计算机安全学复习小记

根据《密码编码学与网络安全——原理与实践》整理

整理者：0.H.P

1 基本概念

1.1 三大目标 (C.I.A)

(1) 机密性/Comfidentiality(防止数据泄露)

- 数据机密：信息不能被非授权者泄露
- 隐私性：确保个人信息哪些可以被怎样操作，向哪些人公开

(2) 完整性/Integrity(防止数据被篡改)

- 数据完整：信息、程序能被特定授权改动
- 系统完整：系统以预定功能执行，避免被非授权者操作

(3) 可用性 Availability(确保资源可用)

- 确保系统能够工作，不能拒绝授权者

1.2 两个概念

(1) 认证性/Authenticity：一个实体具备可被验证可被信任的特征；于信息接收方而言：接受的信息及来源是正确的。

(2) 可追溯性/Accountability：实体的行为可以被唯一追踪。

1.3 OSI 安全框架

目的：评价一个机构的安全需求，对不同安全产品进行选择评价，管理员制定某种系统方法以定义需求和满足的措施，且有以下概念。

(1) 安全攻击：危害信息系统安全的行为

- 被动攻击：对传输进行窃听和检测
- 主动攻击：对数据流进行篡改或伪造数据流

(2) 安全服务：加强数据处理系统和传输安全性服务，有：认证 (包括对等实体认证，如 TCP 连接传输；数据源认证，如 UDP 传输)、访问控制、数据保密、数据完整、不可否认 (防止发送或接收方否认自己的通信行为)、可用性。其目的是用安全机制进行反攻击。

(3) 安全机制：检测、阻止攻击；恢复被攻击的系统为正常状态的过程或实现设备。(机制提供服务)

1.4 攻击 surfaces 与攻击树

攻击 surfaces 实例：
开放端口、接口、服务、工程师、雇员等.
分类：软件、网络、人
攻击树：
根：攻击目标
分支结点：方法，或者子目标
叶子结点：攻击初始化

1.5 网络安全模型

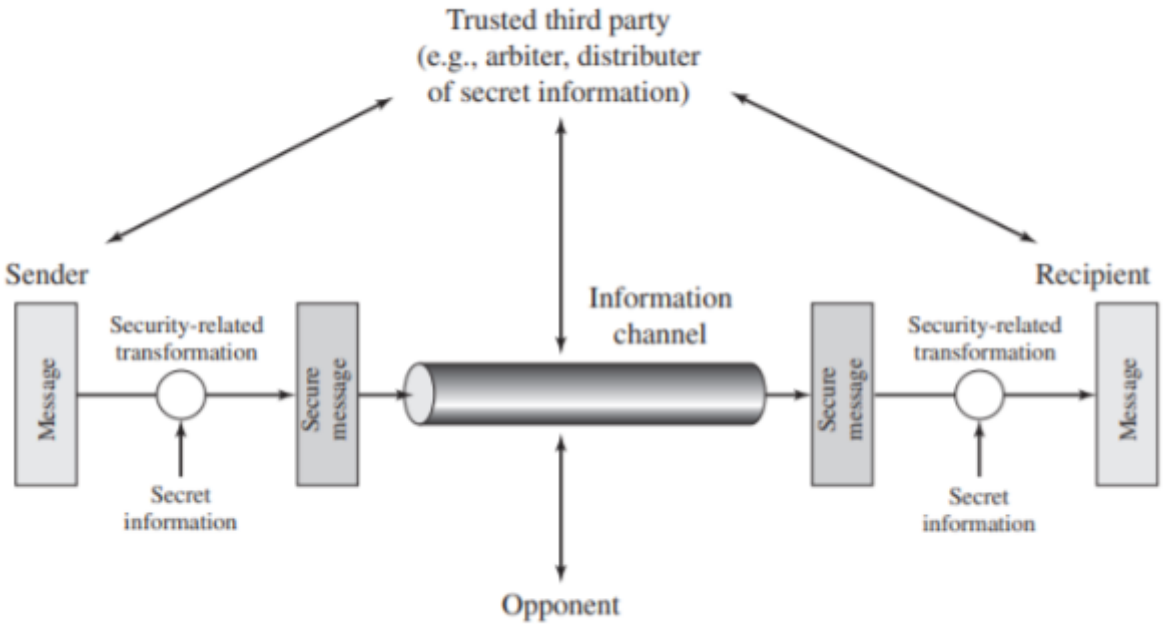

Figure 1.2 Model for Network Security

图 1: 网络安全模型图示

保证安全的方法包括以下两点

(1) 被发送的信息的相关交换；如将信息加密

(2) 发送与接收双方共享秘密信息；如密钥，加密算法等

(3) 可靠的第三方，用于分配秘密信息，仲裁等。可选。

设计安全服务包含以下四点

(1) 算法：执行安全传输的相关**算法**

(2) 算法产生的**秘密信息**

(3) 秘密信息的**共享方法**

(4) 指明协议，利用算法以及秘密信息实现**安全服务**

1.6 密码学基本概念

两个定义:
1. 密码编码学
2. 密码分析学
方法：(攻击强度依次递增)

- 唯密文攻击：用已知密文恢复出明文或者密钥
- 已知明文攻击：从已知密文和部分明文 -密文对中分析明文
- 选择明文攻击：从任意明文 -密文对中攻击
- 选择密文攻击：从不同的密文，恢复对应解密的明文 (用于公钥体制)
- 穷举法

1.6.1 密码系统

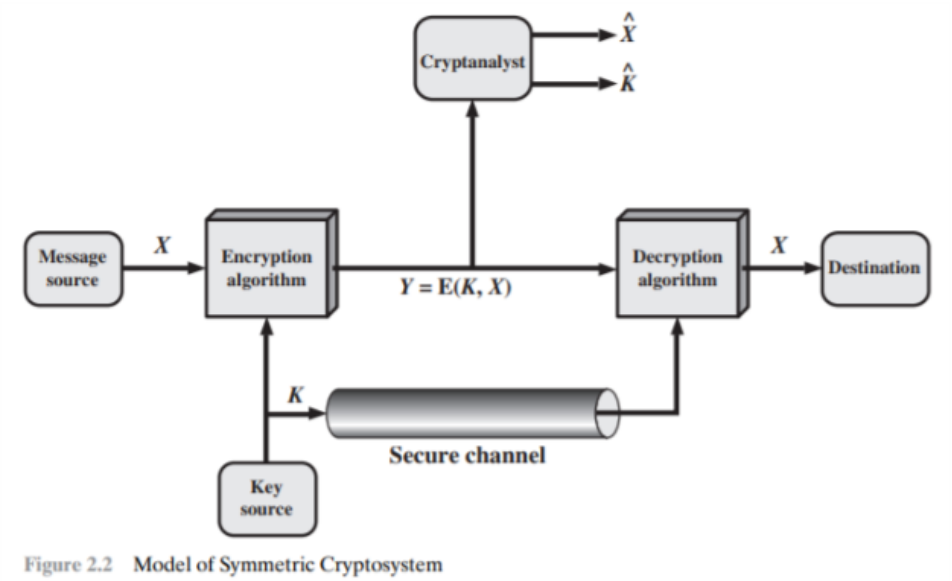

Figure 2.2 Model of Symmetric Cryptosystem

图 2: 密码系统模型图示 (香农模型)

密码系统公式表示

- 加密：E(M,K)=C
- 解密：D(C,K)=M
- 且有：D(E(M,K),K)=M

秘钥空间：秘钥 K 的取值范围，且所有算法的安全性都基于密钥安全性，而非算法细节。

1.6.2 密码体制分类

(1) 单钥体制 (对称算法)：发送方与接收方的密钥相同

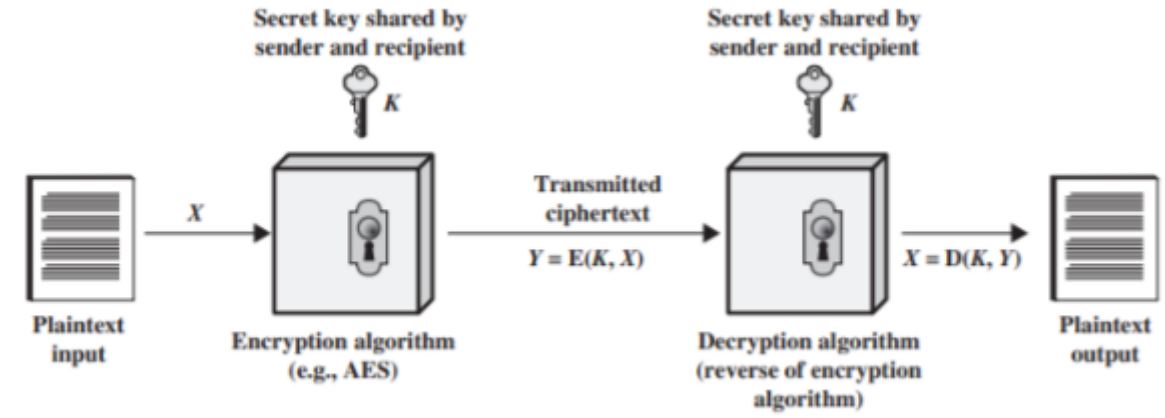

Figure 2.1 Simplified Model of Symmetric Encryption

图 3: 单钥体制图示

加密方式：流密码 (明文按位加密)；对称密码 (明文按分组加密)

(2) 双钥体制 (公钥算法)：发送方与接收方的密钥不同

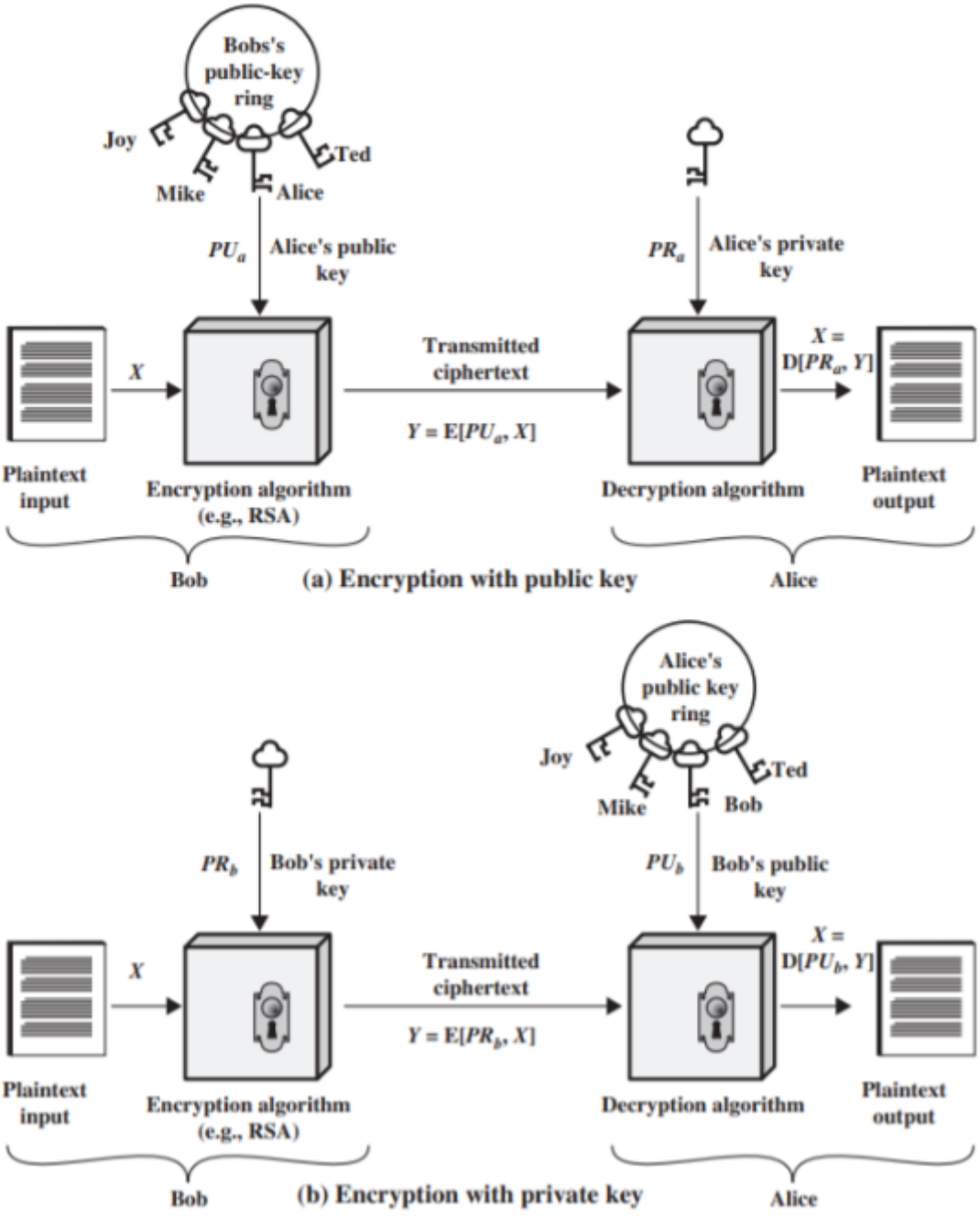

Figure 9.1 Public-Key Cryptography

图 4: 公钥体制图示

用于加密：

加密：E(M,public key)=C

解密：D(C,private key)=M

且有：D(E(M,public key),private key)=M

用于认证：M=E(D(M,private key),public key)

1.6.3 古典密码

两大构造模块：
1. 代换/substitution：明文被其他字符代替 $\rightarrow$ 混淆
2. 置换/Permutation：明文序列改变 $\rightarrow$ 扩散
3. 乘积密码 (为现代密码学打下基础)：迭代使用代换和置换构造算法，即是 $S \times P$ 的网络，简称 SPN

2 数论

2.1 数论基础

屠龙宝刀，号令天下：`https://www.jianshu.com/p/34107ca9b4ee`

2.2 伽罗瓦域

(1) 元素集合：$F = \{0,1\}^8$，即 $[0, 256)$

(2) 元素表示形式：

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + ... + a_1x + a_0 = \sum_{i=0}^{n-1} a_i x^i$$

其中 $a = 0, 1$

(3) 加法操作：异或运算 (XOR), 且 $(F, +)$ 成群

(4) 减法操作：跟加法相同

(5) 乘法操作：位移和异或，且 $(F^*, \times)$ 成群

- $A \times 2$：
  A 左移一位：当最高位为 0 时，结束；当最高位为 1 时，结果异或 283(十进制) ，其中：283 表示为 $x^8 + x^4 + x^3 + x + 1$，称不可约多项式。也可以用其他，不一定是 283.
- $A \times B(B \neq 2)$：
  将 B 分解为若干个 2 的组合

(6) 除法：多项式除法，无借位操作。(跟 CRC 校验码的计算相同)

代码：

```
#multilpy in GF(2^8)
def mul(a,b):
    r=0
    while b:
        if b%2:
            r=r^a #add operation : XOR
        b=b>>1
        if a&int('10000000',2)==0: #first bit's value = 0
            a=a<<1
        else: #first bit's value = 1
            a=a<<1
            a=a^283
    return r
#compute the max index number which < 2^count
#return count, from 0
def highest_bit(n):
    count = 0
    while n:
        count+=1
        n=n>>1
    return count-1
#division about polymerization
#return quotient and remainder
def div(a,b):
    if a==b:
        return 1,0
    if a<b:
        return 0,a
    a_bit = highest_bit(a)
    b_bit = highest_bit(b)
    result = 0
    while not a_bit<b_bit:
        move=a_bit-b_bit
        temp=b<<move
        result=result+(1<<move)
```

```
36            a=a^temp
37            a_bit=highest_bit(a)
38        return result,a
```

3 对称加密

3.1 AES 算法

AES 算法：一种采用对称密钥对数据分组进行加密和解密的算法。
参数：数据分组：每组 128bits; 密钥：128、192、256bits。
基本结构如下 (以 128bits 密钥为例)：

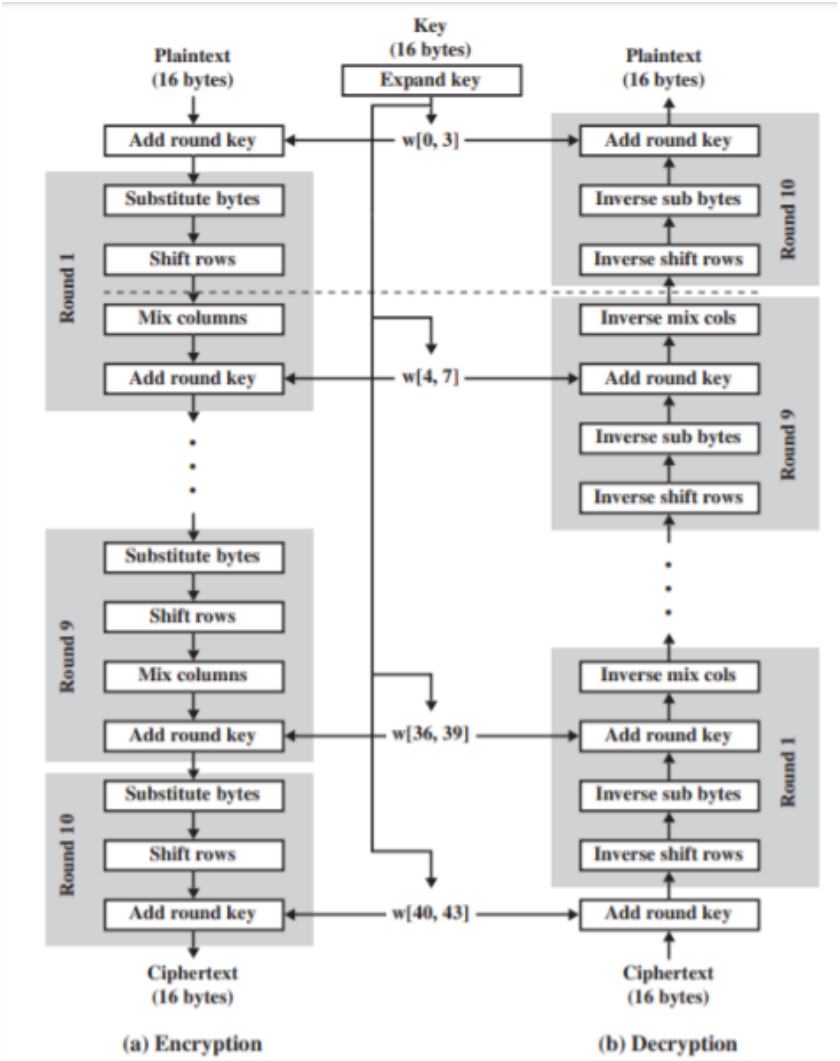

Figure 5.3 AES Encryption and Decryption

图 5: AES 图示

简单描述：便是数据分组首先经过与密钥计算，后经过十轮加密，每一轮的密钥也都不同，最后输出数据。每一轮的加密经过以下几个过程：(第十轮只执行前三步)

(1) 字节代换 (SubBytes): 用一个 S 盒代换每一字节的值。

(2) 行位移 (ShiftRows): 置换过程

(3) 列混淆 (MixColums): 利用 $GF(2^8)$ 的算数特性进行代换

(4) 轮密钥加 (AddRoundKey): 分组与密钥 XOR 运算

3.1.1 字节代换 (SubBytes)

该步骤，用一个 S 盒 (16*16 的矩阵), 将分组的每一个字节进行替换。替换时，将每字节的值分为高四位与低四位，然后对应于 S 盒的索引进行替换。(解密时，用逆 S 盒即可)。例如：输入 95，返回 2A。

Table 5.2 AES S-Boxes

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(a) S-box

图 6: S 盒图示

S 盒的构造方法：

(1) 按字节升序初始化 S 盒。第一行是 00,01...0F... 第十六行是 F0...FF

(2) 将每个字节求在 $GF(2^8)$ 中的逆元。

(3) 每一字节进行如下运算

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{1}$$

3.1.2 行位移 (ShiftRows)

将 16 个字节编排成 4*4 矩阵, 第一行按字节单位左移 0 位；第二行左移 1 字节；第三行左移 2 字节；第四行左移 3 字节。例如：

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

图 7: 行位移图示

3.1.3 列混淆 (MixColums)

将上述计算后的矩阵再乘以一下矩阵。即按一下计算方式计算

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

图 8: 列混淆图示

3.1.4 轮密钥加 (AddRoundKey)

将 128bits 数据与每一轮的密钥异或运算，得到新的数据值。
以下是扩展每一轮的密钥算法。

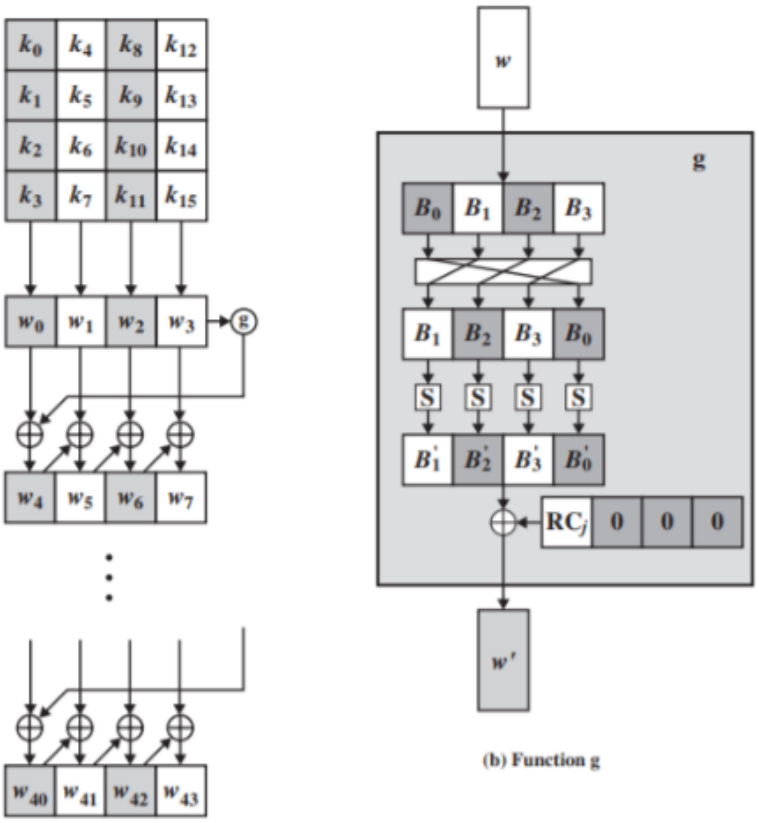

Figure 5.9 AES Key Expansion

图 9: 扩展密钥图示

此图中 $k_0 - k_{15}$ 是初始密钥 128bits，按以下三个步骤进行拓展，最后输出 44 个 32bits 的拓展密钥，每一轮用 128bits，供十轮之用。$w_0 - w_3$ 的构成是将 $k_0 - k_{15}$ 矩阵按列拼接而得，k_i 是 8bits 数据，拼接后 w_i 是 32bits 数据。构造扩展密钥时，主要用以下 g 函数进行扩展，g 函数包括以下几步

(1) 将每一个字中的四个字节左移一个字节

(2) 用 S 盒进行字节代换

(3) 用轮常量异或运算

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

图 10: 扩展密钥轮常量图示

3.2 工作模式

(1) 电码本模式/ECB：直接将分组与同一密钥加密

(2) 密文分组连接模式/CBC：将分组与上一分组异或后再加密，K 仍然相同

(3) 密文反馈模式/CFB：将明文分为每组 s 位。首先用一初始值，用 K 加密后，得到 b 位数据，然后取前面 s 位，与明文异或得到密文，且作为下一分组的初值传入。传入后，将 s 位的数据左移 b-s 位得到 b 位数据参数。以此循环。每次 K 相同

(4) 输出反馈模式/OFB：用一个初值与 K 加密，然后与分组异或，并且作为下一分组加密值的参数传入

(5) 计数器模式/CTR：用一个随机数，与密钥加密再与分组异或，然后将随机值 +1 用于下一分组的计算

4 非对称加密

4.1 RSA

算法步骤

(1) 选择两个素数 p 和 q

(2) 计算 n=p*q

(3) 计算 $\phi(n) = \phi(p) * \phi(q)$

(4) 选择 e 与 $\phi(n)$ 互素且小于 $\phi(n)$

(5) 计算 e 模 $\phi(n)$ 的乘法逆元 d

(6) 得到公钥 (e,n), 私钥 (d,n)

加密时：$C = M^e \ mod \ n$
解密时：$M = C^d \ mod \ n$
攻击时：分解 n; 直接确定 $\phi(n)$; 直接确定 d 三种思路

4.2 Diffie-Hellmam 密钥交换

场景：用户 A 和 B 要交换密钥
算法步骤：

(1) 选定公开参数：一个素数 p 及其本原根 α

(2) 用户 A 选择临时密钥 X_A，计算公开值 $Y_A = \alpha^{X_A} \ mod \ p$

(3) 用户 B 选择临时密钥 X_A，计算公开值 $Y_B = \alpha^{X_B} \ mod \ p$

(4) 接着用户 A 计算密钥 $K = Y_B^{X_A} \ mod \ p$

(5) 用户 B 计算密钥 $K = Y_A^{X_B} \ mod \ p$

密钥交换协议

该协议利用 Diffie-Hellman 算法，当然，在通信前通信双方应知模数 p 以及其本原根, 方法之一是用户 A 选择这两个参数，并将其放在首次通信的信息中，大致算法如图。

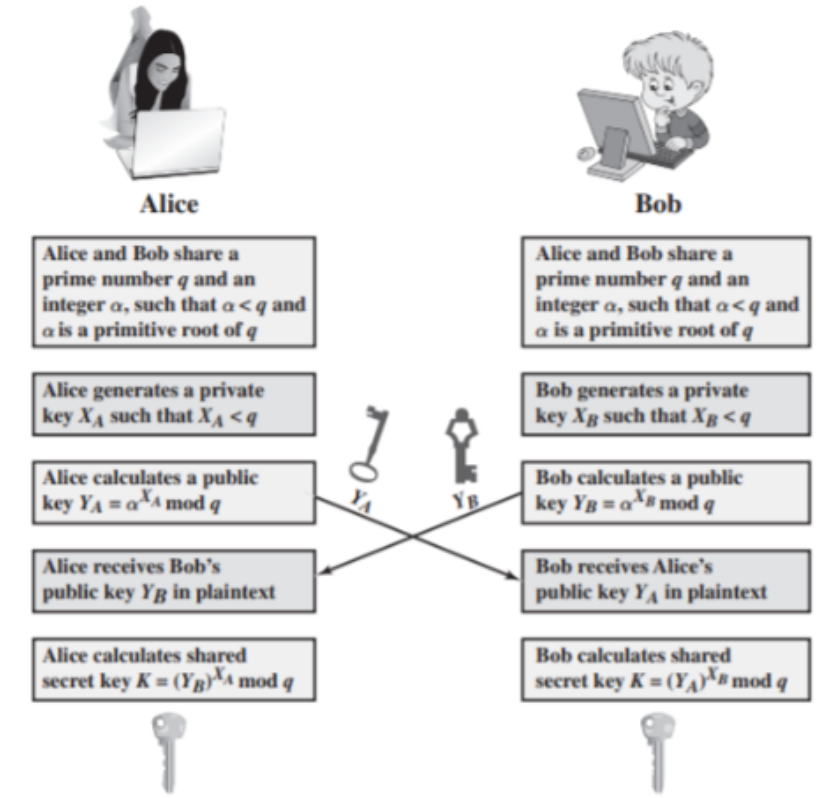

Figure 10.1 The Diffie-Hellman Key Exchange

图 11: 密钥交换协议

中间人攻击

中间人攻击，就是在通信双方 A 和 B 中间截获信息的攻击, 过程如下

(1) 攻击方 C 生成自己的 X_{CA} 和 X_{CB}

(2) A 将 Y_A 发给 B，C 截获，给 B 发送 Y_{CA}，自己计算 $K_B = Y_A^{X_{CB}} \ mod \ p$

(3) B 接收 Y_{CA}，计算 $K = Y_{CA}^{X_B} \ mod \ p$, 发给 A 是 Y_B

(4) C 截获，计算 $K_A = Y_B^{X_{CA}} \ mod \ p$, 并计算 Y_{CA} 发给 A

4.3 ElGamal 密码体系

与 DH 类似，利用离散对数的加密机制，算法步骤如下

(1) 选定素数 p 以及其本原根 α

(2) 用户 A 有私钥 X_A, 以 p 和 α, 生成公钥 (α,p,Y_A), 其中 $Y_A = \alpha^{X_a} \ mod \ p$

(3) 用户 B 利用以上参数对信息加密，选定随机数 k，计算
$K = Y_A^k \ mod \ p, C_1 = \alpha^k \ mod \ p, C_2 = MK \ mod \ p$ 并传输密文对 (C_1, C_2)

(4) 用户利用 A 得到密文对，恢复成明文：$K = C_1^{X_A} \ mod \ p, M = C_2 K^{-1} \ mod \ p$

4.4 椭圆曲线密码学/ECC

4.4.1 基本概念

方程：

$$y^3 = x^2 + ax + b$$

为使该方程有意义，需要判别式：$2^2a^3 + 3^3b^2 = 4a^3 + 27b^2 \neq 0$.
以此为基础定义一个群, 代数集为 $E(a, b)$(所有元素皆为**点集**)，操作为加法。
该判别式保证了任意直线与该椭圆相交，都有三个根 (包含三个不同的根和二重根的情况)；除此之外，椭圆曲线还定义了一个无穷远点/零点 $O(x, 0)$(此点为单位元)

4.4.2 群定义

单位元：O; 有 $-O = O$,P+O=P
逆元：定义 $P(x, y)$, 那么 P 的逆元为 $-P(x, -y)$. 有 $P - P = O(x, 0)$
加法的几何描述：
$R = P + Q$ 表示为有一条直线相交于该椭圆曲线，有椭圆曲线最高次数是 x^3 以及判别式得，必有三个交点；且只有三个不同交点或有二重根两种情况。
该直线交过 P 和 Q，那么必然与椭圆曲线有第三个交点，此交点便是 -R.

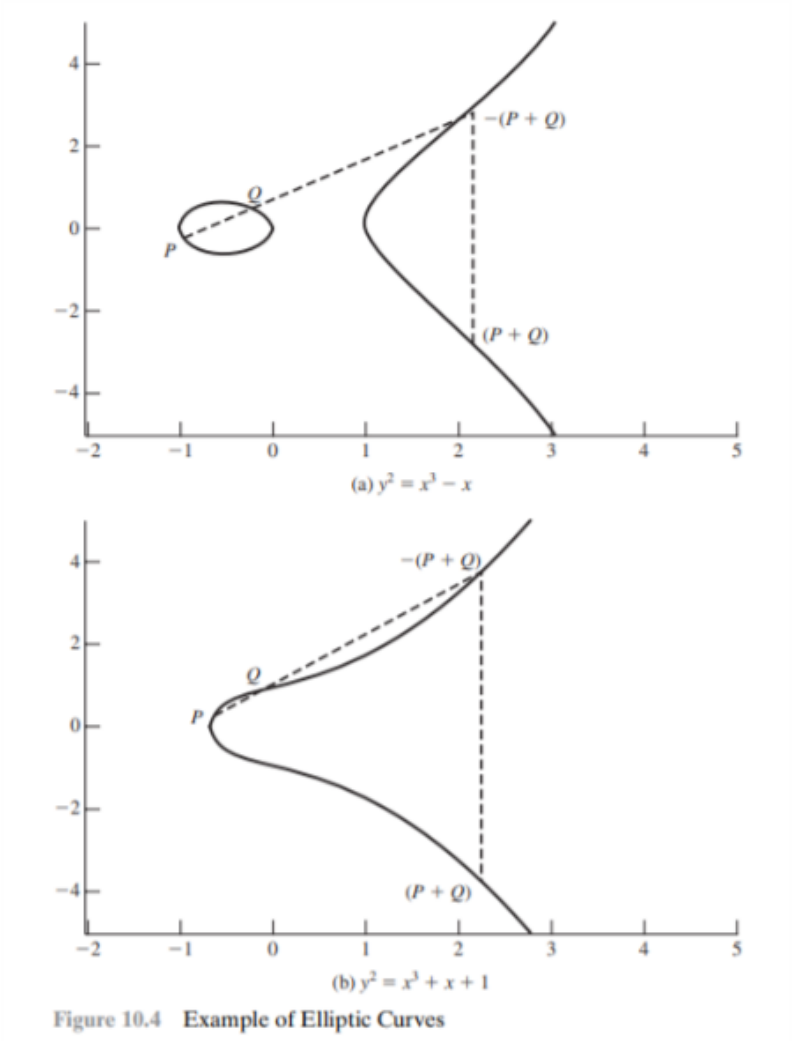

Figure 10.4 Example of Elliptic Curves

图 12: ECC 加法几何描述

4.4.3 Z_p 上的椭圆曲线

p 是素数，群操作是模 p 加法，该群表示为 $E_p(a, b)$;
加法操作：假若计算 $R = P + Q$, 公式如下

$$x_R = \Delta^2 - x_P - x_Q \ mod \ p$$

$$y_R = \Delta(x_P - x_Q) - y_P \ mod \ p$$

Δ 是直线斜率

- 假若 P=Q;
  此时 R=P+P=2P, 即直线与 P 点相切，有二重根，用偏导数求斜率

  $$y^2 = x^3 + x + 1$$

  求偏导

  $$2ydy = 3x^2dx + dx$$

  $$\frac{dy}{dx} = \frac{3x^2 + 1}{2y}$$

  得

  $$\Delta = \frac{3x_P^2 + 1}{2y_P} \ mod \ p$$

- 假若 $P \neq Q$

  $$\frac{dy}{dx} = \frac{y_P - y_Q}{x_P - x_Q}$$

  得

  $$\Delta = \frac{y_P - y_Q}{x_P - x_Q} \ mod \ p$$

乘法操作：乘法定义为重复加法

4.4.4 ECCDH

思路与 Diffie-Hellmam 相同；只是换成在 $E_p(a, b)$ 上的操作
用户 A 产生一个私钥 X_A, 并计算公钥 $x_A * G$，G 是群上的一点 (类似于 DH 中的 g^{x_A})
传给用户 B，用户 B 产生私钥 X_B, 计算公钥 $x_B * G$(类似于 DH 中的 g^{x_B}) 传给 A，自己计算 $x_B * x_A * G$ 作为密钥
A 收到 B 传来的 $x_B * G$, 计算 $x_A * x_B * G$ 作为密钥。

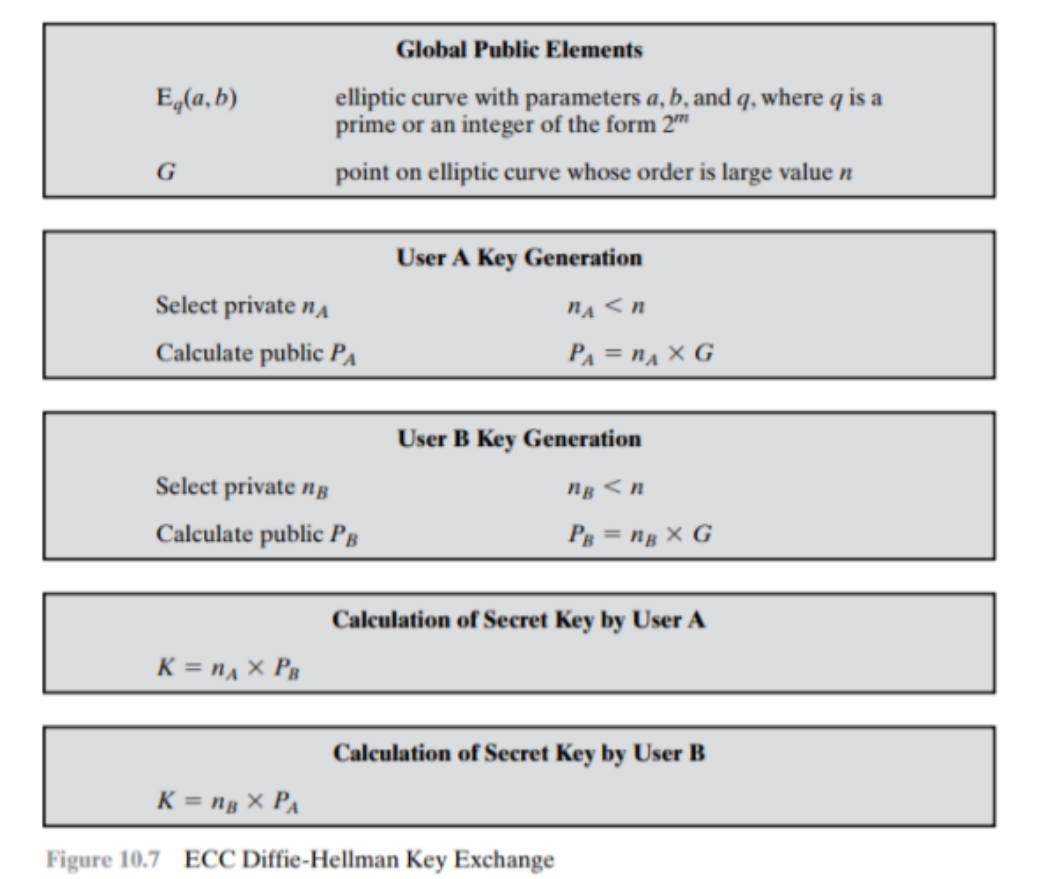

Figure 10.7 ECC Diffie-Hellman Key Exchange

图 13: ECCDH

5 HASH 函数

5.1 基本概念

HASH 函数：输入变长数据，输出固定位数数据的一种算法。使用压缩函数进行迭代，压缩函数分为两类：HASH 专用以及对称分组密码算法，其目的是产生数字指纹。

碰撞攻击：不同的信息输入，产生相同的 HASH 值; 即消息 $A \neq B, H(A) = H(B)$

单向性：用 A 计算 HASH(A) 可行，用 HASH(A) 计算 A 计算上不可行

弱抗碰撞性：寻求任意 $A' \neq A$, 使得 $H(A') = H(A)$ 计算上不可行。

强抗碰撞性：寻求任意 (A, A') 对，使得 $H(A) = H(A')$ 计算上不可行。

理解强与弱：在此强与弱是指要求，即弱抗碰撞是指希望不容易发生的事情不发生，所以实现此是容易的，称为弱抗碰撞；而强抗碰撞是指希望容易发生的事情不发生，所以实现此是困难的，是强需求的，称为强抗碰撞。

应用：

消息认证：使用带密钥的 Hash 函数 (消息认证码/MAC) 实现，MAC 函数将密钥

与消息作为数据块输入，然后产生哈希值附于消息后面，传送给另一方；接收方利用共享的密钥进行相同操作，验证生成的哈希值与传来的是否一致。是则代表消息是完整的。
数字签名：基于非对称密码学，需要被认证的一方用自己的私钥对信息的哈希值加密，认证的一方用公钥解密后得到哈希值，与消息生成的哈希值对比，看是否相等，若是，则证明该签名有效。

5.2 SHA-3

倚天不出，谁与争锋：`https://www.jianshu.com/p/6b2762b3d98f`

5.3 消息认证码/MAC

认证是用来防止主动攻击，针对开放系统设计的。消息认证是用来验证信息完整性，而消息认证码是用来验证发送方非冒犯者。目的是实体认证和报文认证。
其算法是输入可变长度的消息以及密钥，输出固定长度的认证码。
构造方法一：Hash 函数以某种方式与密钥捆绑
构造方法二：使用对称分组密码，将可变长度的输入，变成固定长度的输入。
基于哈希函数的 MAC/HMAC
基本思想：输入密钥和信息，输出固定长度 nbits 认证码。
整体框架：$HMAC(K, M) = H(K^+XOR\ opad||H(K^+XOR\ ipad||M))$
算法步骤：
假定消息分组长度为 b，若秘钥长度 l，输出哈希长度为 n.

(1) 若密钥长度大于 b，哈希至 n 位；填充 0 至 b 位，得到 K^+

(2) 将 K^+ 与 00110110*(b/8) 异或

(3) 将消息接在 K^+ 后面

(4) 哈希得到 n 为哈希值

(5) 填充哈希值至 b 位

(6) 将 K^+ 与 01011100*(b/8) 异或

(7) 接第 (5) 步哈希值

(8) 再哈希的最终结果

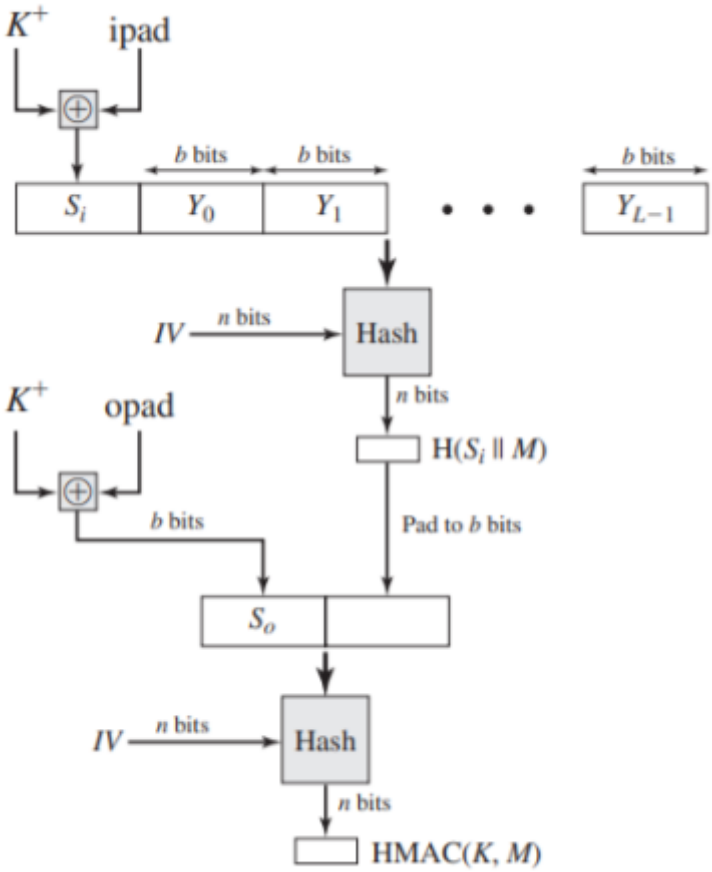

Figure 12.5 HMAC Structure

图 14: HMAC 结构

基于加密的 MAC/CMAC

利用对称分组加密算法 (如 AES) 对消息进行运算，得认证码。

算法步骤：

假定消息分组长度 b 位，共 n 组，K 为 k 位加密算法秘钥。K_1, K_2 为 b 位常量

(1) $C_1 = E(K, M_1)$

(2) $C_2 = E(K, M_2 xor\ C_1)$ 以此循环至

(3) $C_n = E(K, M_n xor\ C_{n-1} xor\ K_1)$

(4) $T = MSB_{Tlen}(C_n)$

T 是认证码，Tlen 是 T 位长度，$MSB_{Tlen}(C_n)$ 是指前 Tlen 位。

假若消息长度无法整除 b，那么填充 10* 使其可整除 b。然后在最后一步用 K_2 计算，关于 K_1 和 K_2 的生成：

$L = E(K, 0^b)$

$K_1 = L \cdot x$

$K_2 = L \cdot x^2$(乘法是伽瓦罗域上的乘法)

5.4 认证加密 CCM 与 GCM

CCM=CMAC+AES+CTR，发送方采用先认证后加密模式；接收方采用先解密后认证模式。
通信双方要共享密钥 K，计数器初始值 CTR_0，临时量 N，相关数据 A
发送方算法如下：

(1) N,A，与明文拼接，得可模 b 位的数据

(2) 然后用 K 以 CMAC 得认证值 T

(3) CTR_0 用 K 加密，前 len(Tag) 位与 Tag 异或

(4) 对明文进行 $AES - CTR$ 加密

(5) 后拼接 (3) 的数据，传给接收方

GCM=GCTR+GHASH，模式采用先加密后认证 (再加密)。以 GCTR 和 GHASH 为基本算法模块.
加密函数 $GCTR_k(ICB, X)$ 表示输入密钥 K，计数器值 ICB，明文 X，输出与 X 一样长的密文 Y
算法步骤如下:

(1) 加入 X 为空，直接输出空

(2) 将 X 分组为 $X_1||X_2...||X_n$(其中 n 小于或等于 128bits)

(3) 初始化计数器值：$CB_1 = ICB$

(4) $CB_i = inc_{32}(CB_{i-1})$ 即 CB_{i-1} 低 32 位加一后模 2^{32}

(5) $Y_i = X_i \ xor \ E(K, CB_i)$ 除了最后一组

(6) $Y_n = X_n \ xor \ MSB_{len(X_n)}(E(K, CB_n))$

(7) 输出 $Y = Y_1||Y_2...||Y_n$

哈希函数 $GHASH$ 需要哈希秘钥 H，以及不定长数据串，输出固定长度 (128bits) 的哈希值，算法步骤如下：

(1) 将数据分组：$X = X_1||X_2||...X_n$

(2) 初始化 Y；$Y_0 = 0^{128}$

(3) $Y_i = (X_i \ xor \ Y_{i-1}) \cdot H$

(4) 输出 Y_n

GCM 利用此两个函数，先将明文用 GCTR 加密，后填充相关参数数据，然后用 GHASH 哈希，然后在用 GCTR 加密，最后输出需求长度的验证码。

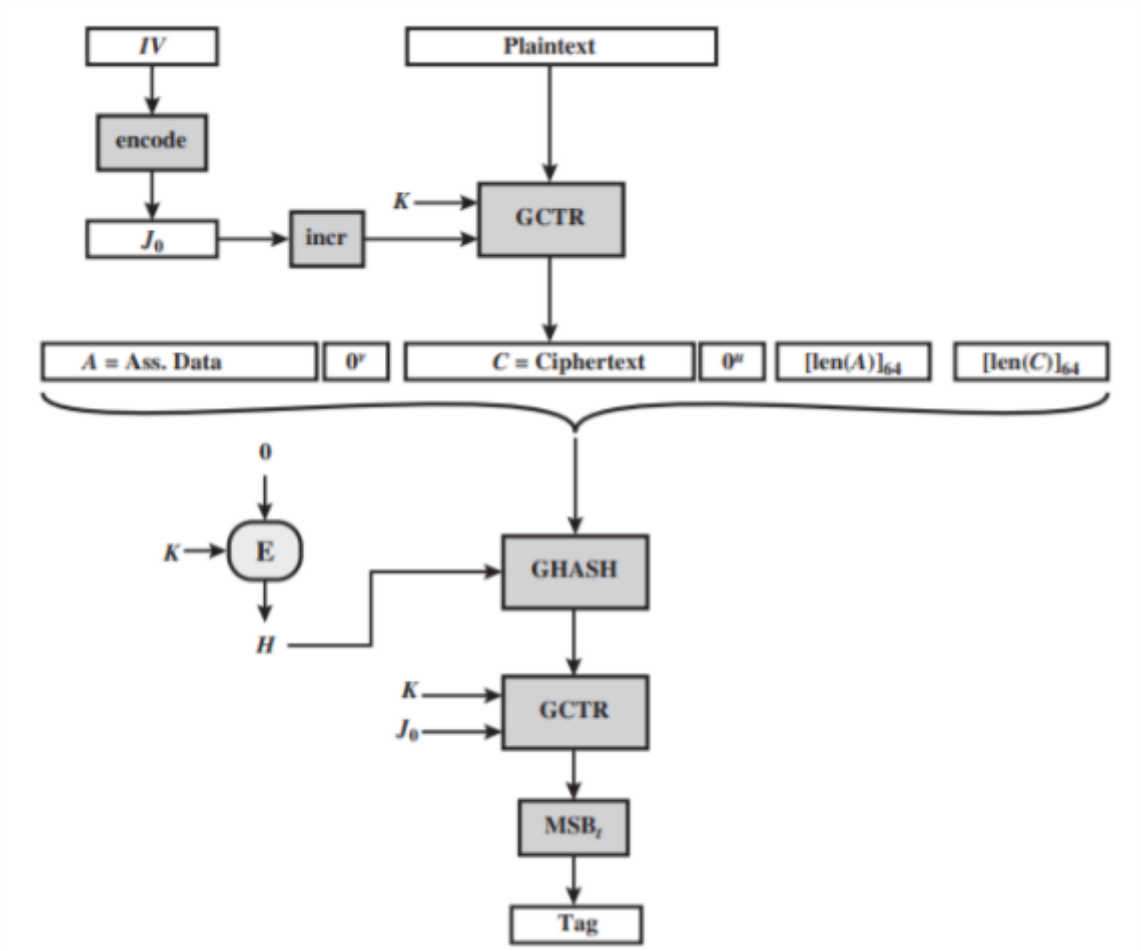

Figure 12.11 Galois Counter—Message Authentication Code (GCM)

图 15: GCM 结构

6 数字签名

数字签名的思路：利用哈希算法与非对称加密算法

6.1 ElGamal 数字签名

对于签名的一方 A：要有消息哈希值：$m = H(M)$，素数 q 及其本原根 α，以及临时私钥 X_A，随机数 K,K 满足 $gcd(K, q-1) = 1$，即 K 与 q-1 互素。
计算 $S_1 = \alpha^K \ mod \ q$，类似于 ElGamal 加密算法
计算 $S_2 = K^{-1}(m - X_A S_1) \ mod \ q - 1$，$S_2$ 作用于指数上，所以根据离散对数原理，包含 X_A，以及模 q-1. 计算完毕后，A 将 (S_1, S_2) 作为签名，以 (α, q, Y_A) 作为公钥。
对于认证的一方：收到 A 的消息、签名及公钥。做出如下计算：
$V_1 = \alpha^m \ mod \ q$
$V_2 = Y_A^{S_1} S_1^{S_2} \ mod \ q$
假若 $V_1 = V_2$ 那么，认证通过。

6.2 Schnorr 数字签名

Schnorr 签名采用先计算，在哈希的方法以提高计算效率。
对于签名的一方 A：要有两个素数 p, q，且 $q|p-1$，选择整数 $\alpha^q = 1 \, mod \, p$,(α, q, p) 构成公开参数；然后，选取 $0 < s < q$ 作为私钥，计算 $v = \alpha^{-s} \ mod \ p$ 为公钥。
产生签名算法如下：
随机生成 $0 < r < q$, 计算 $x = \alpha^r \ modp$, 计算哈希值 $e = H(M||x)$; 计算 $y = r + se \ mod \ q$ 签名便是 (e,y);
认证方利用公钥和已知参数进行验证：计算 $x' = \alpha^y v^e$,$e' = H(M||x')$.
假若 $e = e'$，那么，签名认证成功。

6.3 DSA 数字签名

首先需要公开参数：素数对 p, q, 且 $q|p-1$;
签名方需要私钥 $x, 0 < x < q$, 随机数 $k, 0 < k < q$,$g, g = h^{(p-1)/q} \ mod \ p, g \neq 1$;
生成公钥 $y = g^x \ mod \ p, r = (g^k \ mod \ p) \ mod \ q, s = k^{-1}[H(M) + xr] \ mod \ q$
得签名 (r, s)
验证方计算 (主要思路是计算到 $r = g^x$)：
需要指数上：$w = s^{-1} \ mod \ q, u_1 = H(M)w \ mod \ q, u_2 = wr \ mod \ q$
然后计算：$g^{u_1} y^{u_2} \ mod \ p \ mod \ q$ 若与 r 相同，即签名成功。

7 认证协议

7.1 弱认证/口令认证

用户传输一个口令，计算机根据该口令验证是否与该用户对应，以此确保用户身份。(例如：登陆操作)
安全性：口令认证主要存在字典攻击 (在线，离线)；字典攻击即通过穷举字典中的口令，猜测出用户的口令
故此，需要对口令进行加盐哈希操作。
用户将口令填充数据 (加盐)，然后将整个数据加密，并将“盐”和密文放进口令文件中。然后传输。
每次用户需要被认证时，服务器通过传输过来的口令，从口令文件中得盐再加密，与口令文件中密文比对，看是否一致，以此认证。
这样做的好处是防止重复口令被发现；增加字典攻击难度。

基于口令的秘钥交换：
公共参数：Z_p^* 上的生成元 g, M, M^{-1}。
用户 A 有口令 pw，且产生一个私钥 x, 计算公钥 $g^x * M^{pw}\ mod\ p$, 传送口令和公钥给 B
用户 B 产生一个私钥 y，根据公钥以及口令计算秘钥 $g^y * M^{-pw} * g^x * M^{pw}\ mod\ p = g^{xy}$ 并将产生的公钥 $g^y * M^{-pw}$ 传给 A；
A 采用同样算法产生密钥 g^{xy}

7.2 强认证/质询 -应答

认证与被认证双方协商好秘密 sk 以及加密函数 f，每一次用户需要被认证时，系统向用户发送一个质询 (challenge) 消息 m，用户以 $r = f(m, sk)$ 作为应答。系统通过验证 r 来确认用户的身份。

7.2.1 基于硬件

“token”：对消息提供哈希和加密机制，需要硬件支持时间戳，能保存密钥，以及生成同步信息。

- 基于时间戳的单边认证
  $A \to B : E_k(t, B)$
- 基于随机数 (单边认证)
  $B \to A : r$

$A \to B : E_k(r, B)$

- 基于随机数的双边认证
  $B \to A : r_B$
  $A \to B : E_k(r_A, r_B, B)$
  $B \to A : E_k(r_B, r_A)$

- 基于哈希函数的双边认证
  $B \to A : r_B$
  $A \to B : r_A, H(r_A, r_B, B)$
  $B \to A : H(r_A, r_B, A)$

7.2.2 基于公钥密码体制

- 单边：
  $B \to A : H(r), B, P_r(r, B)$
  $A \to B : r$

- 双边/Needham-Schroeder 协议
  $A \to B : P_B(r_A, A)$
  $B \to A : P_A(r_A, r_B)$
  $A \to B : r_B$

7.2.3 基于签名

- 基于时间戳
  $A \to B : cert_A, t, B, sign_A(t, B)$(sign 是签名算法)

- 基于随机数的单边认证：
  $B \to A : r_B$
  $A \to B : cert_A, r_A, B, sign(r_A, r_B, B)$

- 基于随机数的双边认证：
  $B \to A : r_B$
  $A \to B : cert_A, r_A, B, sign_A(r_A, r_B, B)$
  $B \to A : cert_B, A, sign_B(r_B, r_A, A)$

7.2.4 基于公钥体制认证的秘钥交换协议/HMQV

类似于 DH。针对双方 A 和 B：公私钥对 $(A = g^a, a)(B = g^b, b)$

$A \to B : X = g^x$

$B \to A : Y = g^y$

A 根据 B 传输的信息：Y 以及公钥 B 利用自己的私钥 a 与随机数 x 计算密钥：$(YB^{H(Y)})^{(x+aH(X))}$

B 根据 A 传输的信息：X 以及公钥 A 利用自己的私钥 b 与随机数 y 计算密钥：$(XA^{H(X)})^{(y+bH(Y))}$