的填充方式与 MD5 相同。

每个分组的运算进行 5 轮，每轮分别对 16 个 32b 字进行操作，共 80 步。每一步分为左、右两部分操作，执行逻辑均为：$A=(A+f(B,C,D)+X+K)\lll s+E$；$C=C\lll 10$($\lll s$ 为循环左移 s 位，$+$ 为模 2^{32} 加法)。

定义算法所用的 160 个 32b 常量如下所示，其中：$K(j)$用于左侧操作，$K'(j)$用于右侧操作，j 为 0～79 步操作步骤(每行为 1 轮)：

$K(j)=$ 0x00000000	0	$K'(j)=$ 0x50A28BE6	$2^{30}\cdot\sqrt[3]{2}$	$0\leqslant j\leqslant 15$
$K(j)=$ 0x5A827999	$2^{30}\cdot\sqrt{2}$	$K'(j)=$ 0x5C4DD124	$2^{30}\cdot\sqrt[3]{3}$	$16\leqslant j\leqslant 31$
$K(j)=$ 0x6ED9EBA1	$2^{30}\cdot\sqrt{3}$	$K'(j)=$ 0x6D703EF3	$2^{30}\cdot\sqrt[3]{5}$	$32\leqslant j\leqslant 47$
$K(j)=$ 0x8F1BBCDC	$2^{30}\cdot\sqrt{5}$	$K'(j)=$ 0x7A6D76E9	$2^{30}\cdot\sqrt[3]{7}$	$48\leqslant j\leqslant 63$
$K(j)=$ 0xA953FD4E	$2^{30}\cdot\sqrt{7}$	$K'(j)=$ 0x00000000	0	$64\leqslant j\leqslant 79$

定义 $r(j)$、$r'(j)$为输入分组的 32b 字 $X(i)$的选择下标。在不同的轮次及左右侧操作中，16 个字的计算次序各有不同。

在一种改进的 RIPEMD 算法中，设 $\rho(i)=\{7,4,13,1,10,6,15,3,12,0,9,5,2,14,11,8\}$($i=0,1,2,\cdots,15$)，又设 $\pi(i)=9i+5(\mathrm{mod}\ 16)$，则 5 轮运算中采用的 $r(j)$、$r'(j)$为(均为 mod 16 运算)：

	第 1 轮	第 2 轮	第 3 轮	第 4 轮	第 5 轮
左侧 $r(j)$：	j	$\rho(i)$	$[\rho(i)]^2$	$[\rho(i)]^3$	$[\rho(i)]^4$
右侧 $r'(j)$：	$\pi(i)$	$\rho(i)\times\pi(i)$	$[\rho(i)]^2\times\pi(i)$	$[\rho(i)]^3\times\pi(i)$	$[\rho(i)]^4\times\pi(i)$

但在标准的 RIPEMD 算法中，5 轮运算中采用的 $r(j)$、$r'(j)$固定定义为：

$r(j)=j$ $\quad r'(j)=5,14,7,0,9,2,11,4,13,6,15,8,1,10,3,12$ $\quad 0\leqslant j\leqslant 15$

$r(j)=7,4,13,1,10,6,15,3,12,0,9,5,2,14,11,8$ $\quad r'(j)=6,11,3,7,0,13,5,10,14,15,8,12,4,9,1,2$ $\quad 16\leqslant j\leqslant 31$

$r(j)=3,10,14,4,9,15,8,1,2,7,0,6,13,11,5,12$ $\quad r'(j)=15,5,1,3,7,14,6,9,11,8,12,2,10,0,4,13$ $\quad 32\leqslant j\leqslant 47$

$r(j)=1,9,11,10,0,8,12,4,13,3,7,15,14,5,6,2$ $\quad r'(j)=8,6,4,1,3,11,15,0,5,12,2,13,9,7,10,14$ $\quad 48\leqslant j\leqslant 63$

$r(j)=4,0,5,9,7,12,2,10,14,1,3,8,11,6,15,13$ $\quad r'(j)=12,15,10,4,1,5,8,7,6,2,13,14,0,3,9,11$ $\quad 64\leqslant j\leqslant 79$

各个步骤使用的循环左移位数 $s(j)$和 $s'(j)$（在一种改进的 RIPEMD 算法中移位次数的定义有所不同，且两侧相同）按标准定义为：

$s(j)=11,14,15,12,5,8,7,9,11,13,14,15,6,7,9,8$ $\quad s'(j)=8,9,9,11,13,15,15,5,7,7,8,11,14,14,12,6$ $\quad 0\leqslant j\leqslant 15$

$s(j)=7,6,8,13,11,9,7,15,7,12,15,9,11,7,13,12$ $\quad s'(j)=9,13,15,7,12,8,9,11,7,7,12,7,6,15,13,11$ $\quad 16\leqslant j\leqslant 31$

$s(j)=11,13,6,7,14,9,13,15,14,8,13,6,5,12,7,5$ $\quad s'(j)=9,7,15,11,8,6,6,14,12,13,5,14,13,13,7,5$ $\quad 32\leqslant j\leqslant 47$

$s(j)=11,12,14,15,14,15,9,8,9,14,5,6,8,6,5,12$ $\quad s'(j)=15,5,8,11,14,14,6,14,6,9,12,9,12,5,15,8$ $\quad 48\leqslant j\leqslant 63$

$s(j)=9,15,5,11,6,8,13,12,5,12,13,14,11,8,5,6$ $\quad s'(j)=8,5,12,9,12,5,14,6,8,13,6,5,15,13,11,11$ $\quad 64\leqslant j\leqslant 79$

每轮每个步骤的运算所用的 5 个非线性按位操作函数（⊕异或、∧与、∨或、～非）分别为：

$$f(j,x,y,z)=x\oplus y\oplus z \qquad (0\leqslant j\leqslant 15)$$

$$f(j,x,y,z)=(x\wedge y)\vee(\sim x\wedge z) \qquad (16\leqslant j\leqslant 31)$$

$$f(j,x,y,z)=(x\vee\sim y)\oplus z \qquad (32\leqslant j\leqslant 47)$$

$$f(j,x,y,z)=(x\wedge z)\vee(y\wedge\sim z) \qquad (48\leqslant j\leqslant 63)$$

$$f(j,x,y,z)=x\oplus(y\vee\sim z) \qquad (64\leqslant j\leqslant 79)$$

RIPEMD-160 算法开始时，首先初始化级联变量：$h_0=$0x67452301，$h_1=$0xEFCDAB89，$h_2=$0x98BADCFE，$h_3=$0x10325476，$h_4=$0xC3D2E1F0。算法流程如下：

（1）初始化临时变量。

$$A=A'=h_0,\quad B=B'=h_1,\quad C=C'=h_2,\quad D=D'=h_3,\quad E=E'=h_4$$

（2）对 16 个 32b 字 $X(i)$进行 5 轮共 80 步左、右两侧的运算。

```
for j = 0 to 79 {
    //左侧部分运算
```

$T = (A + f(j, B, C, D) + X[r(j)] + K(j)) \lll s(j) + E;$
$A = E; E = D; D = C \lll 10; C = B; B = T;$
//右侧部分运算
$T = (A' + f(79-j, B', C', D') + X[r'(j)] + K'(j)) \lll s'(j) + E';$
$A' = E'; E' = D'; D' = C' \lll 10; C' = B'; B' = T;$
}
$T = h_1 + C + D'; h_1 = h_2 + D + E'; h_2 = h_3 + E + A';$
$h_3 = h_4 + A + B'; h_4 = h_0 + B + C'; h_0 = T;$

(3) 若存在更多分组,回(1)继续运行;否则算法结束。

最后级联 h_0、h_1、h_2、h_3、h_4,即为 20B 的 RIPEMD-160 哈希。采用 RIPEMD-160 单向函数对字符串进行哈希的样例如下:

RIPEMD160("")=9C1185A5C5E9FC54612808977EE8F548B2258D31
RIPEMD160("a")=0BDC9D2D256B3EE9DAAE347BE6F4DC835A467FFE
RIPEMD160("abc")=8EB208F7E05D987A9B044A8E98C6B087F15A0BFC
RIPEMD160("message digest")=5D0689EF49D2FAE572B881B123A85FFA21595F36
RIPEMD160("abc…z")=F71C27109C692C1B56BBDCEB5B9D2865B3708DBC

3.4.5 SHA 算法

安全哈希算法(Secure Hash Algorithm,SHA)是单向函数加密算法之一,包括 SHA-1 和 SHA-2,SHA-2 具体分为 SHA-224、SHA-256、SHA-384 和 SHA-512,1995 年发布为美国联邦标准。SHA-1 在 SSL、SSH、S/MIME 和 IPSec 等许多安全协议中广为使用,被视为 MD5 的继任者,但人们也对其安全性存在质疑,相比之下 SHA-2 更为安全,其算法与 SHA-1 基本一致,SHA-256 在比特币系统中得到大量运用。

SHA-1 以 512b(32B)分组为处理单位,输出 160b 值;SHA-2 中 SHA-xyz 表示输出 xyzb 值,前两者分组大小与 SHA-1 相同为 512b(32B),后两者为 1024b(64B)。

SHA-1 算法如下(所有变量为 32b 字长,计算均为 mod 2^{32})。

对原消息的预处理与 MD5 算法相同:在原消息的尾部填充一个 1 和连续的 0,直到满足位长度对 512 求余的结果等于 448(即 $n \times 512 + 448$),其后附加 64b 原消息长度值(以 byte 为单位)。

设置级联变量初始值为 $h_0 := 0x67452301$,$h_1 := 0xEFCDAB89$,$h_2 := 0x98BADCFE$,$h_3 := 0x10325476$,$h_4 := 0xC3D2E1F0$。将原消息分为 512b 长度的分组(Chunk),依次

处理每个分组，直到处理完全部分组。

(1) 将分组分为 16 个 32b 字 $w[i](i=0,1,2,\cdots,15)$。

(2) 将 $w[i](i=0,1,2,\cdots,15)$扩展成 80 个 32b 字(<<循环左移，⊕异或)：

```
for i from 16 to 79
w[i] := (w[i-3]⊕w[i-8]⊕w[i-14]⊕w[i-16])<<1
```

(3) 临时变量赋值：$\{a, b, c, d, e\} = \{h_0, h_1, h_2, h_3, h_4\}$。

(4) 主处理程序(循环 80 轮次；& 与，| 或，~非)：

```
for i from 0 to 79
    if 0≤i≤19 then
        f := (b & c) | ((~b) & d); k := 0x5A827999
    else if 20≤i≤39 then
        f := b⊕c⊕d; k := 0x6ED9EBA1
    else if 40≤i≤59 then
        f := (b & c) | (b & d) | (c & d); k := 0x8F1BBCDC
    else if 60≤i≤79 then
        f := b⊕c⊕d; k := 0xCA62C1D6
    temp := (a<<5) + f + e + k + w[i]
    e := d; d := c; c := b<<30; b := a; a := temp
```

(5) 级联变量赋值：$\{h_0, h_1, h_2, h_3, h_4\} = \{h_0+a, h_1+b, h_2+c, h_3+d, h_4+e\}$。

(6) 若已为最后分组，则进行第(7)步；否则返回第(1)步处理下一分组。

(7) Hash 值为 h_0、h_1、h_2、h_3、h_4 顺序级联而成(160b)。

SHA-2 算法加强了各个字的位元混合程度，以提升安全强度。以 SHA-256 为例，算法如下(所有变量为 32B 字长，计算均为 mod 2^{32})。

初始化变量：h_0 := 0x6A09E667，h_1 := 0xBB67AE85，h_2 := 0x3C6EF372，h_3 := 0xA54FF53A，h_4 := 0x510E527F，h_5 := 0x9B05688C，h_6 := 0x1F83D9AB，h_7 := 0x5BE0CD19。

对 64 个常量赋值 $k[0,1,2,\cdots,63]$:=

0x428a2f98	0x71374491	0xb5c0fbcf	0xe9b5dba5	0x3956c25b	0x59f111f1	0x923f82a4	0xab1c5ed5
0xd807aa98	0x12835b01	0x243185be	0x550c7dc3	0x72be5d74	0x80deb1fe	0x9bdc06a7	0xc19bf174
0xe49b69c1	0xefbe4786	0x0fc19dc6	0x240ca1cc	0x2de92c6f	0x4a7484aa	0x5cb0a9dc	0x76f988da
0x983e5152	0xa831c66d	0xb00327c8	0xbf597fc7	0xc6e00bf3	0xd5a79147	0x06ca6351	0x14292967

续表

0x27b70a85	0x2e1b2138	0x4d2c6dfc	0x53380d13	0x650a7354	0x766a0abb	0x81c2c92e	0x92722c85
0xa2bfe8a1	0xa81a664b	0xc24b8b70	0xc76c51a3	0xd192e819	0xd6990624	0xf40e3585	0x106aa070
0x19a4c116	0x1e376c08	0x2748774c	0x34b0bcb5	0x391c0cb3	0x4ed8aa4a	0x5b9cca4f	0x682e6ff3
0x748f82ee	0x78a5636f	0x84c87814	0x8cc70208	0x90befffa	0xa4506ceb	0xbef9a3f7	0xc67178f2

(1) 将分组分为 16 个 32b 字 $w[i](i=0,1,2,\cdots,15)$。

(2) 将 $w[i](i=0,1,2,\cdots,15)$扩展成 64 个 32b 字(>>循环右移,>>>逻辑右移,⊕异或,+加):

for i from 16 to 63

$s_0 := (w[i-15] \gg 7) \oplus (w[i-15] \gg 18) \oplus (w[i-15] \ggg 3)$

$s_1 := (w[i-2] \gg 17) \oplus (w[i-2] \gg 19) \oplus (w[i-2] \ggg 10)$

$w[i] := w[i-16] + s_0 + w[i-7] + s_1$

(3) 临时变量赋值:$\{a, b, c, d, e, f, g, h\} = \{h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7\}$。

(4) 主处理程序(循环 64 轮次; & 与,|或,~非):

for i from 0 to 63

$s_0 := (a \gg 2) \oplus (a \gg 13) \oplus (a \gg 22)$

$maj := (a \,\&\, b) \oplus (a \,\&\, c) \oplus (b \,\&\, c)$

$t_2 := s_0 + maj;\ s_1 := (e \gg 6) \oplus (e \gg 11) \oplus (e \gg 25)$

$r := (e \,\&\, f) \oplus ((\sim e) \,\&\, g);\ t_1 := h + s_1 + r + k[i] + w[i]$

$h := g;\ g := f;\ f := e;\ e := d + t_1$

$d := c;\ c := b;\ b := a;\ a := t_1 + t_2$

(5) 中间变量赋值:

$$\{h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7\}$$

$$= \{h_0 + a, h_1 + b, h_2 + c, h_3 + d, h_4 + e, h_5 + f, h_6 + g, h_7 + h\}。$$

(6) 若已为最后分组,则进行第(7)步;否则返回第(1)步处理下一分组。

(7) Hash 值为 h_0、h_1、h_2、h_3、h_4、h_5、h_6、h_7 顺序链接而成(256b)。

SHA-224 与 SHA-256 的算法基本相同,除了 $h_0 \sim h_7$ 的初始值不同且 SHA-224 输出时截掉 h_7 的值(因此为 224b)。SHA-512 和 SHA-256 的结构相同,但是,SHA-512 处理 64B 字,执行 80 次循环,两者循环移位量不同。SHA-384 和 SHA-512 的不同点为:$h_0 \sim h_7$ 的初始值不同,SHA-384 输出时截掉了 h_6 和 h_7 的值。

3.5 单向陷门函数

单向陷门函数(trap-door function)的基本原理是：设 k 为密钥，函数 $y=f(x, k)$ 的计算很容易，而已知 y 和 k，则不可能计算出 $x=f^{-1}(y, k)$，即函数 f 具有单向性；但是，若存在一个 t 和函数 g，使得知道 t 就可以很容易地计算出 $x=g(y, t)$，则称函数 $y=f(x)$ 为单向陷门函数，而 t 就称为陷门(好比是函数的“后门”)。

非对称密钥加密技术中，计算 $f(x)$ 相当于用公钥加密，即使知道密文和公钥，也无法解密；私钥就是陷门，如果掌握了私钥，自然很容易完成解密。因此，非对称密钥加密算法就是一种典型的单向陷门函数。

3.6 量子密码

量子密码(quantum cryptography)并非一种特定的加密技术，而是利用量子力学的研究成果实现信息传输，具备防窃听、防破译的能力，所以对量子密码的准确理解应当是利用量子远距离传输信息的能力为保密通信服务。

量子是指微观粒子，与人们所熟悉的宏观物理现象和规律截然不同。维尔纳·海森堡(Werner Heisenberg)于 1927 年提出测不准原理(uncertainty principle)，即不可能同时知道一个微观粒子的位置和动量，粒子位置的不确定性必然大于或等于普朗克常数 $\hbar$ 除以 4π(即 $\Delta x\Delta p\geqslant\hbar/4\pi$)。粒子的位置确定越精确，其动量就越不精确，反之亦然。根据这一原理，微观粒子总是以不同的概率存在于不同的地方，对未知状态系统的每一次测量必然改变系统原来的状态，就是说宏观层面的任何观察和干扰，都会立刻改变量子态，引起其坍缩(collapse)，亦称为“观察者效应”。

“薛定谔的猫。”Alice 说，“这让我联想起那个量子力学的著名思维实验。”

Bob 两手一摊，说：“你指的是那个大部分人都听不懂的‘故事’吗？反正我似懂非懂的。”

“如果把猫关进一个封闭的盒子，里面有一个毒气释放装置，释放概率是 50%，那么过了一段时间后，你认为猫是死是活？”

“活着，或者死去。”Bob 像在念台词。

“或者说既是死的又是活的。如果不打开盒子，猫就处于死和活的叠加态，是宏

观世界中非死即活之外的另一种不确定状态。一旦打开盒子，这个状态就瞬间终止，观察者在满足好奇心的同时也破坏了系统的原有状态。”

“看来哈姆雷特王子真的挺悲催的，要回答他的千古一问，也许要付出他的王国因此而‘塌缩’的代价！”

其实在微观世界里还有比“薛定谔的猫”更加令人费解的事情，例如量子纠缠(quantum entanglement)。具有纠缠态的两个粒子无论相距多远，只要一个发生变化，另外一个也会瞬间发生变化。听上去与心灵感应(telepathy)差不多，却已得到实实在在的验证。

以这些理论为基础，20 世纪 80 年代起逐步演化出量子通信(quantum teleportation)技术。许多人之所以在理解量子通信时被弄得云里雾里，是把经典信息传输和量子信息传输这两种不同类型的传输方式混淆在了一起。

1. 量子通信的经典信息传输技术

1984 年，IBM 公司的 Bennett 和加拿大的 Brassard 首次提出了量子信息传输技术及其 BB84 协议，即利用单光子的偏振性来传输信息。

如图 3.29 所示，一个光子有四种偏振方向，可用横竖、斜向两种测量基(方法)来判别，并分别将水平、右下偏振的光子编码为 0，将垂直、右上偏振的光子编码为 1。如果用正确的测量基测量接收到的光子，则可解码正确的 0 或 1；如果用了错误的测量基，则解码正确率为 50%。显然，假如接收方随机选择测量基，则误码率为 50%(选错测量基概率)×50%(解码差错率)=25%。

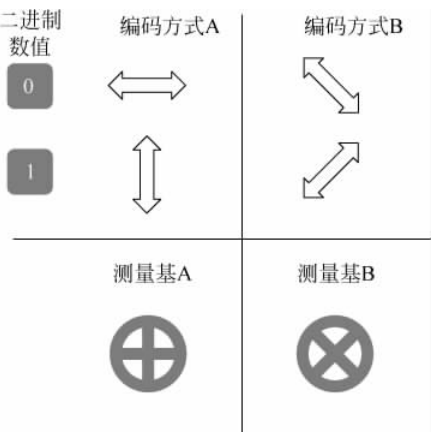

图 3.29 光子偏振测量和编码方法示意图

量子通信的经典信息传输 BB84 协议工作流程如下(示例如图 3.30 所示)：

(1) Alice 用单光子序列发送一串二进制信息(例如随机数密钥)。

(2) Bob 对接收到的每个光子随机选择一种测量基，得到解码结果。

(3) Bob 通过公开信道(可能被窃听)告知 Alice 自己选择的测量基序列。

(4) Alice 对比自己的编码，通过公开信道告诉 Bob 需要扔掉哪些错误的测量结果。

(5) 最终双方留下相同的筛选结果。

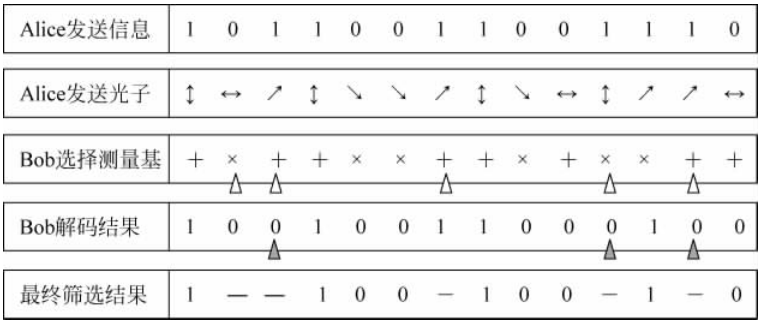

图 3.30 量子通信经典信息传输示例

如果窃听者选错了测量基(50%概率)，会改变光子偏振方向(观察者及其测量方法引起状态塌缩)导致 Bob 接收出错，进而导致 Bob 误码率增大，当误码率超过阈值，Alice 和 Bob 就会发现有窃听者而废弃本次传输。由于每位上有 25%的概率能发现窃听者，则如果发送 100 位，窃听者不被发现的概率仅为$(1-25\%)^{100}\approx 3.2\times 10^{-13}$。因此，采用 BB84 协议可安全传输一次性使用的密钥，用于下一步在任意信道上的加密通信(例如采用对称密钥 AES 算法)。

2. 量子通信的量子信息传输技术

1993 年提出的量子信息传输主要是指量子隐形传态，即先提取发送物的所有信息，然后将这些信息通过相互纠缠的量子的状态变化传送到接收地点，接收者依据这些信息，选取与构成原物完全相同的基本单元还原出复制品。可见这一方法完全不同于经典信息传输，不需要发送编码了信息比特的光子。

但这种方法过于理想化了，难以付诸实施，于是科学家提出了经典与量子相结合的隐形传态方案：将原物的信息分成经典信息和量子信息两部分，分别经由经典通道和量子通道传送给接收者；经典信息是发送者对原物进行某种测量而获得的，量子信息是发送者在测量中未提取的其余信息，接收者获得两种信息后，就可以制备出原物量子态的复制品。

量子信息传输技术完全隔绝了窃听者，具有完美的保密性。

另一个与量子相关的技术是量子计算(quantum computation)，最早由 P. Benioff 于

20 世纪 80 年代初期提出。在二进制量子计算机中，信息单元称为量子位(qubit)，除了处于 0 或 1 态外，还可处于叠加态(superposed state)，像薛定谔的猫，0 和 1 态线性叠加，各以一定的概率同时存在。普通计算机的 2b 寄存器同一时间仅能存储 4 个二进制数(00、01、10、11)中的一个，而量子计算机中的 2 个量子位寄存器可同时存储这四种状态的叠加状态。对于 n 个量子位而言，可以处于 2^n 种状态的叠加，意味着量子计算的每一步会对 2^n 种可能性同时做出操作，具有强大的并行处理能力。

量子计算对于现代密码学技术可能是个梦魇。例如 RSA 公钥体制的安全性是建立在大数的质数因子分解困难性之上，而量子计算机可以将原本困难之事变得轻而易举，从而可能使现有加密算法的安全性变得脆弱不堪。