第3章

区块链密码学基础

密码学是区块链的底层支撑技术，区块链功能的实现基于密码学基础，密码学也是保障区块链系统安全运行的重要理论支撑。最早期的比特币系统涉及的密码学知识较为简单，主要为数字签名、哈希等。随着区块链的去中心化应用的不断推广，越来越复杂的密码学技术，例如零知识证明、同态等被引入区块链领域。本章将从密码学基础角度，对区块链应用中会用到的密码学方案进行介绍。此外，本章还会进阶地描述一些较为复杂的密码学方法。

3.1 信息安全基础特性

信息安全的定义是，在任意一个信息系统中，其中所有硬件设施和内在单元，例如硬件、软件、数据、物理环境等都可以得到安全保护，并且有能力抵御偶然或恶意的破坏、更改、泄露的操作，或者有能力快速恢复，使得该信息系统能够连续可靠地运行，保证该系统能够为用户提供服务，信息服务不会中断，保证系统服务的正确性和连续性。

在20世纪初，信息安全的概念没有被提炼和定义。在经历了一个漫长的历史发展阶段后，在20世纪90年代初，信息安全的概念才得到了进一步深化。进入21世纪后，随着互联网的不断发展，各类信息系统以及信息技术不断增加，网络上的信息数据进入爆炸式增长阶段，信息安全问题也逐渐引起重视。现在，如何确保信息安全已成为全球重点关注的问题。国际上的发达国家对于信息安全的研究起步较早，美国、日本等发达国家或地区已经投入了大量的资源进行网络安全技术的研究。近年来，我国对信息安全的研究也已经上升到新的高度，取得了诸多的成果，并得以在实际中推广应用。

如图3.1所示，信息安全基础特性如下。

(1) 保密性(Anonymity)：指信息按给定要求不泄露给非授权的个人、实体的过程，且提供其利用的特性，即杜绝有用信息泄露给非授权个人或实体，强调有用信息只被授权对象使用的特征。

(2) 完整性(Integrity)：指信息在传输、交换、存储和处理的过程中，始终保持未被修改、未被破坏和未丢失，即始终保持信息原样性，使信息能正确生成、存储、传输，信息安全的基本特性就是指信息的完整性。

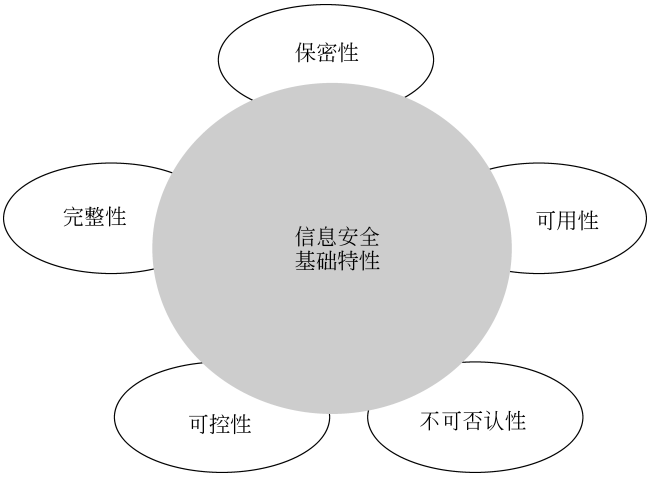

图 3.1　信息安全基础特性

(3) 可用性(Usability)：指信息可以被授权的实体正确访问，并可以按要求正常使用，或者在非正常情况下也能够恢复使用的特征，即在系统正常运行时能正确存取所需信息，当系统遭受攻击者攻击或破坏时，也能够迅速恢复并且投入使用，提供正常服务。信息的可用性是衡量信息系统在面向用户时的一种安全性能指标。

(4) 可控性(Controllability)：指信息以及系统具体内容在网络系统中流通时，系统可以实现有效控制的特性，即系统需要实现对其中的任何信息在一定传输范围和存放空间内有效控制。实施有效控制采用的方法包括传播站点以及传播内容监控这两类，最常用的方法还包括基于密码的托管政策，当加密算法交由第三方管理时，算法步骤就必须严格按规定可控执行。

(5) 不可否认性(Non-repudiation)：指任意通信双方在消息的交互过程中，所有参与者都不可能否认或抵赖本人的真实身份和其提供消息进行否认和抵赖，这需要对参与者本身的身份进行确定，保证参与者所提供的信息的真实同一性和其原样性。

3.2　密码学数学基础

本节将介绍密码学中所需的数论基础知识。如前所述，密码学基础是构建区块链技术的基石。现代密码理论中大部分方案是基于安全假设，即密码学方案的安全性可以被规约到一个困难问题，该困难问题在多项式时间内都无法被解决。困难问题设计的本质是基于数论，下面介绍有关数论的基础知识。

3.2.1　群基本定义及性质

1. 群的定义

在非空集合 G 上定义一个二元运算符“$\cdot$”，并且该集合满足下面 4 个属性。

(1) 封闭性：对于任意 $a, b \in G$，有 $a \cdot b \in G$。

(2) 结合律：对于任意 $a,b,c\in G$，有 $a\cdot b\cdot c=(a\cdot b)\cdot c=a\cdot(b\cdot c)$。

(3) 存在单位元：$\exists i\in G$，对于元素 $a\in G$，都可以得到 $a\cdot i=i\cdot a=a$。

(4) 存在逆元：对于任意 $a\in G$，存在一个元素 $a^{-1}\in G$，使得 $a\cdot a^{-1}=a^{-1}\cdot a=i$。$a^{-1}$ 则为元素 a 在集合 G 上的逆元。

如果某个代数系统满足上述所有性质，则称其为群(Group)，记为$(G,\cdot)$。

交换律：对任意的 $a,b\in G$，有 $a\cdot b=b\cdot a$。

如果某个群除了满足以上封闭性、结合律、存在单位元、存在逆元 4 个属性之外，还满足交换律，则称该群为交换群或者阿贝尔群。

2. 群的性质

(1) 群中的单位元是唯一的。

(2) 群中任一元素的逆元是唯一的。

(3) 对于任意的 $a,b,c\in G$，如果有 $a\cdot b=a\cdot c$，则有 $b=c$。

(4) 对于任意的 $a,b\in G$，有$(a\cdot b)^{-1}=b^{-1}\cdot a^{-1}$。

(5) 对于任意的 $a\in G$，元素 a 与自身的 n 次运算，其中 $n\in\mathbf{N}$，可以表示为

$$a^n=a\cdot a\cdot\cdots\cdot a$$

这里需要注意 $n=0$ 的情况，定义 $a^0=i$，其中元素 i 是 G 中的单位元。

如果分别使用普通乘法和普通加法的符号来定义 G 中的二元运算“+”和“·”，则对于乘法运算，a^n 表示 a 的 n 次幂，加法运算 $na=a+a+\cdots+a$，可以得到如表 3.1 所示的性质。

表 3.1　普通乘法与普通加法的运算性质

普 通 乘 法	普 通 加 法
$a^na^m=a^{m+n}$	$na+ma=(n+m)a$
$(a^n)^m=a^{mn}$	$m(na)=mna$
$a^{-n}=(a^{-1})^n$	$(-n)a=n(-a)$

如果群 G 中包含的所有元素个数是一个有限整数，则称 G 为有限群，否则称为无限群。有限群 G 中元素的个数，称为这个有限群的阶。若 G 是一个有限群，它的阶表示为$|G|$。

特别地，如果一个集合在某种运算上是一个群，但其在其他运算上未必是一个群。

3. 有限域

有限域指的是包含有限个元素的域。显然，$\mathbf{Z}_q$(q 为素数)就是一个有限域。可以得到关于有限域的性质为，有限域的阶必为素数的幂。举例而言，假设 F 是一个有限域，则存在素数 p 和正整数 n，使得$|F|=p^n$，将这个有限域记为 F_p^n 或 GF(p^n)。反之，对于每一个这样的素数幂 p^n，都存在这样唯一的阶为 p^n 的有限域。当 $n=1$ 时，把有限域 GF(p)称为素数域。密码学中普遍采用的域是阶为 p 的素数域 GF(p)和特征为 2 的 GF(2^m)域。

3.2.2 整数群、椭圆曲线群和双线性映射群

1. 整数群

由整数集合 $\mathbf{Z}$ 构成的群，即为整数群。整数群有如下的一些性质。

(1) 整数集合 $\mathbf{Z}$ 在加法运算上可以构成阿贝尔群，写作($\mathbf{Z}$,+)，它的全部元素都是由 1 生成的，0 是这个群的单位元，每一个元素的逆元是自身的相反数。

(2) 整数集合 $\mathbf{Z}$ 在乘法运算上不能够构成群，因为除了元素 1 和 −1 外，其他所有元素的乘法逆元都不存在。

特别地，($\mathbf{Z}_p$,+)称为模 p 的整数群，它包含的元素是模 p 的 p 个剩余类{[0],[1],[2],…,[$p-1$]}。显然，$\mathbf{Z}_p$ 是一个加法循环群，$\mathbf{Z}_p=<[1]>$，其中[1]就是它的一个生成元。对于任何与 p 互素的整数 r，[r]都是 $\mathbf{Z}_p$ 的生成元。

2. 椭圆曲线群

椭圆曲线密码(Elliptic Curves Cryptography，ECC)是基于椭圆曲线数学理论的算法，属于公钥加密算法，是迄今为止被实践证明安全有效的三类公钥密码体制之一，其以高效性能在实际系统中被广泛应用。椭圆曲线离散对数问题的困难性决定了 ECC 算法的安全性。椭圆曲线离散对数问题是一个比大整数因子分解问题(RSA 算法基础)以及离散对数问题(DSA 算法基础)更困难的问题。相比于 RSA 算法，ECC 算法的优势在于其能够用更少的位长度获得与 RSA 同等强度的安全性，因此 ECC 算法可以节约计算开销、密钥管理存储开销以及通信带宽，使得 ECC 算法可以适用于一些计算能力较弱或者对移动性要求较高的设备，例如手机、智能卡等小型或移动型设备。

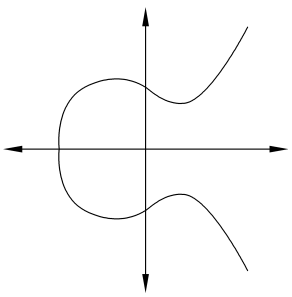

图 3.2 椭圆曲线示意图

所有满足椭圆曲线方程点的集合就构成了椭圆曲线群，下面将介绍 3 种基于不同域的椭圆曲线。

1) 实数域上的椭圆曲线

椭圆曲线并非字面上理解的椭圆，只是它的曲线方程与计算椭圆的周长方程类似(见图 3.2)。一般来说，椭圆曲线指的就是维尔斯特拉斯(Weierstrass)方程，即

$$y^2+axy+by=x^3+cx^2+dx+e \tag{3.1}$$

椭圆曲线是由满足方程的全体解集(x,y)再加上一个无穷远点 O 构成的集合，该方程中的参数 a,b,c,d 是满足某些特定条件的实数。上述曲线方程通过坐标转化为以下的简化式，即

$$y^2=x^3+ax+b \tag{3.2}$$

由这个方程确定的椭圆曲线记为 $E(a,b)$或 E。椭圆曲线是关于 x 轴对称的。椭圆曲线的加法运算可以定义为：如果椭圆曲线上 3 个点位于同一直线，那么它们的和为 O。椭圆曲线上的加法也具有加法运算的一般性质，如交换律、结合律等。

2) 有限域 GF(p)上的椭圆曲线

有限域 GF(p)上的椭圆曲线定义为满足式(3.3)同余式的所有几何解(x,y)∈GF(p)和一个无穷远点 $O=(x,\infty)$构成的集合：

$$y^2 \equiv x^3 + ax + b \pmod{p} \tag{3.3}$$

式中，p 是一个大于 2^{160}的素数；a,b∈GF(p)是满足 $4a^3+27b^2 \neq 0 \pmod{p}$的常数。这类椭圆曲线通常用 $E_p(a,b)$来表示，即

$$E_p(a,b)=\{(x,y) \cup O \mid y^2 \equiv x^3 + ax + b \pmod{p}\} \tag{3.4}$$

该椭圆曲线上只有有限个点数 N(称为椭圆曲线的阶，包含有无穷远点)。N 越大，安全性越高，即群中元素越多，越能抵抗穷举搜索攻击。

比特币系统的区块链实现采用的是椭圆曲线 secp256k1，secp256k1 是基于有限域 GF(p)上的椭圆曲线，其中 $p=2^{256}-2^{32}-2^9-2^8-2^7-2^6-2^4-1$。

3) 有限域 GF(2^m)上的椭圆曲线

有限域 GF(2^m)中的元素是 m 位的字符串，这些字符串可以表示为系数在 GF(2)上的多项式：

$$\{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0 : a_i \in \{0,1\}\} \tag{3.5}$$

GF(2^m)上的椭圆曲线是定义为满足同余式(3.6)的所有几何解(x,y)∈GF(2^m)和一个无穷远点 $O=(x,\infty)$构成的集合。

$$y^2 + xy = x^3 + ax^2 + b \tag{3.6}$$

其中，$m>160$，a,b∈GF(2^m)且 $b\neq 0$。这类椭圆曲线通常可以用 $E_{2^m}(a,b)$来表示，即

$$E_{2^m}(a,b)=\{(x,y) \cup O \mid y^2 + xy = x^3 + ax^2 + b\} \tag{3.7}$$

3. 双线性映射群

双线性映射也称双线性配对，一个双线性映射是由两个向量空间中的元素，一起生成第三个向量空间中的一个元素的函数，并且该函数对每个参数都是线性的。总而言之，给出一个双线性群组的定义(G_1,G_2,G_T,p,g_1,g_2,e)，其中 G_1,G_2 和 G_T 是阶为素数 p 的乘法循环群，g_1 和 g_2 分别是 G_1 和 G_2 的一个生成元。一个映射 $e:G_1\times G_2 \rightarrow G_T$ 称为双线性映射，当它满足如下 3 个性质。

(1) 双线性：对于 $\forall a,b \in \mathbf{Z}_p$，均有 $e(g_1^a, g_2^b)=e(g_1,g_2)^{ab}$。

(2) 非退化性：$\exists g_1 \in G_1, g_2 \in G_2$，使得 $e(g_1,g_2)\neq 1$。

(3) 可计算性：对于 $\forall u \in G_1, v \in G_2$，存在有效的算法计算 $e(u,v)$。

若 $G_1=G_2$，则称这个双线性映射是对称的，否则是非对称的。

3.3　密码学困难问题

在密码学中，几乎所有的加密系统的安全性都建立在计算问题的困难性假设基础上。这些困难性假设问题，经过专家的长期研究，认为从群论的角度上是计算困难的，因为它与计算复杂度理论存在着密切的联系。随着计算机技术的不断发展，这些假设是否成立，仍然是密码学的一个重要的研究课题。接下来将主要介绍有关群论方面的

困难性问题。

3.3.1 离散对数问题

基于离散对数的公钥加密算法作为常用的公钥加密算法之一，其加密算法的安全性要远远高于基于大整数分解的 RSA 加密算法，而且离散对数问题(Discrete Logarithm Problem，DLP)的计算复杂度要比大整数分解问题更高。经过多年研究，人们无法构造出一个概率多项式时间来解决离散对数问题的方案。

1. 离散对数问题定义

离散对数问题的定义为：当给定一个素数 p，以及有限域 $\mathbf{Z}_p$ 上的一个本原元 a 时，对 $\mathbf{Z}_p$ 上的任意正整数 b，都可以找到唯一的整数 c，$0\leqslant c\leqslant p-2$，使得 $a^c\equiv b(\mathrm{mod}\ p)$，通常用 $\log_a b$ 来表示 c。一般而言，如果仔细选择 p，则可以认为该问题是难解的，目前还没有找到可以在多项式时间内计算离散对数问题的算法。为了抵抗已知的攻击，选择的素数 p 需要满足以下两个条件：一是为至少 150 位的十进制整数；二是 $p-1$ 至少有一个大的素数因子。

2. 离散对数问题性质

离散对数问题的困难性与有限域上的生成元无关，任何可以计算 $\log_a b$ 的离散对数算法同样可以用来计算任意其他底数的离散对数问题。

3.3.2 大整数分解问题

大整数分解问题(Integer Factorization Problem，IFP)指的是对于任意给定的两个大素数的乘积，找出该乘积的因子，这个问题是困难的。大整数分解问题是数论中的一个基本问题，普遍认为不存在高效的概率多项式时间算法能够分解一个大整数，因此它是一个非常重要的研究问题。

将大整数分解问题具体定义为：任意选定两个长度为 λ 的大素数 p 和 q，并计算 $n=pq$。对于给定 n，若存在一个可忽略的函数 $\mathrm{negl}(\lambda)$，使得 n 能够被分解为 p 乘以 q 的概率不大于 $\mathrm{negl}(\lambda)$，则可以称该大整数 n 的分解问题是一个困难问题。

3.3.3 CDH/DDH 问题

计算 Diffie-Hellman(Computational Diffie-Hellman，CDH)假设是 Whitfield Diffie 和 Martin Hellman 两位教授在构造密钥协议协商的时候提出来的。一般把任意群 G 的三元素组 (g^x, g^y, g^{xy}) 称为 DH 组。对于任意阶为素数 q 的循环群 G，$g\in G$ 是一个生成元，二元函数 $\mathrm{dh}_g: G^2\rightarrow G$ 的定义为：随机选择两个元素 $X, Y\in G$，令 $X=g^x$，$Y=g^y$，计算 $\mathrm{dh}_g(X,Y)=g^{xy}$。CDH 问题的困难性指的是在已知 G, g, g^x, g^y 的情况下，计算二元函数 dh_g 是困难的，即存在一个可忽略的函数 $\mathrm{negl}(\lambda)$，使得计算出二元函数 dh_g 的概率不大于 $\mathrm{negl}(\lambda)$。

判定 Diffie-Hellman(Decisional Diffie-Hellman，DDH)假设是 CDH 假设的判定形

式，也就是计算问题对应的判定问题，用来判断不可分辨性。DDH 假设具体定义如下：对于任意阶为素数 q 的循环群 G，$g \in G$ 是一个生成元，随机选取 3 个元素 $X, Y, Z \in G$，其中 $X = g^x$，$Y = g^y$，且有 $\mathrm{dh}_g = g^{xy}$。存在一个可忽略的函数 negl(λ)，使得区分 $Z = \mathrm{dh}_g$ 或者 Z 是随机选取的概率不大于 negl(λ)。

3.3.4　BDH/DBDH 问题

双线性 Diffie-Hellman（Bilinear Diffie-Hellman，BDH）问题是对给定的 3 个元素 g^a, g^b, g^c，计算 $e(g,g)^{abc}$。具体定义如下。

计算 BDH 假设是对于任意阶为素数 q 的循环群 G_1, G_2, G_3，其中 $g \in G_1$，$h \in G_2$ 是两个生成元，存在一个可高效计算的同构 $\varphi: G_1 \to G_2$，使得对于生成元 g，h，有 $\varphi(g) = h$。随机选择 $a, b, c \in \mathbf{Z}_q$，有 g^a, g^c, h^a, h^b，BDH 问题的难解性在于计算 $e(g,h)^{abc}$ 是困难的，即存在一个可忽略的函数 negl(λ)，使得计算出 $e(g,h)^{abc}$ 的概率不大于 negl(λ)。

同样地，BDH 假设的判定形式是 DBDH（Decisional Bilinear Diffie-Hellman）假设。对于任意阶为素数 q 的循环群 G_1, G_2, G_3，其中 $g \in G_1$，$h \in G_2$ 是两个生成元，存在一个可高效计算的同构 $\varphi: G_1 \to G_2$，使得对于生成元 g，h，有 $\varphi(g) = h$。随机选择 $a, b, c \in \mathbf{Z}_q$，$W \in G_3$，已知 g^a, g^c, h^a, h^b 和 $e(g,h)^{abc}$，存在一个可忽略的函数 negl(λ)，使得区分 $W = e(g,h)^{abc}$ 或者 W 是随机选取的概率不大于 negl(λ)。

3.4　对称密码体制

3.4.1　基本概念

对称密码体制是一种传统的密码体制，也称私钥密码体制。在对称密码体制中，加密和解密采用相同的密钥。由于加解密采用的密钥相同，所以通信的双方必须选择和保存相同的密钥，并且双方密钥的传输必须通过安全信道，保证不会被攻击者窃听或者截获，此外双方也必须信任对方都不会将密钥泄露出去。以此实现数据的保密性和完整性。

对称密码模型如图 3.3 所示，基本元素包含有原始的明文、加密算法、密钥、密文、解密算法、信息发送者以及信息接收者。其中，传输加密信息的信道是不安全且公开的，因此发送者用于加密消息的密钥则需要通过安全的密钥交换渠道来传输给接收者。

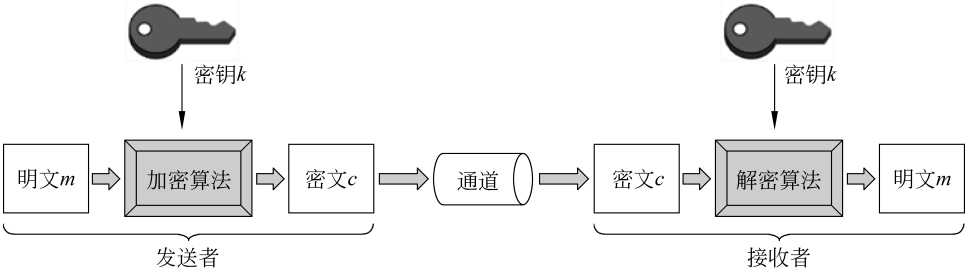

图 3.3　对称密码模型

发送者通过加密算法 E 根据输入的消息 m 和密钥 k 生成密文 c：

$$c = E_k(m) \tag{3.8}$$

接收者通过解密算法 D 根据输入的密文 c 和共同协商的密钥 k 恢复出明文 m：

$$m = D_k(c) \tag{3.9}$$

攻击者可以基于不安全的公开信道观察密文 c，它无法接触到明文 m 或者密钥 k，但攻击者可以试图恢复明文 m 或者密钥 k。由于加密算法和解密算法是公开的，假设攻击者对某个特定的消息感兴趣，他可以针对性地通过分析方法（例如穷举攻击）分析出明文 m 或者密钥 k。

通过上述的例子可以知道，对称密码体制的安全性主要取决于以下两种因素。

（1）加密算法：加密算法无须保密，使得仅仅通过密文以及算法，而不通过密钥，就想破译出明文是非常困难的。

（2）密钥：密钥的安全性决定了明文信息的安全性，因此密钥空间必须保证足够大。

对称密码体制的优势是不仅速度快，并且具有较高的保密强度，一些常用的对称加密算法明显地快于当前任何非对称加密算法。此外，有些对称加密方案甚至可以达到经受国家级破译力量的分析和攻击的安全性。对称密码体制的缺点在于，对称密码体制的安全性是建立在它的密钥必须通过安全可靠的路径进行传输之上的，这样对称密码的密钥安全传输和管理成了影响系统安全性的关键因素。

国际上经典的对称加密算法包括数据加密标准（Data Encryption Standard，DES）算法、三重数据加密标准（Triple DES，或称为三重 DES）算法、国际数据加密算法（IDEA）、RC5 算法以及高级加密标准（Advanced Encryption Standard，AES）算法等。DES 算法由美国国家标准局提出，主要应用场景是银行业的电子资金转账（EFT）。DES 是一个分组密码算法，采用的是替换和移位的方法，并使用 56 位的密钥，用于处理 64 位的明文数据。DES 算法的优势是加解密速度快、易于软件实现。但是，DES 算法的缺点是密钥过短，56 位的密钥长度会影响它的保密强度。为了克服这一缺点，提出了一系列的 DES 变形算法。如 Triple DES 算法，它使用两个独立的 56 位密钥对交换的信息进行 3 次加密，从而使其有效长度达到 112 位。类似于 DES 算法，IDEA 也是一种对称的数据块加密算法，采用了 128 位的密钥和 8 个循环，每轮加密都使用从完整的加密密钥中生成的一个子密钥，同一算法既可用于加密又可用于解密。该算法本身的密码结构，使其具有对采用软件实现和采用硬件实现同样快速的优势。此外，RC5 是一种具有可变字长、可变轮数和可变密钥长度的对称分组密码算法。该算法的特点是使用依赖于数据的循环，安全性依赖于循环运算和不同运算的混合使用。AES 算法在密码学中又称 Rijndael 加密法，已经作为美国联邦政府采用的一种区块加密标准。该标准用来替代原先的 DES 算法，现如今已经被多方分析证明安全，且被全世界所使用。从 2006 年开始，AES 算法已经正式取代了 DES 算法，成为对称密钥加密中最流行的算法之一。

3.4.2　对称加密算法 AES

美国国家标准与技术研究所(National Institute of Standards and Technology，NIST)在 1997 年发起，向全球所有研究人员提出征集 AES 算法，用于取代 DES 算法。NIST 首先要求算法的分组长度为 128 位，允许 3 个不同的密钥大小：128 位、192 位或 256 位，同时算法必须是可以公开的。2000 年，NIST 通过对候选算法的安全性(是否基于稳定的数学基础、无算法弱点、具备测试算法抗密码分析的强度、测试算法输出的随机性)、性能(是否能在多种平台上以较快的速度实现)、大小(算法不能占用大量的存储空间和内存)、实现特性(灵活性、可扩展性、硬件和软件适应性、算法的简单性等)等方面进行综合评估，最终采用了两位比利时研究者 Vincent Rijmen 和 Joan Daemen 所提出的 Rijndael 算法，NIST 于 2001 年正式发布了 AES 算法。

AES 算法作为分组密码算法的一种，即通过把明文分成一组一组的等长数据，每次加密一组数据，直到加密完整个明文。在 AES 算法中，明文的分组长度只能是 128 位，也就是说，每个分组为 16 字节(每字节 8 位)。密钥的长度则可以使用 128 位、192 位或 256 位。一般而言，加密轮数取决于密钥长度，即密钥长度不同，加密轮数也不同，如表 3.2 所示。

表 3.2　AES 密钥标准

标　　准	密钥长度(32 位)	分组长度(32 位)	加密轮数
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

AES 算法涉及 4 种操作步骤：字节替代、行移位、列混淆和轮密钥加。下面给出 4 种操作的详细描述。

1. 字节替代

字节替代(SubBytes)是一种非线性变换，主要功能是通过 $\boldsymbol{S}$ 盒来完成一字节到另一字节的映射。其中，$\boldsymbol{S}$ 盒用于提供密码算法的混淆性。通过一个简单的对 $\boldsymbol{S}$ 盒查询的操作，它将输入或者中间态的每一字节映射成为另一字节。这种映射方法具体表述为：将输入的字节的高 4 位作为 $\boldsymbol{S}$ 盒的行值，低 4 位作为列值，然后输出 $\boldsymbol{S}$ 盒或 $\boldsymbol{S}^{-1}$ 盒中对应行和列的元素。表 3.3 为 $\boldsymbol{S}$ 盒，表 3.4 为 $\boldsymbol{S}^{-1}$ 盒($\boldsymbol{S}$ 盒的逆)。

$\boldsymbol{S}$ 盒和 $\boldsymbol{S}^{-1}$ 盒都为 16×16 的矩阵，包含了 8 位所能表示的 256 个数的一个置换，完成一个 8 位输入、8 位输出的等位映射。字节替换的简单示意如式(3.10)：

$$\begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{pmatrix} \rightarrow \boldsymbol{S} \text{ 盒} \rightarrow \begin{pmatrix} S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \end{pmatrix} \tag{3.10}$$

表 3.3 *S* 盒

列	行															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	0x63	0x7c	0x77	0x7b	0xf2	0x6b	0x6f	0xc5	0x30	0x01	0x67	0x2b	0xfe	0xd7	0xab	0x76
1	0xca	0x82	0xc9	0x7d	0xfa	0x59	0x47	0xf0	0xad	0xd4	0xa2	0xaf	0x9c	0xa4	0x72	0xc0
2	0xb7	0xfd	0x93	0x26	0x36	0x3f	0xf7	0xcc	0x34	0xa5	0xe5	0xf1	0x7l	0xd8	0x31	0xl5
3	0x04	0xc7	0x23	0xc3	0xl8	0x96	0x05	0x9a	0x07	0xl2	0x80	0xe2	0xeb	0x27	0xb2	0x75
4	0x09	0x83	0x2c	0xla	0xlb	0x6e	0x5a	0xa0	0x52	0x3b	0xd6	0xb3	0x29	0xe3	0x2f	0x84
5	0x53	0xd1	0x00	0xed	0x20	0xfc	0xb1	0x5b	0x6a	0xcb	0xbe	0x39	0x4a	0x4c	0x58	0xcf
6	0xd0	0xef	0xaa	0xfb	0x43	0x4d	0x33	0x85	0x45	0xf9	0x02	0x7f	0x50	0x3c	0x9f	0xa8
7	0x51	0xa3	0x40	0x8f	0x92	0x9d	0x38	0xf5	0xbc	0xb6	0xda	0x21	0xl0	0xff	0xf3	0xd2
8	0xcd	0x0c	0xl3	0xec	0x5f	0x97	0x44	0xl7	0xc4	0xa7	0x7e	0x3d	0x64	0x5d	0xl9	0x73
9	0x60	0x81	0x4f	0xdc	0x22	0x2a	0x90	0x88	0x46	0xee	0xb8	0xl4	0xde	0x5e	0x0b	0xdb
A	0xe0	0x32	0x3a	0x0a	0x49	0x06	0x24	0x5c	0xc2	0xd3	0xac	0x62	0x91	0x95	0xe4	0x79
B	0xe7	0xc8	0x37	0x6d	0x8d	0xd5	0x4e	0xa9	0x6c	0x56	0xf4	0xea	0x65	0x7a	0xae	0x08
C	0xba	0x78	0x25	0x2e	0xlc	0xa6	0xb4	0xc6	0xe8	0xdd	0x74	0xlf	0x4b	0xbd	0x8b	0x8a
D	0x70	0x3e	0xb5	0x66	0x48	0x03	0xf6	0x0e	0x61	0x35	0x57	0xb9	0x86	0xc1	0xld	0x9e
E	0xe1	0xf8	0x98	0xl1	0x69	0xd9	0x8e	0x94	0x9b	0xle	0x87	0xe9	0xce	0x55	0x28	0xdf
F	0x8c	0xal	0x89	0x0d	0xbf	0xe6	0x42	0x68	0x41	0x99	0x2d	0x0f	0xb0	0x54	0xbb	0xl6

表 3.4　S^{-1}盒

列	行															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	0x52	0x09	0x6a	0xd5	0x30	0x36	0xa5	0x38	0xbf	0x40	0xa3	0x9e	0x81	0xf3	0xd7	0xfb
1	0x7c	0xe3	0x39	0x82	0x9b	0x2f	0xff	0x87	0x34	0x8e	0x43	0x44	0xc4	0xde	0xe9	0xcb
2	0x54	0x7b	0x94	0x32	0xa6	0xc2	0x23	0x3d	0xee	0x4c	0x95	0x0b	0x42	0xfa	0xc3	0x4e
3	0x08	0x2e	0xa1	0x66	0x28	0xd9	0x24	0xb2	0x76	0x5b	0xa2	0x49	0x6d	0x8b	0xd1	0x25
4	0x72	0xf8	0xf6	0x64	0x86	0x68	0x98	0xl6	0xd4	0xa4	0x5c	0xcc	0x5d	0x65	0xb6	0x92
5	0x6c	0x70	0x48	0x50	0xfd	0xed	0xb9	0xda	0x5e	0xl5	0x46	0x57	0xa7	0x8d	0x9d	0x84
6	0x90	0xd8	0xab	0x00	0x8c	0xbc	0xd3	0x0a	0xf7	0xe4	0x58	0x05	0xb8	0xb3	0x45	0x06
7	0xd0	0x2c	0xle	0x8f	0xca	0x3f	0x0f	0x02	0xc1	0xaf	0xbd	0x03	0x01	0xl3	0x8a	0x6b
8	0x3a	0x91	0xl1	0x41	0x4f	0x67	0xdc	0xea	0x97	0xf2	0xcf	0xce	0xf0	0xb4	0xe6	0x73
9	0x96	0xac	0x74	0x22	0xe7	0xad	0x35	0x85	0xe2	0xf9	0x37	0xe8	0xlc	0x75	0xdf	0x6e
A	0x47	0xf1	0xla	0x71	0xld	0x29	0xc5	0x89	0x6f	0xb7	0x62	0x0e	0xaa	0xl8	0xbe	0xlb
B	0xfc	0x56	0x3e	0x4b	0xc6	0xd2	0x79	0x20	0x9a	0xdb	0xc0	0xfe	0x78	0xcd	0x5a	0xf4
C	0xlf	0xdd	0xa8	0x33	0x88	0x07	0xc7	0x31	0xb1	0xl2	0xl0	0x59	0x27	0x80	0xec	0x5f
D	0x60	0x51	0x7f	0xa9	0xl9	0xb5	0x4a	0x0d	0x2d	0xe5	0x7a	0x9f	0x93	0xc9	0x9c	0xef
E	0xa0	0xe0	0x3b	0x4d	0xae	0x2a	0xf5	0xb0	0xc8	0xeb	0xbb	0x3c	0x83	0x53	0x99	0x61
F	0x17	0x2b	0x04	0x7e	0xba	0x77	0xd6	0x26	0xe1	0x69	0xl4	0x63	0x55	0x21	0x0c	0x7d

与 DES 算法的 $\boldsymbol{S}$ 盒相比，AES 算法的 $\boldsymbol{S}$ 盒具有严格的数学推导和计算，能进行代数学上的定义，涉及了有限域 GF(2^8)和系数在 GF(2^8)上的多项式。

2. 行移位

行移位(ShiftRows)的目的是实现一个 4×4 矩阵内部字节之间的置换，完成基于行的循环位移操作，其具体操作：保持第一行不变，循环左移第二行 1 字节，循环左移第三行 2 字节，循环左移第四行 3 字节。变换如式(3.11)：

$$\begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{pmatrix} \rightarrow \begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,1} & S_{1,2} & S_{1,3} & S_{1,0} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,3} & S_{3,0} & S_{3,1} & S_{3,2} \end{pmatrix} \tag{3.11}$$

3. 列混淆

列混淆(MixColumns)是将每一列的值与固定多项式进行乘法运算，为了确保运算结果不会溢出定义域，运算中涉及的加法和乘法都是定义在 GF(2^8)上的。列混淆可以用式(3.12)进行描述。

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{pmatrix} = \begin{pmatrix} S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \end{pmatrix} \tag{3.12}$$

在式(3.12)中，为确保每列的所有字节具有良好的混淆性，采用的矩阵的系数是基于码字间的最大距离的线性编码。经过几轮列混淆变换和行移位变换后，所有的输入位与所有的输出位线性相关。逆向列混淆变换可由式(3.13)矩阵乘法定义。

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{pmatrix} = \begin{pmatrix} S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \end{pmatrix} \tag{3.13}$$

4. 轮密钥加

轮密钥加的(AddRoundKey)目的是将轮密钥与状态进行逐位异或操作如式(3.14)：

$$\begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{pmatrix} \oplus \begin{pmatrix} W_{0,0} & W_{0,1} & W_{0,2} & W_{0,3} \\ W_{1,0} & W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,0} & W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,0} & W_{3,1} & W_{3,2} & W_{3,3} \end{pmatrix} = \begin{pmatrix} H_{0,0} & H_{0,1} & H_{0,2} & H_{0,3} \\ H_{1,1} & H_{1,2} & H_{1,3} & H_{1,0} \\ H_{2,2} & H_{2,3} & H_{2,0} & H_{2,1} \\ H_{3,3} & H_{3,0} & H_{3,1} & H_{3,2} \end{pmatrix} \tag{3.14}$$

由此可见，轮密钥加的逆运算与正向的轮密钥加运算是完全一致的，这是由于异或运算后进行逆操作得到其自身。轮密钥加变换虽然简单明了，但却能够影响 $\boldsymbol{S}$ 数组中的每

一位，满足对于对称加密算法要求的对于明文处理的扩散性。

以密钥长度为 128 位的 AES 算法为例，图 3.4 给出了 AES 的加解密流程，从图中可以看出：解密算法分别对应加密算法中的每一步的逆操作，即加解密所有操作的顺序正好是相反的。正是由于这些过程，同时加上每步的加密算法与解密算法的操作均为互逆，共同保证了算法的加解密的正确性。加解密中每轮的密钥分别由种子密钥经过密钥扩展算法得到，算法中 16 字节的明文、密文和轮子密钥都用一个 4×4 的矩阵表示。

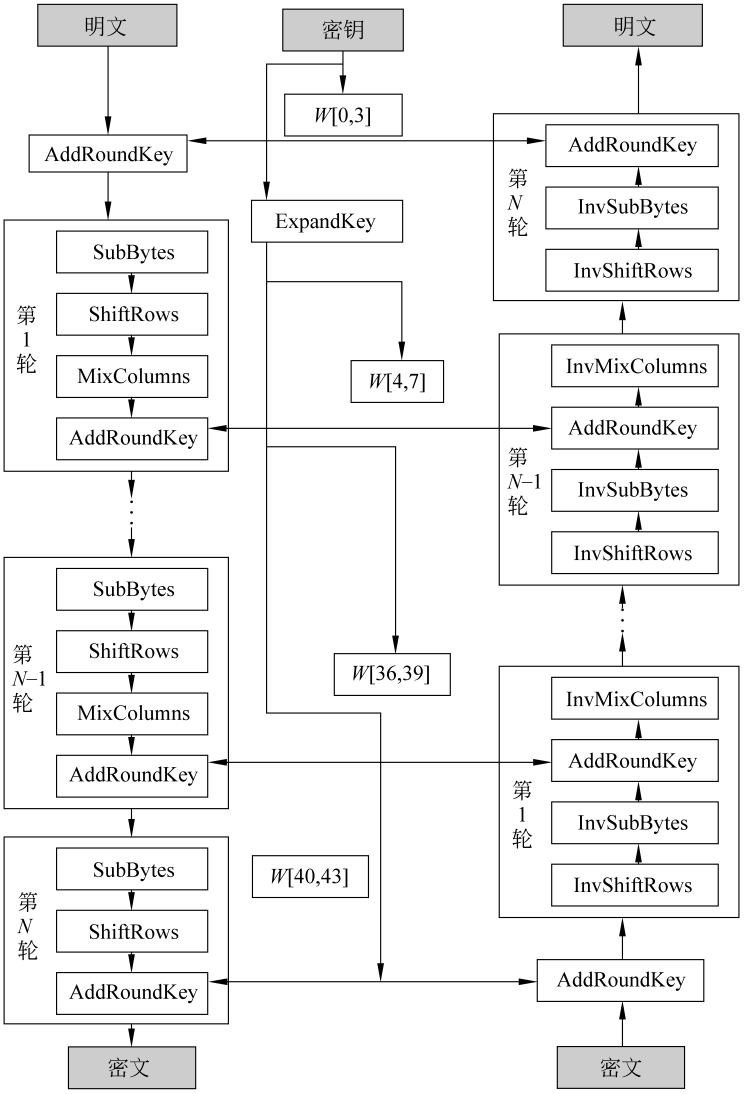

图 3.4　AES 算法流程图

3.5 哈希函数

3.5.1 哈希函数定义

密码学上的哈希函数(Hash Function),也称杂凑函数。其目的是将任意长度的消息 m 压缩或者映射为某一固定长度的消息摘要,表示为 $H(m)$,该映射是一个多对一的映射。$H(m)$也可以当作消息 m 的指纹,一旦 m 发生变化,$H(m)$也会随之改变。

根据是否需要密钥,哈希函数可分为以下两类。

(1) 带密钥的哈希函数:有两个不同的输入,分别为消息和密钥。一个带密钥的哈希函数通常用来作为消息认证码,即消息验证码(Message Authentication Code,MAC)。

(2) 不带密钥的哈希函数:只有一个消息输入,这样产生的消息摘要必须被安全存放,不能被篡改。

根据设计结构,哈希函数可以分为 3 类。

(1) 标准哈希函数。

(2) 基于分组加密的哈希函数。

(3) 基于模数运算的哈希函数等。

目前,主要的标准哈希函数可以分为两类。

(1) MD 系列的哈希函数,例如 MD4、MD5、MDS、HAVAL、RIPEMD 等。

(2) SHA 系列的哈希函数,例如 SHA-1、SHA-256、SHA-384、SHA-512 等。

3.5.2 哈希函数主要性质

哈希函数 H 必须满足以下 6 种性质。

(1) H 的输入可以是任何大小的数据块,即 H 的输入可以是任意长度的。

(2) H 的输出是固定大小的数据块。

(3) 对任意消息 x,求 $H(x)$是可行的,并可用软件或硬件实现。

(4) 对任意给定的哈希值 h,找到 x 且满足 $H(x)=h$,计算上是不可行的。

(5) 对任意给定的值 x,找到 y,满足 $H(x)=H(y)$并且 $x\neq y$,计算上是不可行的。

(6) 找到任意的 x,y 且 $x\neq y$,使得 $H(x)=H(y)$,计算上是不可行的。

基于以上性质,可以将哈希函数分为两大类:弱碰撞哈希函数和强碰撞哈希函数。弱碰撞哈希函数与强碰撞哈希函数必须同时满足前 3 个条件,主要区别在于后 3 方面。

1. 弱碰撞哈希函数

满足上面性质(4)和(5)的哈希函数,称这个哈希函数 $H(x)$是弱碰撞的。

2. 强碰撞哈希函数

满足上面性质(4)、(5)和(6)的哈希函数,称这个哈希函数 $H(x)$是强碰撞的。

在以上的性质中,性质(4)的意义是指哈希函数是单向性的,通常也称抗原像性,那么

$H(x)$也是单向杂凑函数;性质(5)代表弱碰撞性,可以称为抗第二原像性;性质(6)代表强碰撞性,也可以称为抗碰撞性,用来防止攻击者对哈希函数实施自由起始碰撞攻击,即生日攻击。

3.5.3　哈希函数安全性

哈希函数在密码系统中使用十分广泛,包括加密方案、签名方案等都有哈希函数的存在,因此哈希函数也已经成为很多黑客攻击的目标。对哈希函数的攻击主要在于破坏其某些特性。例如,可以根据其输出求得输入,找到某条消息使得它的输出与原消息的输出相同,或者找到不同的两条消息,使它们的输出相同。最具有代表性的对哈希函数的攻击为以下两种。

1. 字典攻击

字典攻击(Dictionary Attack)的定义是,在破解密码或密钥过程中,通过逐一尝试用户自定义词典中所有可能的密码(单词或短语),这种攻击方式称为字典攻击。字典攻击与暴力破解相似,区别在于,暴力破解会逐一尝试所有可能的组合密码,没有预先预定好的单词列表,而字典攻击会使用一个预先定义好的单词列表(可能的密码)进行逐一尝试。因此,字典攻击对用单向哈希函数加密的口令文件特别有效。攻击者通过编制含有多达几十万个常用口令的表,然后用单向哈希函数对所有口令进行运算,并将结果存储到文件中,攻击者通过非法途径获得加密的口令文件后,比较这两个文件,观察是否有匹配的口令密文,这种攻击方式的成功率非常高。

2. 生日攻击

生日攻击(Birthday Attack)的原理是一个统计问题。一个房间里有多少人才能够使得最少有一人与你的生日相同的概率大于 1/2 呢? 答案是 253。那么应该有多少人才能够使得他们中至少有两人的生日相同的概率大于 1/2 呢? 23 人即可。寻找特定生日的某人,或者是寻找两个随机的具有相同生日的两个人,与对单向哈希函数进行穷举攻击的方法是相同的原理:前者对应了给定消息 m 的哈希值 $H(m)$,攻击者不断生成其他消息 m',以使得 $H(m)=H(m')$;后者对应的场景是,攻击者寻找两个随机的消息 m 和 m',并使得 $H(m)=H(m')$,而后者称为生日攻击,这种攻击方式没有利用哈希函数的结构和任何代数弱性质,只依赖于消息摘要的长度,即哈希值的长度。为了抵抗生日攻击,哈希函数安全性的一个必要条件就是消息摘要必须足够长。

3.5.4　哈希函数主要应用

在密码学中,哈希函数有非常多的应用场景,包含数据认证、数字签名、伪随机数生成和密钥生成等。

1. 数据认证

哈希函数的重要应用之一就是生成文件或者消息 m 的摘要值 $H(m)$,该值被安全地

存储，对消息 m 的任意改动都可以通过对改动后的 m 进行相同的哈希运算而被检测出来，这样的操作保证了数据的完整性，也可以实现对消息的安全认证。对于哈希函数的不同应用，包括 HMAC，即基于哈希函数的消息验证码，其基本思想就是用密钥和消息一起进行哈希操作，确保消息的完整性，使得在传输过程中对消息的任何修改都能够被接收者检测到。

2. 数字签名

哈希函数经常被应用在数字签名算法中。在现实应用中，需要签名认证的消息 m 本身非常大，如果直接在消息 m 上做数字签名效率会非常低。通常情况下，为实现对消息 m 的高效数字签名，首先计算出 m 的哈希值 $H(m)$，然后采用私钥对 $H(m)$ 进行签名操作，所得到的 $\mathrm{Sig}(H(m))$ 就是用户对消息 m 的签名。利用哈希函数进行数字签名的优势是，可以实现对于消息的不可否认性，消息发布者只能用自己的私钥对哈希值进行签名，接收者使用发布者的公钥进行解密，从而确认消息确实由该消息发布者发出。

3. 伪随机数生成

从哈希函数的定义来看，哈希函数具有生成随机性质的数据序列的特征。通过选择一个随机函数，把消息的随机函数值作为它的哈希值来产生，从而得到伪随机数。因此，哈希函数也可以用作伪随机数的生成器。

4. 密钥生成

利用哈希函数的单向性，用旧的密钥计算出新的密钥序列，从而使得现有密钥具有泄露后不危及先前所用的密钥的性质，也就是使用哈希函数能够产生具有前向安全的密钥。

总体而言，哈希函数在密码学的应用中一直发挥着重要的作用，在区块链中也有着十分广泛的应用，如区块链中的链式结构就是通过让每一个区块包含上一个区块的区块头哈希值来实现的。此外，区块链中的默克尔树构造、工作量证明算法、钱包地址等都用到了哈希算法。

3.6 默克尔树

1. 基本概念

默克尔树(Merkle Tree)，也称 Hash Tree，是用于存储哈希值的二叉树。默克尔树包含叶节点和非叶节点，其中叶节点用于存放数据块(例如，文件或者文件的集合)的哈希值，所有非叶节点是其对应所有子节点的组合结果的哈希值。

图 3.5 为一个默克尔树的结构。在树的最底层，也就是叶节点，类似于哈希列表，需要计算的数据被分割成不同的数据块，将数据块的哈希值存放于叶节点。接着计算时并不是直接计算叶节点的哈希值，而是把相邻的两个叶节点的哈希值合并成一个字符串，然后运算这个合并字符串的哈希值。例如，图 3.5 中 $H_6 = H(H_2, H_3) = H(H(L_1),$

$H(L_2)$),从而得到一个“子哈希值”。如果最底层的叶节点总数不是双数,那么必然出现一个无法配对的单身哈希数据块,这种情况就直接对它进行哈希运算,也能得到它的子哈希值。以此类推,就可以得到数目更少的新一层的哈希值,最终必然形成一个哈希树结构,直到树根的位置,就只剩一个根哈希值,称为默克尔树根节点值(Merkle Tree Root)。

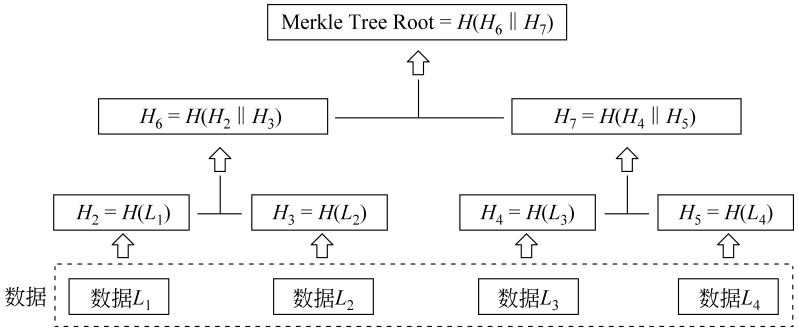

图 3.5　默克尔树示意图

2. 主要特点

默克尔树相比其他数据结构,具有一些特殊的性质。

(1) 默克尔树是一种树状结构,大多数是二叉树,也有可能是多叉树,但无论是几叉树,它都具有树状结构的所有特点。

(2) 默克尔树的所有叶节点的值是数据集合的单元数据的哈希值。

(3) 默克尔树非叶节点的值是根据它左右子节点的值,按照特定的哈希算法计算得出。

一般而言,通常采用 SHA-2 和 MD5 作为加密的哈希算法。但如果仅仅防止数据不是蓄意的损坏或篡改,可以改用一些安全性较低但效率较高的校验和算法,如循环冗余校验(Cyclic Redundancy Check,CRC)。

在区块链中,假设一个默克尔树包含 N 笔交易数据,当需要验证一笔指定的交易是否存在于某一区块中时,参与者至多花费 $2\log_2(N)$次的计算就可以得到结果。这主要是由 Merkle 树的构建方式来决定的,它的叶节点值由交易数据的哈希计算得到,并自底向上计算哈希值构建更新上一层节点的数值,直到计算得到默克尔树根节点值,并验证其是否发生变化。假设区块中的交易数量为奇数,则可以重复某笔交易来获得偶数笔交易,避免叶节点无法配对的现象出现,并实现默克尔树的构建。

3.7　布隆过滤器

1. 基本概念

20 世纪 70 年代,Burton Howard Bloom 教授提出了布隆过滤器(Bloom Filter)的概念,它的核心包含一系列哈希映射函数和一个很长的二进制向量,用于快速检索判断一个

元素是否存在于某个集合中。布隆过滤器的优点在于空间效率和查询效率非常高，远远超过一般的算法，因为它在进行查询时，采用位数组表示一个集合，这样的算法结构大大节省了存储空间，提高了效率，但它也有不足之处：删除困难，以及存在一定概率的误识别，也就是假正例(False Positives)的现象，即当布隆过滤器经过运算向系统报告某一元素在集合中时，事实上它判断错误，该元素并不在集合中，布隆过滤器将一个不属于集合的元素误检测为集合中的元素。相对于假正例现象，布隆过滤器不会发生假反例(False Negatives)，也就是如果一个元素不在查询集合中，那么布隆过滤器是不会向系统报告该元素属于集合。换句话说，布隆过滤器判断一个元素不在集合中，就一定不在，但当它判断一个元素在集合中，就存在一定的判断误差，可能导致结果不准确。布隆过滤器的优缺点反映了它是用较小的错误率来换取较高的运算效率，适用能容忍容错率的应用场景，而无法适用“零错误”的场景。

布隆过滤器的常用的应用场景包含：判别某个元素是否存在于一个集合中，为实现这个目标，通常的做法是采用哈希表(Hash Table)数据结构，它可以通过一个哈希函数将一个元素映射成一个位阵列中的某个点，并且这些位阵列的初始值都为 0，当有元素映射到位阵列中的某一点时，就将这一点的值设置为 1。因此，要知道某个集合中是否有该元素，只需要判断这个点在哈希表中对应的位置是否为 1，就可以知道集合中有没有该元素。这就是布隆过滤器的基本思想，算法的具体过程如图 3.6 所示。将所有元素通过布隆过滤器保存至集合中，然后通过比较确定该元素是否存在于该集合。链表、树等结构也采用同样的方式进行判断。

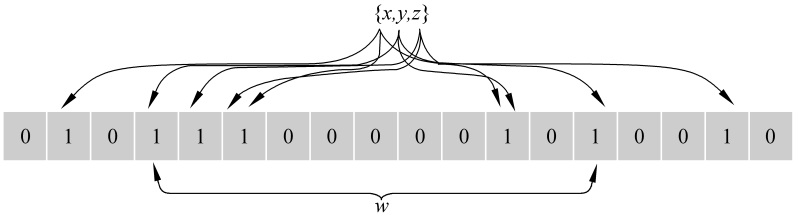

图 3.6　布隆过滤器

但采用这样的方式进行数据存储和查询，存在着一些难以避免的问题：首先，随着集合中元素的增加，哈希表所需要的存储空间越来越大，检索速度也会变得越来越慢。其次，这样的结构面临的另一个问题就是冲突，假设哈希函数是合适并且良好的，如果位阵列长度为 m 个点，那么若想将冲突率降低到 1%，这个哈希表就只能容纳 $m/100$ 个元素，显然就无法实现所有空间有效的要求；为解决无法实现所有空间有效的问题，解决方法是可以使用多个哈希函数，数据可以采用多个哈希函数进行映射存储，便可以实现空间有效，正确判断某个元素是否在集合中。

2. 布隆过滤器的算法过程

(1) 选取 k 个哈希函数，每个函数可以把目标数据 d 哈希为 1 个整数。

(2) 初始化一个长度为 n 位的数组，每位初始化为 0。

(3) 在某个数据 d 加入集合时，用 k 个哈希函数计算出 k 个哈希值，并把数组中对应的位置为 1。

(4) 在判断数据 d 是否存在于该集合时，用 k 个哈希函数计算出 k 个哈希值，并查询数组中对应的位，如果所有的位都是 1，则认为该 key 在集合中；反之，则不存在。

3. 布隆过滤器的主要特点

布隆过滤器的优势有以下 5 点：①空间上的低存储量和时间上的高效性，布隆过滤器的存储空间、插入和查询时间都是常数，相比于其他的数据结构，布隆过滤器有着天然具备的高效性；②布隆过滤器中采用的哈希函数相互之间没有关系，具有硬件并行实现的编辑性；③布隆过滤器不需要存储元素本身，因此对于某些保密要求非常严格的数据和场合，在安全性上相比其他数据结构具有绝对优势；④布隆过滤器可以用于表示全集数据，实现数据的完备性；⑤布隆过滤器还具备的特点是，当哈希函数的数量 k 和阵列数量 m 相同时，使用同一组哈希函数的两个布隆过滤器的交并差运算可以使用位操作进行。

布隆过滤器也存在一定劣势：①随着存入的元素数量增加，数据查询的误算率也会随之增加。②从布隆过滤器中安全地删除元素信息较为困难，通常采用的方法是把位列阵变成整数数组，对插入元素进行计数，每插入一个元素相应的计数器加 1，这样删除元素时将计数器减 1。然而，这样的操作可能导致信息的泄露，因为首先必须保证删除的元素的确在布隆过滤器里面，而这一点单凭过滤器是无法保证的。

3.8　公钥密码体制

3.8.1　基本概念

20 世纪 70 年代，Whitfield Diffie 和 Martin Hellman 在 *New Directions in Cryptography* 中提出了公钥密码体制，又称非对称密码体制或公开密钥密码系统的概念，是现代密码学蓬勃发展的关键一步，开创了一个全新的密码学时代。他们在这篇划时代的文章中做出了如下设想：假设系统中有两个用户 Alice 和 Bob，Alice 拥有一对公钥 k_p（加密密钥）和私钥 k_s（解密密钥），其中公钥可以在全网公开，私钥由 Alice 自己保存。当 Bob 想要发送一条消息给 Alice，同时他又不希望别人看到，Bob 首先在系统中找到 Alice 的公钥 k_p，然后用 k_p 对消息进行加密，再将加密后的密文传输给 Alice。Alice 收到密文后，使用自己保存的私钥 k_s 对密文解密，就可以获得明文消息。如果存在攻击者截获了 Bob 发送给 Alice 的密文消息，但由于它不知道 Alice 的私钥，也是没有办法解密密文、读取明文的。在这个过程中，加密密钥是公开的，因此它和解密密钥必须不同，同时在已知加密密钥的前提下，不能反推出解密密钥，从而保证密码体制的安全性。在对称密码体制中，系统的通信双方需要提前通过安全信道交换彼此的密钥，但这一条件非常苛刻，现实生活中很难找到绝对安全的信道，公钥密码体制的思想就很好地解决了这个问题，至此开启了现代密码学的新篇章，相继出现了诸如 RSA、ECC 和 ElGamal 等公钥密码方案。

如图 3.7 所示，数据发送者和数据接收者的通信使用公钥密码体制，明文 m 用加密

密钥 k_p 来加密，数据接收者则采用与加密密钥不同的解密密钥 k_s 来解密。

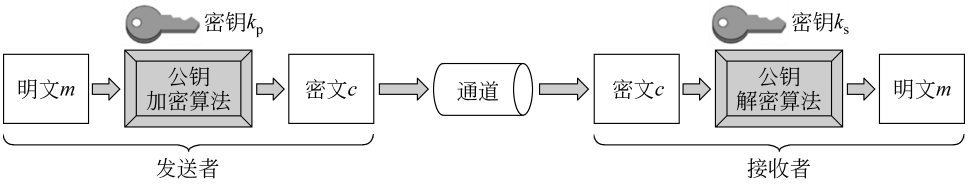

图 3.7 公钥密码体制

若将公钥密码体制表示成一个五元组$\{M, C, K, E_K, D_K\}$，则其数学公式满足以下条件。

(1) M 是可能的消息集合。

(2) C 是可能的密文集合。

(3) 密钥空间 K 是一个可能的密钥有限集。

(4) 对于密钥空间 K 中的每一个元素 $K=\{K_1, K_2\}$，都有与之对应的加密算法 E_{K_1} 和解密算法 D_{K_2}，使得对于任意明文 $m \in M$ 都满足 $c=E_{K_1}(m)$，$c \in C$，并且有 $m=D_{K_2}(c)$；其中 E_{K_1} 是公开函数，K_1 表示公钥，而 D_{K_2} 是保密函数，K_2 表示私钥，用户秘密保存。公钥密码体制的核心问题便是如何对 E_{K_1} 和 D_{K_2} 进行描述。

3.8.2 ElGamal 公钥加密

1984 年，斯坦福大学的 Tather ElGamal 提出 ElGamal 公钥密码体制，这种密码体制不仅适用于加密算法，而且也能用于数字签名算法。Tather ElGamal 在提出 ElGamal 公钥密码体制之后，于 1985 年设计出 ElGamal 数字签名方案，美国著名的数字签名算法(DSA)就是 ElGamal 数字签名算法的演化。基于有限域上离散对数问题难以解决的特性，ElGamal 公钥密码体制的安全性得以保障。在 ElGamal 加密过程中，算法都会选择一个随机数参与运算，产生密文，因此对于同一个消息，每次加密后得到的密文都是不同的，攻击者不仅不能从密文中获得任何明文信息，也不能通过观测密文来判断加密的是否是同一个明文。此外，加密算法产生的密文形式包含两部分，一个可以看作是 Diffie Hellman 的密钥交换，另一个是对明文的隐藏，所以密文的长度是明文长度的两倍。下面是 ElGamal 公钥加密算法的详细描述。

(1) 密钥产生：对一个大素数，有限域 GF(p)上的离散对数问题是难以解决的。首先任选一个大素数 p 和 GF(p)上的生成元 α。将参数 p 和 α 公布出去，选择一个随机整数 d 作为私钥，$2 \leqslant d \leqslant p-2$。计算 $y=\alpha^d \bmod p$，选取 y 为公钥。

(2) ElGamal 加密：对明文 M 加密，随机地选取一个整数 k，$2 \leqslant k \leqslant p-2$，使用式(3.15)计算

$$\begin{aligned} C_1 &= \alpha^k \bmod p \\ C_2 &= M \cdot y^k \bmod p \end{aligned} \tag{3.15}$$

最后，得到密文为 $C=(C_1, C_2)$。

(3) ElGamal 解密：计算 $M=C_2/C_1^d \bmod p$，可以由密文得明文 M。