第六章 公钥密码

胡 伟

网络空间安全学院

weihu@nwpu.edu.cn

章节课程安排

授课内容	学时
6.1.1 公钥密码原理 6.1.2 RSA公钥密码	2
6.2 RSA的实现	2
6.3 RSA时间侧信道攻击及防御	2
6.4.1 离散对数问题 6.4.2 EIGamal公钥密码 6.4.3 椭圆曲线离散对数问题 6.4.4 椭圆曲线公钥密码 6.4.5 中国商用椭圆曲线公钥密码SM2	2

课程回顾：分组密码的特点

- 分组密码（Block Cipher）的特点
  - 加解密速度快（相对于公钥密码）
  - 便于硬件实现
  - 易于标准化
- 分组密码的应用
  - 批量数据的加密
  - 伪随机数发生器
  - 消息认证码
  - 杂凑函数

课程回顾：分组密码的原理

- 将明文按规定的长度分组
- 实质是较复杂的单表代替密码
- 密文的一特比特与整个明文分组相关

密钥$k = (k_1, k_2, \ldots, k_n)$

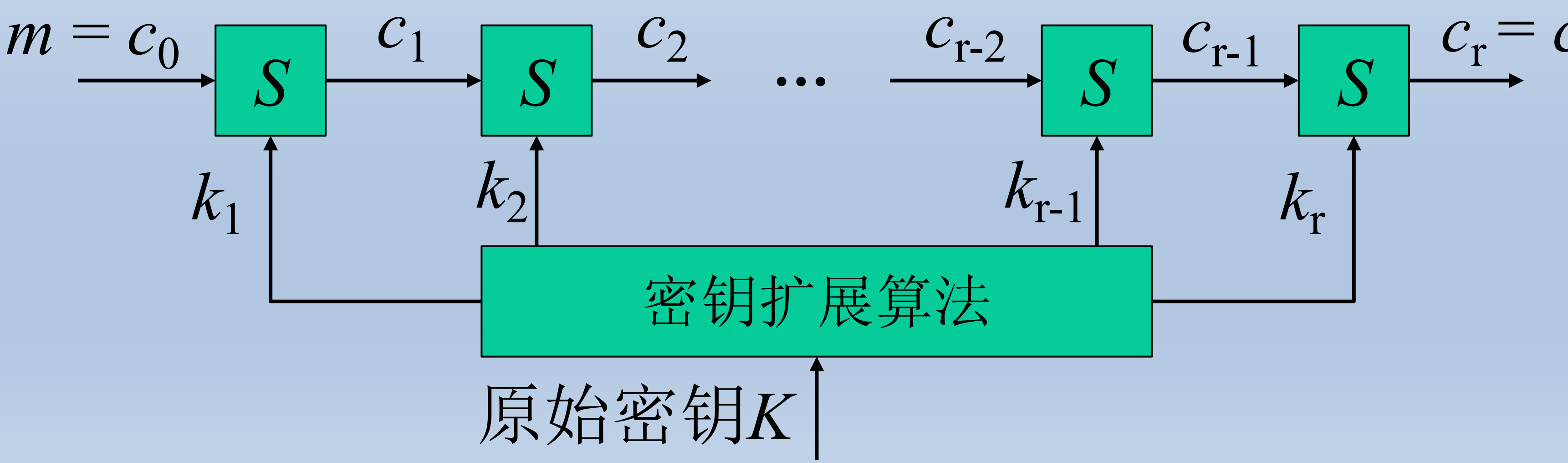
密钥$k = (k_1, k_2, \ldots, k_n)$

$m = (m_1, m_2, \ldots, m_n)$ $c = (c_1, c_2, \ldots, c_n)$ $m = (m_1, m_2, \ldots, m_n)$

明文 → 加密算法 → 密文 → 解密算法 → 明文

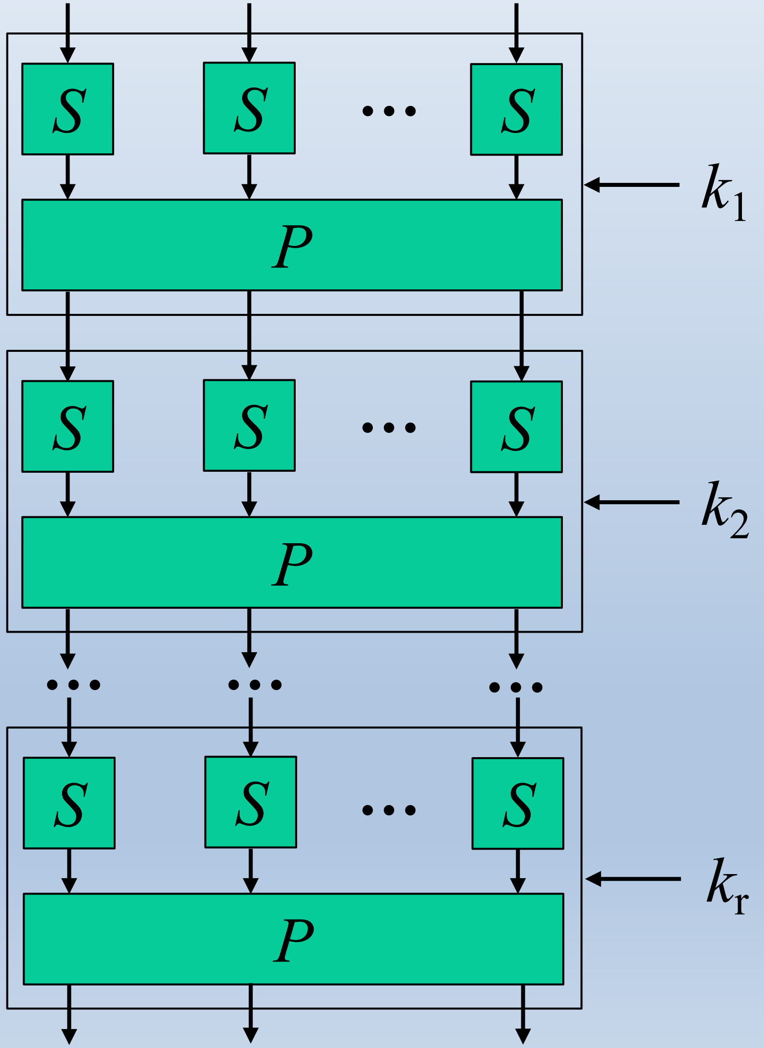
课程回顾：迭代密码

- 定义5.3：对于不非幂等的密码S，将自身做n次乘积得到的密码S^n称为S的n重迭代密码
  - 迭代密码可以通过简单密码得到高强度密码
  - 迭代密码是现代分组密码和杂凑函数的核心设计思想
- 迭代型分组密码将原始密钥经密钥扩展算法得到多个轮密钥，每一轮使用一个子密钥



课程回顾：S-P网络

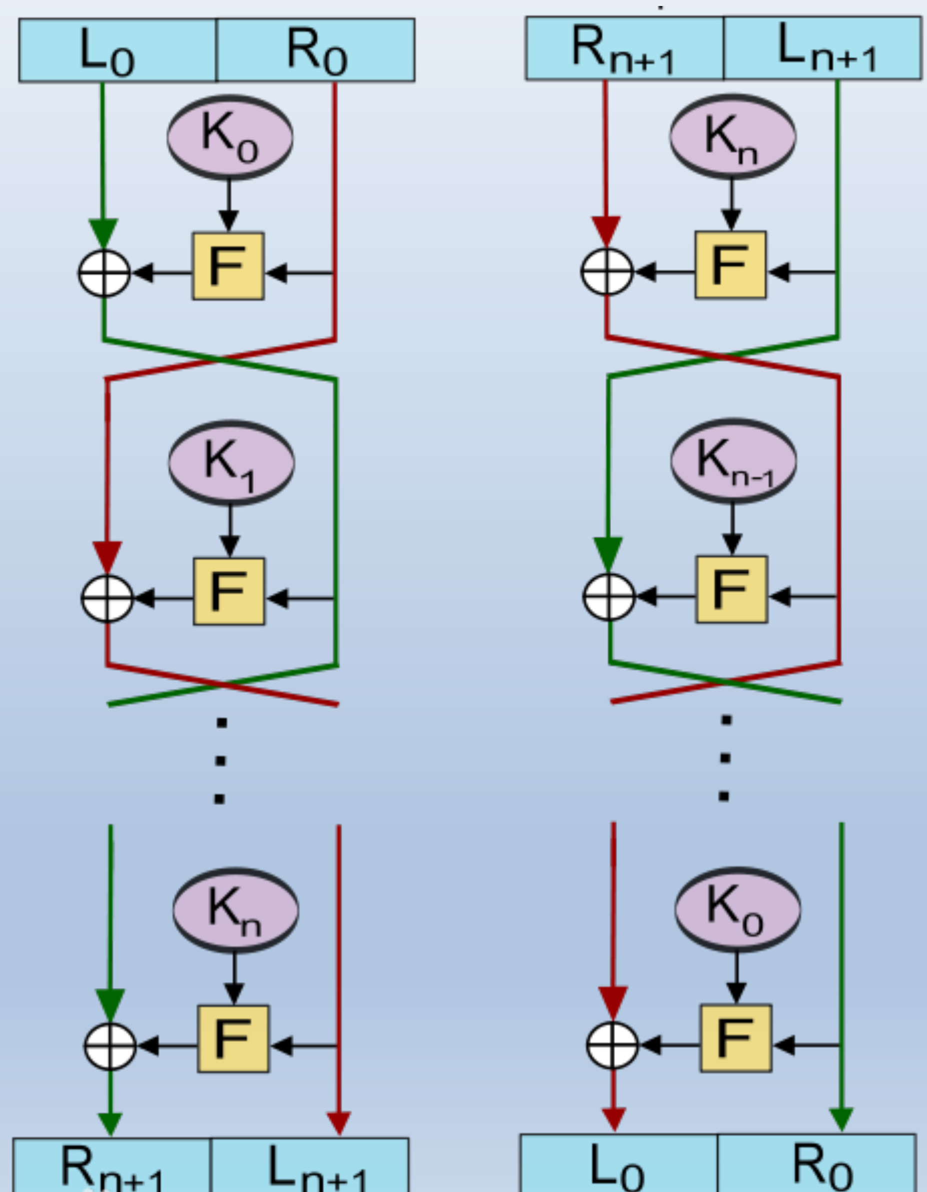
- 非线性代替S层实现分组小块的混淆和扩散
- 置换P层实现整体扩散
- S-P网络的特点
  - 结构简单
  - 扩散速度快
  - 加解密结构不同



课程回顾：Feistel模型

- 相对于S-P网络加解密速度更慢
- 优势在于加解密具有相同结构
- 轮密钥使用次序不同

$$\begin{cases} R_i = f(R_{i-1}, k_i) \oplus L_{i-1} \\ L_i = R_{i-1} \\ i = 1,2,\cdots,r \end{cases}$$



课程回顾：分组密码工作模式

加密模式		特 点
Electronic Code Book(ECB)	电子密码本模式	简单快速，可并行计算
Cipher Block Chaining(CBC）	密码分组链接模式	仅解密支持并行计算
Cipher Feedback Mode(CFB)	密文反馈模式	仅解密支持并行计算
Output Feedback Mode(OFB)	输出反馈模式	不支持并行运算
Counter (CTR)	计算器模式	支持并行计算

课程回顾：几种分组密码算法的比较

密码算法	正式 公布时间	网络结构	分组长度	密钥长度	轮数	S盒规模
DES	1977	Feistal	64	64 (56)	16	6→4
3-DES	1999	Feistal	64	112/168	16*3	6→4
AES	2001	S-P	128	128/192/256	10/12/14	8→8
SM4	2006	滑动窗口	128	128	32	8→8
PRESENT	2009	S-P	64	80/128	31	4→4

6.1.1 公钥密码原理

密码算法的分类

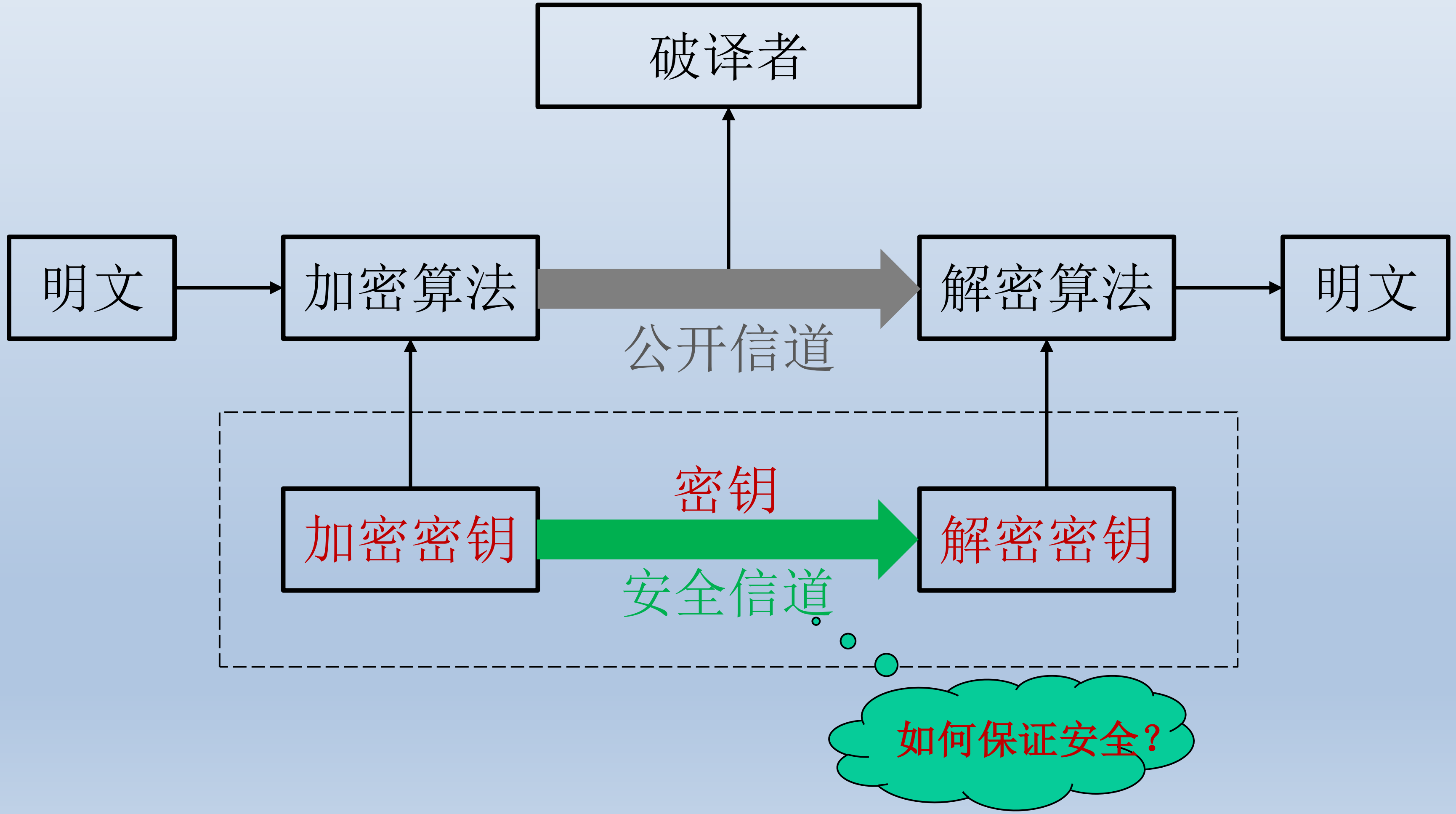
- 对称密码算法：序列密码和分组密码
  - 秘密密钥算法，单钥密码算法
  - 加密密钥与解密密钥相同，或实质上等同
  - 适合加密大量数据
  - 典型算法：DES、AES、IDEA、SM4
- 非对称密码算法
  - 公钥密码算法，双钥密码算法
  - 加密密钥与解密密钥不同，而且解密密钥不能根据加密密钥计算出来
  - 典型算法：RSA、ECC、SM2

公钥密码原理

- 公钥密码的产生背景
- 公钥密码的基本思想
- 公钥密码的工作方式

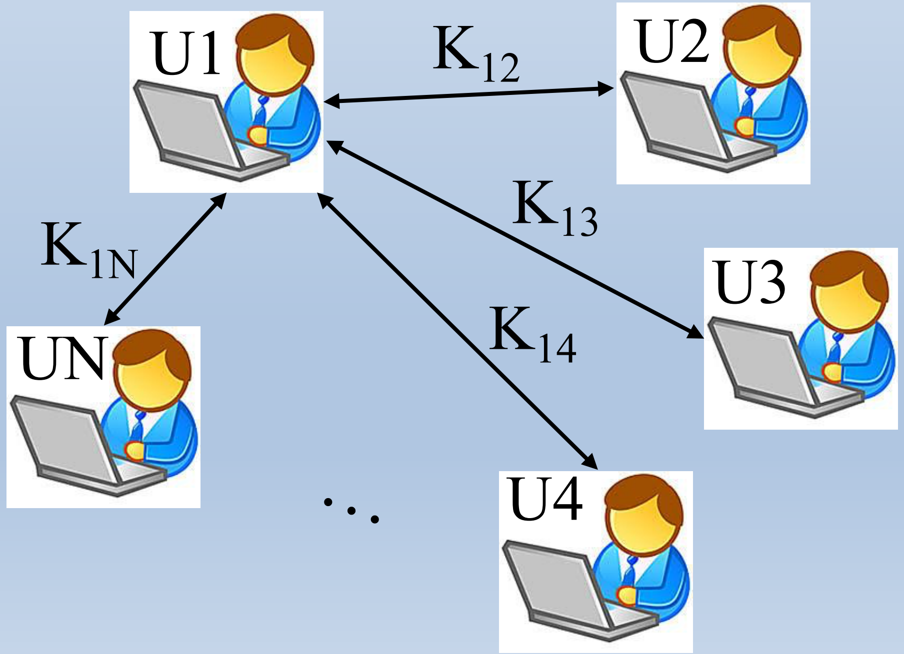
公钥密码的产生背景

- 传统密码体制的不足：密钥难共享



公钥密码的产生背景

- 传统密码体制的不足：密钥难管理
  - N个实体的网络中
  - 每对实体通信都需要一个共享密钥
  - 一共需要N*(N-1)/2个密钥
  - 还需要同样数量的保密信道用于密钥传输



公钥密码的产生背景

- 传统密码体制的不足：难以解决签名和认证问题
  - 对称密码算法能够对数据进行加解密（机密性）
  - 但是，由于通信双方共享密钥
  - 消息接收方可伪造原文
  - 消息发送方可否认所发消息
  - 未解决消息的真实性和不可否认性问题

会导致何问题？

机密性	→	防泄露
真实性	→	防假冒
完整性	→	防篡改
不可否认性	→	防抵赖

Diffie-Hellman密钥交换协议

- 20世纪70年代，斯坦福大学Diffie和Hellman研究了密钥分发问题，提出了一种通过公开信道共享密钥的方案，即Diffie-Hellman密钥交换协议
- 1976年， Diffie和Hellman发表了他们的结论（《密码学的新方向》（New Directions in Cryptography）的论文），论文概述了公钥密码的思想

美国计算机协会（ACM）将2015年的图灵奖授予Sun Microsystems的前首席安全官惠特菲尔德·迪菲（Whitfield Diffie）以及斯坦福大学电气工程系名誉教授马丁·赫尔曼（Martin Hellman），以表彰他们在现代密码学中所起的至关重要的作用。

公钥密码的基本思想

- 将密钥 K一分为二：K_e和 K_d。K_e专门加密，K_d专门解密， $K_e \neq$ Kd
- 由K_e不能计算出K_d，于是可将K_e公开，使密钥K_e分配简单
- 由于$K_e \neq K_d$且由K_e不能计算出K_d，所以K_d便成为用户的指纹，于是可方便地实现数字签名

称上述密码为公开密钥密码，简称为公钥密码

公钥密码的基本条件

- ① 保密条件：E和D互逆；$D(E(M)) = M$
- ② 安全条件：$K_e \neq K_d$且由K_e不能计算出K_d
- ③ 使用条件：E和D都高效；
- ④ 保真条件：$E(D(M)) = M$
  - 如果满足①②③可用于保密
  - 如果满足②③④可用于保真
  - 如果①②③④都满足，可同时用于保密和保真

公钥密码的理论模型

- **单向函数**: 设函数$y = f(x)$，如果满足以下两个条件，则称为单向函数：
  - 如果对于给定的x，要计算出$y = f(x)$很容易
  - 而对于给定的y，要计算出$x = f^{-1}(y)$很难
- 利用**单向函数构造密码**
  - 用正变换作加密，加密效率高
  - 用逆变换作解密，安全，敌手不可破译
  - 但是合法收信者也无法解密

单向函数是否适于构造密码

公钥密码的理论模型

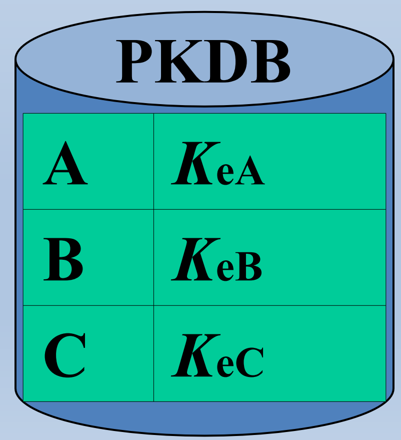
- 单向陷门函数：设函数$y = f(x)$，且f具有陷门，若满足以下条件，则称为单向陷门函数：
  - 如果对于给定的x，要计算出$y = f(x)$ 很容易
  - 而对于给定的y，如果不掌握陷门要计算出$x = f^{-1}(y)$很难，而如果掌握陷门要计算出$x = f^{-1}(y)$就很容易
- 利用单向陷门函数构造密码
  - 用正变换作加密，加密效率高
  - 用逆变换作解密，安全
  - 把陷门信息作为密钥，且只分配给合法用户。确保合法用户能够方便地解密，而非法用户不能破译

单向函数的研究现状

- 理论上：尚不能证明单向函数一定存在
- 实际上：密码学认为只要函数单向性足够应用即可
- 一些单向性足够的函数：
  - ① 大合数的因子分解问题
  - 大素数的乘积容易计算($p \times q \Rightarrow n$)，而大合数的因子分解困难($n \Rightarrow p \times q$)
  - ② 有限域上的离散对数问题
  - 有限域上大素数的幂乘容易计算($a^b \Rightarrow c$)，而对数计算困难($\log_a c \Rightarrow b$)
  - ③ 椭圆曲线离散对数问题
  - 设$\boldsymbol{d}$是正整数，**G**是解点群的基点，计算$\boldsymbol{dG = Q}$是容易的，而由$\boldsymbol{Q}$求出$\boldsymbol{d}$是困难的

公钥密码的工作方式

- 基本概念
  - 设M为明文，C为密文，E为加密算法，D为解密算法
  - 每个用户都配置一对密钥：K_e为公开的加密钥，K_d为保密的解密钥
  - 将所有用户的公开的加密钥K_e存入共享的密钥库PKDB
  - 保密的解密钥K_d由用户妥善保管



公钥密码的工作方式

- 确保数据机密性
- 发方
  - ① A首先查PKDB，查到B的公开的加密钥K_{eB}
  - ② A用K_{eB}加密M得到密文C：$C = E(M，K_{eB})$
  - ③ A发C给B
- 收方
  - ① B接收C
  - ② B用自己的K_{dB}解密，得到明文$M = D(\mathrm{C}, K_{dB}) = \mathrm{D(E(M，}K_{eB}), K_{dB})$

公钥密码的工作方式

- 确保数据秘密性（安全性分析）：
- ① 只有B才有K_{dB}，因此只有B才能解密，所以确保了数据的秘密性
- ② 任何人都可查PKDB得到B的K_{eB}，所以任何人都可冒充A给B发送数据。不能确保数据的真实性

公钥密码的工作方式

- 确保数据真实性
- 发方
  - ① A首先用自己的K_{dA}对M加密，得到$C = D(M, K_{dA})$
  - ② A发C给B
- 收方
  - ① B接收C
  - ② B查PKDB查到A的公开的加密钥K_{eA}
  - ③ B用K_{eA}加密C，得到明文$M = E(C, K_{eA}) = E(D(M, K_{dA}), K_{eA})$

公钥密码的工作方式

- 确保数据真实性（安全性分析）：
- ① 只有A才有K_{dA}，因此只有A才能解密产生C，所以确保了数据的真实性
- ② 任何人都可查PKDB得到A的K_{eA}，所以任何人都可加密得到明文。不能确保数据的秘密性

公钥密码的工作方式

- 同时确保数据机密性和真实性
- 发方
  - ① A首先用自己的K_{dA}对M加密，得到S：$S = D(M, K_{dA})$
  - ② A查PKDB，查到B的公开的加密钥K_{eB}
  - ③ A用K_{eB}加密S得到C：$C = E(S, K_{eB})$
  - ④ A发C给B
- 收方
  - ① B接收C
  - ② B用自己的K_{dB}解密C，得到S：$S = D(C, K_{dB})$
  - ③ B查PKDB，查到A的公开的加密钥K_{eA}
  - ④ B用A的公开密钥K_{eA}解密S，得到M：$M = E(S, K_{eA})$

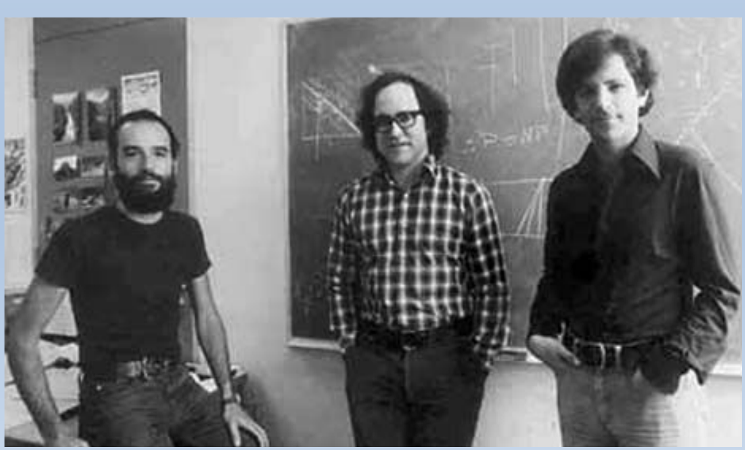
公钥密码的工作方式

- 同时确保数据机密性和真实性(安全性分析)
- ① 只有A才有K_{dA}，因此只有A才能解密产生S，所以确保了数据的真实性
- ② 只有B才有K_{dB}，因此只有B才能获得明文，所以确保了数据的机密性

6.1.2 RSA公钥密码

RSA密码概述

- 1977年由美国麻省理工学院的Ron Rivest、Adi Shamir和Len Adleman提出，1978年正式公布
- 算法建立在大整数因子分解的困难性之上
- 目前应用最广泛的公钥密码算法之一
- 既可用于加密，又可用于数字签名
- 目前常使用的密钥长度1024、2048、4096
- RSA的计算量远大于DES和AES, 加密速度慢



RSA密码算法

- 加解密算法
  - 随机地选择两个大素数p和q，而且保密
  - 计算$n = p * q$，将n公开
  - 计算$\varphi(n) = (p\text{-}1) * (q\text{-}1)$，对$\varphi(n)$保密
  - 随机地选取一个正整数e，$1 < e < \varphi(n)$且$(\mathrm{e}, \varphi(n)) = 1$，将$e$公开
  - 根据 $ed = 1 \bmod \varphi(n)$，求出d，并对d保密
  - 加密运算：$C = \mathrm{M}^{\mathrm{e}} \bmod n$
  - 解密运算：$M = \mathrm{C}^{\mathrm{d}} \bmod n$
  - 公开密钥$K_e = <e, n>$，保密密钥$K_d = <p, q, d, \varphi(n)>$

RSA算法论证

- 欧拉函数：对正整数n，欧拉函数$\varphi(n)$是小于或等于n的正整数中与n互质的数的数目
- 欧拉定理（也称费马-欧拉定理）：是一个关于同余的性质。若n, a为正整数，且n, a互质，则:

$$a^{\varphi(n)} \equiv 1 \, mod \, n$$

RSA算法论证

- E和D的可逆性(P125)
  - 要证明：$D(E(M)) = M$，即要证明

    $$M = C^d = (M^e)^d = M^{ed} \bmod n$$

  - 因为$ed = 1 \bmod \varphi(n)$，这说明$ed = t\varphi(n) + 1$,其中t为整数。所以，$M^{ed} = M^{t\varphi(n)+1} \bmod n$
  - 因此要证明 $M^{ed} = M \bmod n$，只需证明

    $$M^{t\varphi(n)+1} = M \bmod n$$

  - 在$(M, n) = 1$的情况下，根据数论(Euler定理)，

    $$M^{t\varphi(n)} = 1 \bmod n,$$

  - 于是$M^{t\varphi(n)+1} = M \bmod n$

RSA算法论证

- E和D的可逆性(P125)
  - 在$(M，n) \neq 1$的情况下，分两种情况：
  - 第一种情况：$M \in \{1, 2, 3, \ldots, n - 1\}$
  - 因为$n = pq$，p和q为素数，$M \in \{1, 2, 3, \ldots, n - 1\}$，且$(M, n) \neq 1$
  - 这说明M必含p或q之一为其因子，而且不能同时包两者，否则将有$M \geq n$，与$M \in \{1, 2, 3, \ldots, n - 1\}$矛盾
  - 不妨设$M = ap$。又因q为素数，且M不包含q，故有$(M, q) = 1$，于是有，

$$M^{\varphi(q)} = 1 \bmod q$$

RSA算法论证

- E和D的可逆性(P125)
  - 进一步有$M^{t(p-1)\varphi(q)}=1 \bmod q$。因为$q$是素数，$\varphi(q)=(q-1)$，所以有$t(p-1)\varphi(q)=t\varphi(n)$，

$$M^{t\varphi(n)}=1 \bmod q$$

  - 于是，$M^{t\varphi(n)}=bq+1$，其中b为整数。两边同乘M，$M^{t\varphi(n)+1}=bqM+M$。因为$M=ap$，故

$$M^{t\varphi(n)+1}=bqap+M=abn+M$$

  - 取模n得，$M^{\varphi(n)+1}=M \bmod n$
  - 第二种情况：M＝0
  - 当M＝0时，直接验证，可知命题成立

RSA算法论证

- 加密和解密运算的可交换性
  - $D(E(M)) = (M^e)^d = M^{ed} = (M^d)^e = E(D(M)) \bmod n$
  - 因此，**RSA**密码可同时确保数据的机密性和真实性
- 加解密算法的有效性
  - **RSA**密码的加解密运算是模幂运算，是比较效的

RSA算法的安全性

- 在计算上由公开的加密钥不能求出解密钥
  - 大合数的因子分解却是十分困难的
  - 大合数的因子分解的时间复杂度下限目前尚无定论
  - 迄今为止的各种因子分解算法提示人们这一时间下限将不低于O(EXP(lnNlnlnN)1/2)
  - 可见，只要合数足够大，进行因子分解是相当困难的

RSA算法的安全性

- 在计算上由公开的加密钥不能求出解密钥
  - 假设攻击者截获了密文C，想求出明文M
  - 他知道$M \equiv C^d \mod n$，因为n是公开的
  - 要从C中求出明文M，必须先求出d，而d是保密的
  - 但他知道，$ed \equiv 1 \bmod \varphi(n)$，$e$是公开的
  - 要从中求出d，必须先求出$\varphi(n)$，而$\varphi(n)$是保密的

RSA算法的安全性

- 在计算上由公开的加密钥不能求出解密钥
  - 但他又知道，$\varphi(n) = (p - 1)(q - 1)$，要从中求出$\varphi(n)$，必须先求出$p$和$q$，而$p$和$q$是保密的
  - 但他知道，$n = pq$，要从n求出p和q，只有对n进行因子分解。而当n足够大时，这是困难的
  - 只要能对n进行因子分解，便可攻破RSA密码
  - 此可以得出，破译RSA密码的困难性 $\leq$ 对n因子分解的困难性。目前尚不能证明两者是否能确切相等
  - 尚不能确知除了对n进行因子分解的方法外，是否还有别的更简捷的破译方法

课后学习任务

- RSA在线计算器，http://nmichaels.org/rsa.py
- 欧拉函数，https://baike.baidu.com/item/欧拉函数/1944850?fr=Aladdin
- 欧拉定理，https://baike.baidu.com/item/欧拉定理/891345?fr=aladdin

章节课程安排

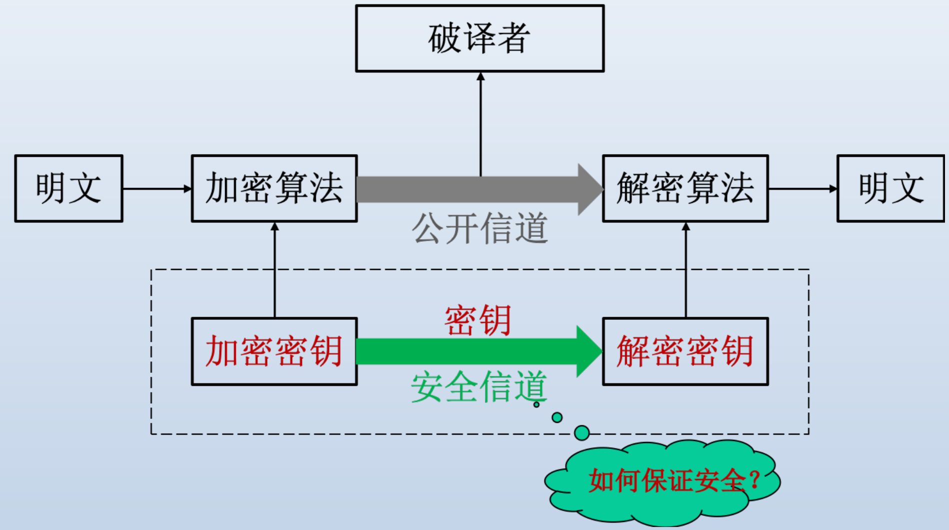
授课内容	学时
6.1.1 公钥密码原理 6.1.2 RSA公钥密码	2
6.2 RSA的实现	2
6.3 RSA时间侧信道攻击及防御	2
6.4.1 离散对数问题 6.4.2 EIGamal公钥密码 6.4.3 椭圆曲线离散对数问题 6.4.4 椭圆曲线公钥密码 6.4.5 中国商用椭圆曲线公钥密码SM2	2

密码算法的分类

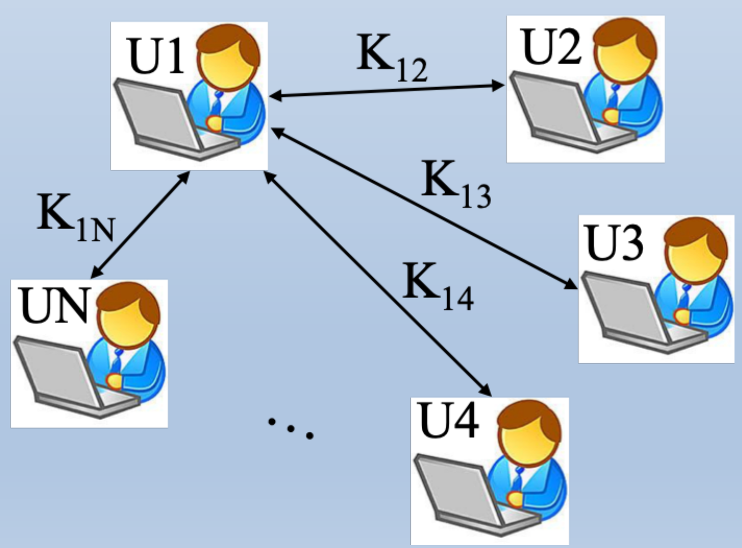
- 对称密码算法：序列密码和分组密码
  - 秘密密钥算法，单钥密码算法
  - 加密密钥与解密密钥相同，或实质上等同
  - 适合加密大量数据
  - 典型算法：DES、AES、IDEA、SM4
- 非对称密码算法
  - 公钥密码算法，双钥密码算法
  - 加密密钥与解密密钥不同，而且解密密钥不能根据加密密钥计算出来
  - 典型算法：RSA、ECC、SM2

课程回顾：传统密码体制的不足

- 密钥难共享
- 密钥难管理
- 无法解决签名和认证问题



N*(N-1)/2个密钥



机密性	防泄露
真实性	防假冒
完整性	防篡改
不可否认性	防抵赖

Diffie-Hellman密钥交换协议

- 20世纪70年代，斯坦福大学Diffie和Hellman研究了密钥分发问题，提出了一种通过公开信道共享密钥的方案，即Diffie-Hellman密钥交换协议
- 1976年， Diffie和Hellman发表了他们的结论（《密码学的新方向》（New Directions in Cryptography）的论文），论文概述了公钥密码的思想

美国计算机协会（ACM）将2015年的图灵奖授予Sun Microsystems的前首席安全官惠特菲尔德·迪菲（Whitfield Diffie）以及斯坦福大学电气工程系名誉教授马丁·赫尔曼（Martin Hellman），以表彰他们在现代密码学中所起的至关重要的作用。

公钥密码的基本条件

- ① 保密条件：E和D互逆；$D(E(M)) = M$
- ② 安全条件：$K_e \neq K_d$且由K_e不能计算出K_d
- ③ 使用条件：E和D都高效；
- ④ 保真条件：$E(D(M)) = M$
  - 如果满足①②③可用于保密
  - 如果满足②③④可用于保真
  - 如果①②③④都满足，可同时用于保密和保真

公钥密码的工作方式

- 保证机密性

$$A \xrightarrow{K_{eB}(M)} B$$

- 保证真实性

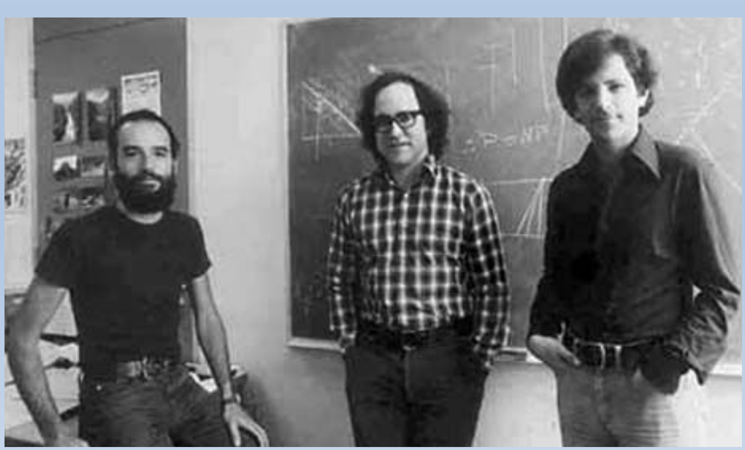
$$A \xrightarrow{K_{dA}(M)} B$$

- 同时保证机密性和真实性

$$A \xrightarrow{K_{eB}(K_{dA}(M))} B$$

RSA密码概述

- 1977年由美国麻省理工学院的Ron Rivest、Adi Shamir和Len Adleman提出，1978年正式公布
- 算法建立在大整数因子分解的困难性之上
- 目前应用最广泛的公钥密码算法之一
- 既可用于加密，又可用于数字签名
- 目前常使用的密钥长度1024、2048、4096
- RSA的计算量远大于DES和AES, 加密速度慢



RSA密码算法

- 加解密算法
  - 随机地选择两个大素数p和q，而且保密
  - 计算$n = p * q$，将n公开
  - 计算$\varphi(n) = (p-1) * (q-1)$，对$\varphi(n)$保密
  - 随机地选取一个正整数e，$1 < e < \varphi(n)$且$(\mathrm{e}, \varphi(n)) = 1$，将$e$公开
  - 根据 $ed = 1 \bmod \varphi(n)$，求出d，并对d保密
  - 加密运算：$C = M^e \bmod n$
  - 解密运算：$M = C^d \bmod n$
  - 公开密钥$K_e = <e, n>$，保密密钥$K_d = <p, q, d, \varphi(n)>$

RSA安全性分析

- 公开密钥$K_e = <e, n>$
- 保密密钥$K_d = <p, q, d, \varphi(n)>$
- 假设攻击者截获密文C，*需恢复明文*

属于哪一类攻击

$$n = pq$$

$$\varphi(n) = (p - 1)(q - 1)$$

$$ed \equiv 1 \bmod \varphi(n)$$

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

$$C$$

6.2.1 RSA的实现

参数选择

- (1) ***p*和*q*要足够大**
  - RSA分解当前的记录是768位2进制数（232位10进制数）
  - 目前主流的密钥长度为1024位 – 4096位
  - 一般应用：*p*和*q*应至少为512位，使n达1024位
  - 重要应用：*p*和*q*应至少为1024位，使n达2048位
- (2) ***p*和*q*应为强素数**

定义6.3: p为素数，若p满足以下两个条件，则称p为强素数或一级素数
(1) 存在两个大素数p_1和p_2，使得$p_1 \mid p-1$，$p_2 \mid p+1$
(2) 存在4个大素数r_1，r_2，r_3和r_4，使得$r_1 \mid p_1-1$，$r_2 \mid p_1+1$，$r_3 \mid p_2-1$，$r_4 \mid p_2+1$

只要**(*p*-1)**、**(*p*+1)**、**(*q*-1)**、**(*q*+1)**之一有小的素因子，***n***就容易分解

https://en.wikipedia.org/wiki/RSA_Factoring_Challenge

参数选择

- (3) p和q位数相差不能太小，也不能太大
- 若p和q相差很小，可以估算$(p+q)/2$约为$\sqrt{n}$，可以用$\sqrt{n}$来估算 $(p+q)/2$，联合$p*q$和$p+q$即可分解n
  - 例，假设$p=2$，$q=3$，$n=6$，$(p+q)/2=2.5$, $\sqrt{6}\approx 2.45$，$p+q$应该在4.5附近取值
  - 例，假设p和q相差很小，$n=164009$，$\sqrt{n}\approx 405$，可以估计$(p+q)/2=405$，又$p*q=164009$，可得$p=409$，$q=401$
- 若p和q相差太大，则从可从小的素数开始尝试分解n

参数选择

- (4) 在给定的条件下，找到的$d = e$，这样的密钥必须舍弃
- (5) $p-1$和$q-1$的最大公因子要小，最好是2，可选择p和q为理想的强素数
  - 在唯密文攻击中，假设攻击者截获了某个密文

$$C_1 = M^e \bmod n$$

  - 攻击者进行迭代攻击

$$C_i = C_{i-1}^e = M^{e^i} \bmod n$$

  - 如果有$e^i = 1 \bmod n$，则有$C_i = M \bmod n$
  - 如果使得$e^i = 1 \bmod n$成立的i值很小，则很容易进行密文迭代攻击

参数选择

- 如果有$e^i = 1 \bmod n$，根据欧拉定理

$$i = \varphi(\varphi(n)) = \varphi((p-1)(q-1))$$
$$= \varphi(p-1)\varphi(q-1)D/\varphi(D)$$

  - 其中$D = gcd(p-1, q-1)$，$D/\varphi(D)$随D减小而增加，从而使i增大
  - $p-1$和$q-1$的最大公因子最好是2，可选择p和q为理想的强素数，设$p = 2a+1$，$q = 2b+1$，其中a和b为素数，则有

$$i = \varphi(\varphi(n)) = \varphi(2a*2b)$$
$$=2\,\varphi(a)\varphi(b) = 2(a-1)(b-1)$$

欧拉定理（也称费马-欧拉定理）：是一个关于同余的性质。若n, a为正整数，且n, a互质，则:

$$a^{\varphi(n)} \equiv 1 \bmod n$$

参数选择

- 例，设$p = 17$，$q = 11$，$n = 17*11 = 187$，$e = 7$，$M = 123$，则有
  - $C_0 = M = 123$
  - $C_1 = {C_0}^e = 123^7 \ mod \ 187 = 183$
  - $C_2 = {C_1}^e = 183^7 \ mod \ 187 = 72$
  - $C_3 = {C_2}^e = 72^7 \ mod \ 187 = 30$
  - $C_4 = {C_3}^e = 30^7 \ mod \ 187 = 123 = M$
  - $C_5 = {C_4}^e = 123^7 \ mod \ 187 = 183$
- 可见，迭代加密出现$C_5 = C_1$，$C_4 = M$，周期$t = 4$，$\varphi(n) = (p-1)(q-1) = 160$，周期$t$是$\varphi(n)$的因子

参数选择

- (6) e的选择
  - 随机且二进制表示中含1多就安全，但加密速度慢
  - 为了使加密速度快，二进制表示中含1应尽量少
  - 有的学者建议取$e = 2^{16} + 1 = 65537$，它是素数，且二进制表示中只含两个1
  - 若e太小，对于小的明文M，$C = M^e$未超过n，无需对n取模，此时直接对C开e次放即可求出明文M，也不安全
- (7) d的选择
  - 为了使加密速度快，希望选用尽量小的d
  - d不能太小，要足够大，否则不安全
  - 当d小于n的1/4时，已有求出d的攻击方法

参数选择

- 模数n的使用限制
  - 不要许多用户共用一个模n，否则易受共模攻击
  - 设用户A的加密密钥为e_A，用户B的加密密钥为e_B，他们使用同一个模数n，对于同一条明文有

$$C_A = M^{e_A} \bmod n$$
$$C_B = M^{e_B} \bmod n$$

  - 当e_A和e_B互素时，可利用欧几里德算法求出两个整数r和s，使得

$$re_A + se_B = 1$$

  - 于是，${C_A}^r {C_B}^s = M^{re_A + se_B} = M \bmod n$

素数产生

- 根据素数的定义，因子只有1和p本身
- 测试算法
  - $1 \sim p-1$，能够整除p的整数的个数
  - $1 \sim \sqrt{p}$即可
  - 如何快速筛选出1到整数n之间的所有素数？

更有效的算法？

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	…
✓		×		×		×		×		×		×		×		×		×		…

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	…
✓	✓	×		×		×	×	×		×		×	×	×		×		×	×	…

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	…
✓	✓	×	✓	×	✓	×	×	×	✓	×	✓	×	×	×	✓	×	✓	×	×	…

大素数的产生

- 概率产生法
  - 目前最常用的概率性算法是Miller检验算法
  - Miller检验算法已经成为美国的国家标准
- Miller检验算法
  - 欧拉定理：若n, a为正整数，且n, a互素，则有$a^{\varphi(n)} \equiv 1 \bmod n$
  - 费马小定理（欧拉定理的特殊情况）：如果p是一个素数，而整数a不是p的倍数（a和p互素），则有$a^{p-1} \equiv 1 \bmod p$
  - 不断取$a \in [1，p-1]$，且$a \in Z$，验证$a^{p-1} \equiv 1 \bmod p$是否成立，不成立则p肯定不是素数，共取s次
  - 若s次均通过测试，则p不是素数的概率不超过2^{-s}

RSA密钥产生及加解密

- OpenSSL，http://slproweb.com/products/Win32OpenSSL.html

Download Win32/Win64 OpenSSL

Download Win32/Win64 OpenSSL today using the links below!

File	Type	Description
Win64 OpenSSL v1.1.1d Light EXE | MSI (experimental)	3MB Installer	Installs the most commonly used essentials of Win64 OpenSSL v1.1.1d (Recommended for users by the creators of OpenSSL). Only installs on 64-bit versions of Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win64 OpenSSL v1.1.1d EXE | MSI (experimental)	43MB Installer	Installs Win64 OpenSSL v1.1.1d (Recommended for software developers by the creators of OpenSSL). Only installs on 64-bit versions of Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win32 OpenSSL v1.1.1d Light EXE | MSI (experimental)	3MB Installer	Installs the most commonly used essentials of Win32 OpenSSL v1.1.1d (Only install this if you need 32-bit OpenSSL for Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win32 OpenSSL v1.1.1d EXE | MSI (experimental)	30MB Installer	Installs Win32 OpenSSL v1.1.1d (Only install this if you need 32-bit OpenSSL for Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win64 OpenSSL v1.1.0L Light	3MB Installer	Installs the most commonly used essentials of Win64 OpenSSL v1.1.0L (Recommended for users by the creators of OpenSSL). Only installs on 64-bit versions of Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win64 OpenSSL v1.1.0L	37MB Installer	Installs Win64 OpenSSL v1.1.0L (Recommended for software developers by the creators of OpenSSL). Only installs on 64-bit versions of Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win32 OpenSSL v1.1.0L Light	3MB Installer	Installs the most commonly used essentials of Win32 OpenSSL v1.1.0L (Only install this if you need 32-bit OpenSSL for Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win32 OpenSSL v1.1.0L	30MB Installer	Installs Win32 OpenSSL v1.1.0L (Only install this if you are a software developer needing 32-bit OpenSSL for Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win64 OpenSSL v1.0.2t Light	3MB Installer	Installs the most commonly used essentials of Win64 OpenSSL v1.0.2t (Recommended for users by the creators of OpenSSL). Only installs on 64-bit versions of Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win64 OpenSSL v1.0.2t	23MB Installer	Installs Win64 OpenSSL v1.0.2t (Recommended for software developers by the creators of OpenSSL). Only installs on 64-bit versions of Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win32 OpenSSL v1.0.2t Light	2MB Installer	Installs the most commonly used essentials of Win32 OpenSSL v1.0.2t (Only install this if you need 32-bit OpenSSL for Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
Win32 OpenSSL v1.0.2t	20MB Installer	Installs Win32 OpenSSL v1.0.2t (Only install this if you are a software developer needing 32-bit OpenSSL for Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.

RSA密钥产生及加解密

- 产生私钥
  - openssl rsa -pubout -in rsa_2048_priv.pem -out rsa_2048_pub.pem
- 产生公钥
  - rsa -pubout -in rsa_2048_priv.pem -out rsa_2048_pub.pem
- RSA加密
  - openssl rsautl -encrypt -inkey rsa_2048_pub.pem -pubin -in plaintext.txt -out ciphertext.txt
- RSA解密
  - openssl rsautl -decrypt -inkey rsa_2048_priv.pem -in ciphertext.txt -out plaintext2.txt

RSA的实现

- 加解密运算
  - 加密运算：$C = M^e \mod n$
  - 解密运算：$M = C^d \mod n$
- 模幂运算算法
  - 基本定义
  - 重复平方算法
  - 滑动窗口算法
  - CRT算法
  - 蒙哥马利算法

基本定义

- 模幂运算的基本定义
  - $C = ((M * M \bmod \mathrm{n}) * M \bmod \mathrm{n}) * M \bmod n\ldots$
  - 简单直观
  - 只适用于e很小的情况
  - 当e较大（如$e = 65537$）时效率低

```
unsigned long mod_exp(unsigned long m, unsigned long e, unsigned long n)
{
    unsigned long result = 1;
    while (e > 0){
        result = (result * m) % n;

        e = e - 1;
    }

    return result;
}
```

重复平方算法

- 从右至左：$M = C^d \mod n$

$$d = (d_{w\text{-}1}, d_{w\text{-}2}, \ldots, d_1, d_0)$$

$$d = d_{w\text{-}1}*2^{w\text{-}1}+ d_{w\text{-}2}*2^{w\text{-}2}+ \ldots+ 2^1*d_1 + 2^0*d_0$$

$$M = C^d \mod n = C^{d_{w-1}*2^{w-1}+d_{w-2}*2^{w-2}+\cdots+d_1*2^1+d_0}$$

$$= (C^{d_{w-1}})^{2^{w-1}} * (C^{d_{w-2}})^{2^{w-2}} * \cdots (C^{d_1})^2 * C^{d_0}$$

- 通过逐次平方，依次计算$C^{2^{w-1}}$， $C^{2^{w-2}}$， …, C^2
- 当$d_i = 1\ (0 \le i \le w-1)$时， $C^{d_i} = 1$，结果乘以C^{2^i}
- 当$d_i = 0$时，乘以 1 (即1^{2^i})

重复平方算法

❖ 从右至左：$M = C^d \bmod n$

```
m[0]:=1
s[0]:=c
for i := 0 to w−1 do
    if d[i] == 1 then
        m[i+1] := m[i] * s[i] mod n
    else
        m[i + 1] := m[i] * 1
    end if
    s[i+1] := s[i] * s[i] mod n
end for
return m[w]
```

重复平方算法

❖ 从右至左：$M = C^d \mod n$

```
// right to left
unsigned long Rep_Sqr(unsigned long c, unsigned long d, unsigned long n)
{
    unsigned long result = 1;

    while (d > 0){
        if ((d & 0x01) == 1)
            result = (result * c) % n;

          c = (c * c) % n;
          d >>= 1;
    }

    return result;
}
```

例，计算$5^{13} \bmod 17$

d & 0x01	c	result	c'

重复平方算法

- 从左至右：$M = C^d \mod n$

$$d = (d_{w\text{-}1}, d_{w\text{-}2}, \ldots, d_1, d_0)$$

$$d = d_{w\text{-}1}*2^{w\text{-}1}+ d_{w\text{-}2}*2^{w\text{-}2}+ \ldots + 2^1*d_1 + 2^0*d_0$$

$$M = C^d \mod n = C^{d_{w-1}*2^{w-1}+d_{w-2}*2^{w-2}+\cdots+d_1*2^1+d_0}$$

- 从最高的非0密钥位($d_{w\text{-}1} = 1$)开始
- 以$C^{d_i*2^i}$的计算为例

$$d_i * 2^i = \left(d_i * 2^{i-1}\right) * 2 = \ldots = (d_i * 2^0)* \overbrace{2 * 2 * \ldots * 2}^{i\text{个}2\text{，即}2^i}$$

当$d_i = 1$时，$C^{d_i*2^i} = C^{2^i}$，逐次平方

当$d_i = 0$时，$C^{d_i*2^i} = 1$，逐次平方仍然为1

重复平方算法

❖ 从左至右：$M = C^d \bmod n$

```
s[w] := 1
for i := w − 1 to 0 do
    if d[i] == 1 then
        m[i] := s[i + 1] * c mod n
    else
        m[i] := s[i + 1] * 1
    end if
    s[i] := m[i] * m[i] mod n
end for
return m[0]
```

重复平方算法

❖ 从左至右：$M = C^d \mod n$

```
// left to right
unsigned long Rep_Sqr(unsigned long c, unsigned long d, unsigned long n)
{
    unsigned long d_val, d_width = 1;
    unsigned long s = 1, result;

    d_val = d; // copy d to d_val for processing
    while(d_val > 1){ // determine the position of the MSB of d
        d_width <<= 1;
        d_val >>= 1;
    }

    while (d_width > 0){
        if ((d & d_width) != 0) result = (s * c) % n;
        else result = s;

        s = (result * result) % n;
        d_width >>= 1;
    }

    return result;
}
```

重复平方算法

❖ 从左至右：$M = C^d \mod n$

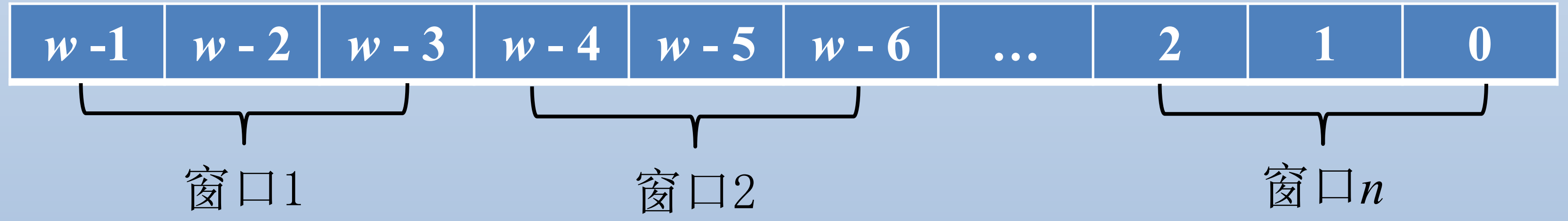
```
// left to right
unsigned long Rep_Sqr(unsigned long c, unsigned long d, unsigned long n)
{
    unsigned long d_val, d_width = 1;
    unsigned long s = 1, result;

    …

    while (d_width > 0){
        if ((d & d_width) != 0) result = (s * c) % n;
        else result = s;

        s = (result * result) % n;
        d_width >>= 1;
    }

    return result;
}
```

d & d_width	s	result	s'

例，计算$5^{13} \mod 17$

滑动窗口算法

- 每次处理一个窗口大小(k比特密钥)
- 分为固定窗口和可变窗口
- 分为从左至右和从右至左
- 重复平方法是窗口大小为1的特例



滑动窗口算法

❖ 滑动窗口算法：$M = C^d \mod n$

输入c，$d = (d_t, d_{t-1}, \ldots, d_1, d_0)$，其中$d_t = 1$，整数$k \geq 1$

输出c^d

1. 预计算 // $c, c^2, c^3, \ldots, c^{2^k-1}$

   $g_1 = c, g_2 = c^2$

   for $i = 1$ to $2^{k-1} - 1$ do $g_{2i+1} = g_{2i-1} * g_2$ // 初始化表

2. $A = 1, i = t$

3. while $i \geq 0$ do {

   if($d_i == 0$) then $A = A^2, i = i - 1$

   else { 寻找$i - L + 1 \leq k$，且$d_L = 1$的最长密钥串$p = (d_i, d_{i-1}, \ldots, d_L)$

   $A = A^{2^{i-L+1}} * g_p, i = L - 1$

   }

}

CRT算法

- 计算$M = C^d \bmod n$分两步进行
- 首先，计算$m_1 = C^{d1} \bmod n$，$m_2 = C^{d2} \bmod n$，其中d_1和d_2根据CRT算法由d预算计算得到
- 然后，使用CRT算法组合m_1和m_2的到m

蒙哥马利算法

- 优化模乘$x \cdot y \bmod q$运算中取模环节
- 原始取模操作：除法取余
- 蒙哥马利算法：转化为模减
- 算法思想：将以q为模的约简转化为以2^n为模的约简
  - 以R表示约简操作
  - 首先将变量转化为Montgomery形式，如x转化为$xR \bmod q$
  - $xR \cdot yR = zR^2$, 通过Montgomery约简得$zR^2 \cdot R^{-1} = zR$
  - 约简后的zR仍为Montgomery形式，可继续参与后续运算
  - 最终结果乘以$R^{-1} \bmod q$转回标准形式（非Montgomery ）

课后任务

- 筛选出2 - 1000000范围内的全部素数
- *openssl*实践环节
- 阅读和理解Miller-Rabbin算法
- 选择一个方法实现RSA
  - 重复平方法（从左至右）
  - 重复平方法（从右至左）
  - 滑动窗口法