中山大學
SUN YAT-SEN UNIVERSITY

Module I. Fundamentals of Information Security

Chapter 2
Cryptographic Techniques

Web Security: *Principles & Applications*

School of Data & Computer Science, Sun Yat-sen University

Outline

1. Introduction

2.4.1 Introduction

- **Public Key Cryptography**
  - Asymmetric Cryptography is a cryptographic system that uses two keys for message encryption - a *public key* known to everyone and a *private* or *secret key* known only to the recipient of the message.
- **Scenery**
  - When *Bob* wants to send a secure message to *Alice*, he uses *Alice*'s public key to encrypt the message. *Alice* then uses her private key to decrypt it.
- **Property**
  - It is important that the public and private keys are related in such a way that only the public key can be used to encrypt messages and only the corresponding private key can be used to decrypt them. Moreover, it is virtually impossible to deduce the private key from the known public key.

1. Introduction

- 一个经典的非对称加密信息传输情景
  - *Bob* 需要通过不可靠的传信人 *Tracy* 送一个信物给 *Alice*。
    1. *Bob* 将物品放进一个结实的箱子里，加上一把锁 L_1，只有 *Bob* 有 L_1 的钥匙 K_1；
    2. *Tracy* 将用 L_1 锁好的箱子送给 *Alice*；
    3. *Alice* 给箱子再加上一把锁 L_2，只有 Alice 有 L_2 的钥匙 K_2；
    4. *Tracy* 将用 L_1 和 L_2 锁好的箱子送回给 *Bob*；
    5. *Bob* 用 K_1 打开 L_1，箱子上留下了*Alice* 的锁 L_2；
    6. *Tracy* 将用 L_2 锁好的箱子送给 *Alice*；
    7. *Alice* 用 K_2 打开 L_2，取得物品。
  - 注意到：
    - (1) K_1 和 K_2 不同；(2) 没有发生钥匙的传递，也即 K_1 和 K_2 分别是 *Bob* 和 *Alice* 的私钥；(3) *Tracy* 走了3趟。

1. Introduction

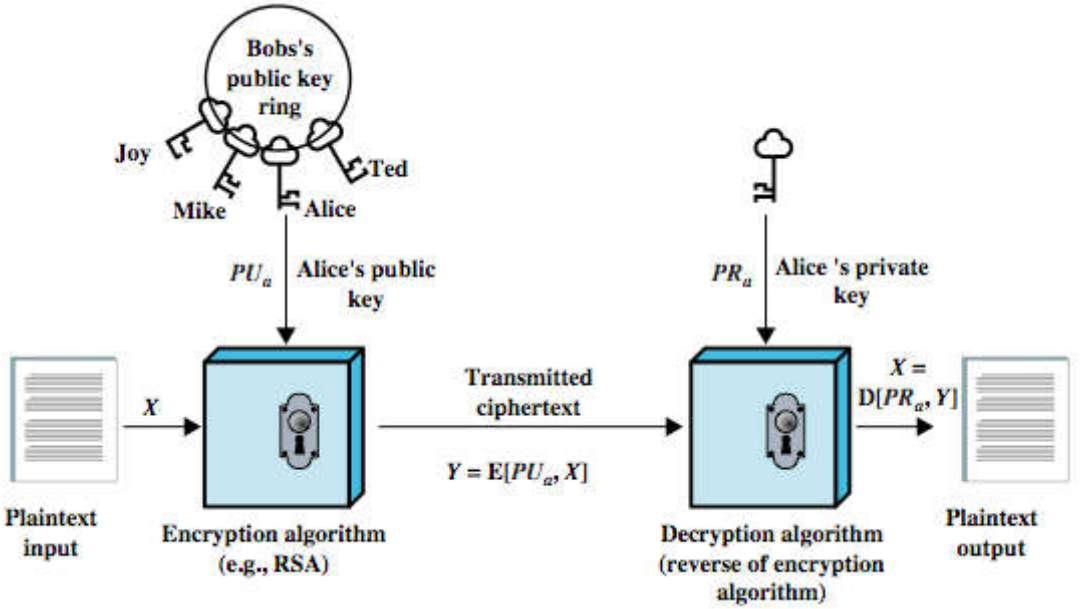
- 公钥密码学的起源
  - 1976年 *Whitfield Diffie* 和 *Matin Hellman* 发表了 “New Directions in Cryptography” (密码技术的新方向)，奠定了公钥密码学的基础。
    - ✧ *Whitfield Diffie* 和 *Matin Hellman* 据此获得2015年图灵奖。
  - 公钥密码技术是二十世纪最伟大的思想之一，它改变了密钥分发的方式，并可以广泛用于数字签名和身份认证服务。

1. Introduction

- 公钥密码学的基本思想
  - 非对称加密算法需要两个密钥：公开密钥 (公钥, public-key) 和私有密钥 (私钥, private-key)，因此也称公钥密码学算法。参与加密通信的主体生成一对密钥，将其中的一把作为私有密钥严密收藏，另一把作为公开密钥向其它方公开。
  - 公开密钥与私有密钥是严格对应的，如果用一个公开密钥对数据进行加密，就只有用其对应的私有密钥才能解密；如果用一个私有密钥对数据进行加密，那么也只有用其对应的公开密钥才能解密。
  - 算法的加密过程和解密过程使用的是两个不同的密钥，所以称之为非对称加密算法。
  - 非对称加密算法消除了最终用户交换密钥的需要，具有比较好的保密性。

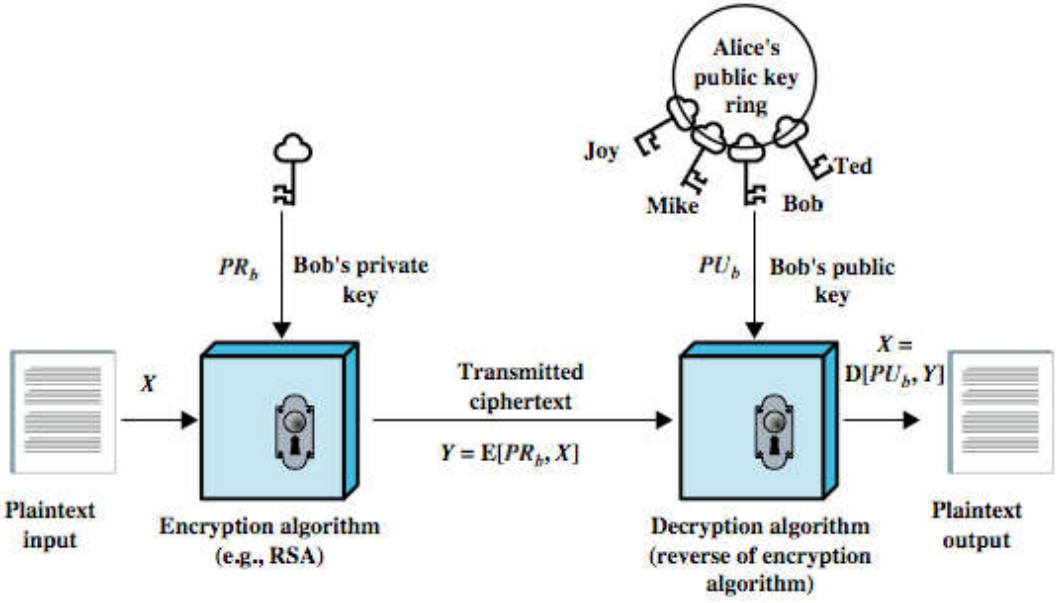
1. Introduction

- 公钥密码学算法的基本过程
  - 假设 *Bob* 向 *Alice* 发起一次加密通信。
    - ✧ 用于数据加密：
      - *Bob* 使用 *Alice* 的公钥加密明文信息，将得到的密文信息 (通过不安全渠道) 发送给 *Alice*；
      - *Alice* 用自己的私钥对密文解密，得到原始明文信息。



1. Introduction

- 公钥密码学算法的基本过程
  - 假设 *Bob* 向 *Alice* 发起一次加密通信。
    - ✧ 用于数字签名：
      - ◌ *Bob* 使用自己的私钥加密明文信息，将密文发送给 *Alice*；
      - ◌ *Alice* 使用 *Bob* 发布的公钥对密文解密，确信得到的原始明文信息是由 *Bob* 的私钥加密的。



1. Introduction

- 公钥密码学算法的安全性
  - 对称密码体制中只有一种密钥，并且是非公开的，双方需要共享密钥才能实现加解密，因此也称为私钥密码体制。共享方的可靠性和密钥传递的安全性对加密安全性有重大影响。
  - 非对称密码体制的安全性依赖于其算法与密钥。双钥制加解密机制不需要像对称密码技术那样传输对方的私有密钥，安全性得以大大增强。
  - 算法的复杂程度以及密钥长度的增加使得非对称密码体制的加密解密速度相对于对称密码体制要慢得多。

1. Introduction

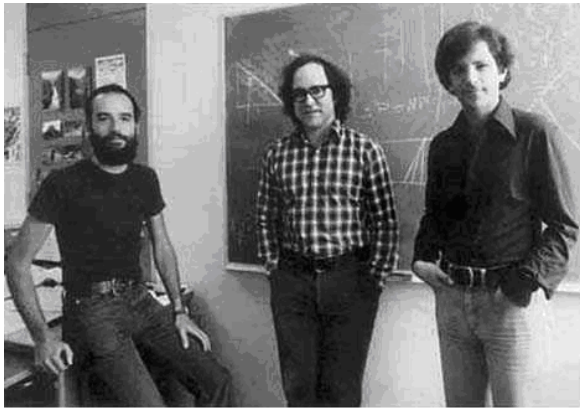
- **Asymmetric Key Cryptographic Requirements**
  - Computationally easy to create key pairs.
  - Computationally easy for sender knowing public key to encrypt messages.
  - Computationally easy for receiver knowing private key to decrypt cipher text.
  - Computationally infeasible for opponent to determine private key from public key.
  - Computationally infeasible for opponent to otherwise recover original message.

1. Introduction

- **Asymmetric Key Cryptographic Requirements**
  - 一些讨论
    - ✧ 公钥和私钥必须相关，而且从公钥到私钥不可推断。
      - ◌ 必须要找到一个难题，从一个方向走是容易的，从另一个方向走是困难的；然后把这个困难问题跟加解密结合起来
      - ◌ 找到一个困难问题不一定能产生一个保密性很好的密码系统吗，还需要有适当的构造方法。
    - ✧ 计算复杂性理论的指导。
      - ◌ 计算可行和计算不可行的界限。
      - ◌ 计算复杂性通常针对一个孤立的问题进行研究，并主要考虑最坏的情形。
      - ◌ 公钥密码学往往需要考虑一些相关的问题，比如密码分析还需要考虑已知明文、选择明文等相关的情形。

1. Introduction

- **Asymmetric Key Cryptographic Algorithms**
  - **RSA**
    - ✧ Developed in 1977 by *Rivest, Shamir* and *Adleman.*
    - ✧ Only widely accepted public-key encryption algorithm.
    - ✧ Given tech advances need 1024+ bit keys.



  - **Digital Signature Standard (DSS)**
    - ✧ Provides only a digital signature function with SHA-1.
  - **Elliptic Curve Cryptography (ECC)**
    - ✧ New, security like RSA, but with much smaller keys.

2. Knapsack Problem and *MH* Algorithm

2.3.2 Knapsack Problem and *MH* Algorithm

- **Knapsack Problem (*Tobias Dantzig*, 1897)**
  - 背包问题：有一个容积为 S 的背包和体积分别为 $a_1, a_2, \ldots, a_n$ 的 n 个物品，已知从这些物品中选出若干个刚好能够装满这个背包，求一个选择的方案。
  - 背包问题：有一个容积为 S 的空水桶和容积分别为 $a_1, a_2, \ldots, a_n$ 的 n 个装满水的杯子，已知从这些杯子中选出若干个，它们装的水刚好能够倒满水桶，求一个选择的方案。
  - 一般的背包问题是一个 NP 完全问题，解决问题所需要的时间与 n 呈指数增长。
    - ✧ 相当于含有 n 个原子的逻辑表达式的解释数目。

2. Knapsack Problem and *MH* Algorithm

- **Knapsack Problem**
  - 0-1 背包问题
    - ✧ 给定一个正整数 S 和一个 $A = (a_1, a_2, \ldots, a_n)$，其中 a_i 是正整数，求满足方程

      $$S = \sum a_i x_i \quad (i = 1 \ldots n).$$

      的二进制向量 $X = (x_1, x_2, \ldots, x_n)$。
      - ◌ $A = (a_1, a_2, \ldots, a_n)$ 称为背包向量。
      - ◌ $x_i \in \{0, 1\}, i = 1 \ldots n$.
      - ◌ 通常以背包向量 A 为密钥，正整数 S 为密文，二进制向量 X 为明文。
      - ◌ 问题有可能无解。

2. Knapsack Problem and *MH* Algorithm

- **Knapsack Problem**
  - 背包问题用于公钥密码学
    - ✧ 背包向量的结构特征决定了两类背包问题：
      - ◌ 简单背包：可以在线性时间内求解，也称易解背包；
      - ◌ 难解背包：不一定能在线性时间内求解的非简单背包，即一般背包。
    - ✧ 基于背包问题的公钥密码系统的设计
      - ◌ 以 X 为明文、S 为密文，设计一个易解的背包向量作为私钥，再把易解的背包问题转换成难解的背包问题，将构造得到的难解背包问题的背包向量作为公开密钥。
    - ✧ 背包问题用于公钥密码学，构成了第一个公钥密码系统。但在实践过程中，大多数的背包方案已被破解，或者被证明存在缺陷。

2. Knapsack Problem and *MH* Algorithm

- **Knapsack Problem**
  - 易解的背包问题—超递增背包
    - ✧ 设有背包 $A = (a_1, a_2, \ldots, a_n)$。如果对所有的 i, $i = 1 .. n$ 都有

      $a_i > \sum a_j (j = 1, \ldots, i\text{-}1)$.

      这类背包称为超递增背包，是一类简单背包。
    - ✧ 超递增背包的求解
      - ◌ 已知密文正整数 S 和作为解密密钥的超递增背包 $A = (a_1, a_2, \ldots, a_n)$，求明文 $X = (x_1, x_2, \ldots, x_n)$。

        (1) $i = n$.

        (2) If $i = 0$ failed.

        (3) If $S > a_i$ then $x_i = 1$, $S = S - a_i$; else $x_i = 0$. $i = i - 1$.

        (4) If $S = 0$ return ; else goto (2).

2. Knapsack Problem and *MH* Algorithm

- **Knapsack Problem**
  - 易解的背包问题—超递增背包
    - ✧ 超递增背包的求解
      - ◌ 例：简单背包 {2, 3, 6, 13, 27, 52}，求解 S = 70 的背包问题。

i	a	S	x
6	52	70	1
5	27	18	0
4	13	18	1
3	6	5	0
2	3	5	1
1	2	2	1
		0	

      - ◌ 结果为 $X = (x_1, x_2, x_3, x_4, x_5, x_6) = (1, 1, 0, 1, 0, 1)$

2. Knapsack Problem and *MH* Algorithm

- ***MH* 公钥密码算法**
  - *Ralph Merkle, Martin Hellman*, 1978
    - ✧ *MH* 算法是一个基于背包问题的公钥密码系统。
    - ✧ 1982年 *Adi Shamir* 破解了使用一次循环的 *MH* 背包密码。
    - ✧ *MH* 背包算法始终未能付诸实用。
  - 私钥：选择一个超递增背包 $A = (a_1, a_2, \ldots, a_n)$ 做为私钥。
  - 公钥：基于私钥背包 $A = (a_1, a_2, \ldots, a_n)$ 产生相应的公钥。
    - ✧ 选择一个整数 $m > \sum a_i$, $i = 1 .. n$;
    - ✧ 选择一个与 m 互素的整数 w，构造

      $$a_i' = wa_i \bmod m, \ i = 1 .. n.$$

      得到一个非超递增背包 $A' = (a_1', a_2', \ldots, a_n')$；这里的 a_i' 是伪随机分布的;
    - ✧ 把 $A' = (a_1', a_2', \ldots, a_n')$ 作为公钥发布。

2. Knapsack Problem and *MH* Algorithm

- ***MH* 公钥密码算法**
  - 加密
    - ✧ 将明文分组。设有长度为 n 位的明文二进制块

      $X = (x_1, x_2, \ldots, x_n)$。
    - ✧ 利用公钥 $A' = (a_1', a_2', \ldots, a_n')$ 将明文 X 变为密文 S

      $S = E_{A'}(X) = \sum(a_i' x_i),\ i = 1 \ldots n.$
  - 解密
    - ✧ 先计算 $S' = w^{-1}S \bmod m$，其中 w^{-1} 是 w 的模 m 逆元。
    - ✧ 再利用私钥 $A = (a_1, a_2, \ldots, a_n)$ 求解关于 S' 的超递增背包问题：

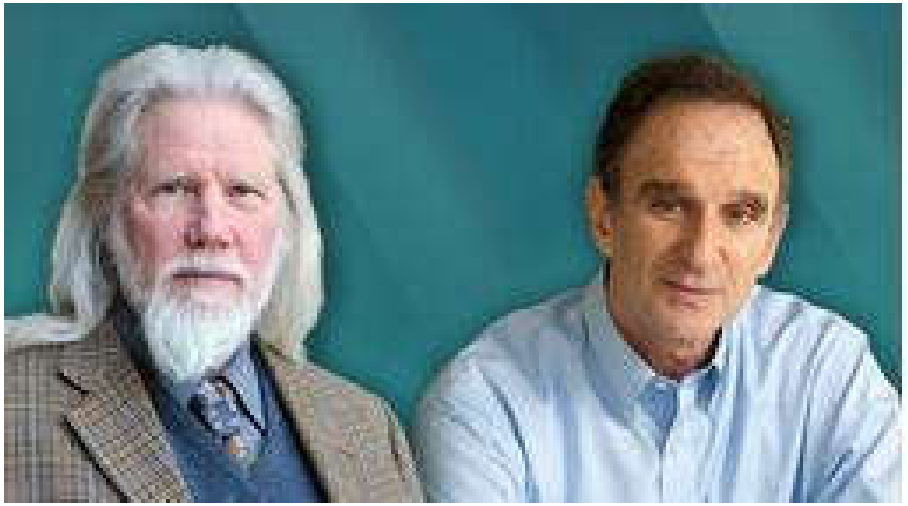
      $S' = \sum(a_i x_i),\ i = 1 \ldots n.$

      得到的解 $X = (x_1, x_2, \ldots, x_n)$ 就是恢复的明文。
  - 公钥和私钥配对才能进行加解密，否则构造的背包问题无解。
  - 思考：证明 *MH* 算法的有效性。

3. *Diffie-Hellman* Key Exchange Algorithm

2.4.3 *Diffie-Hellman* Key Exchange Algorithm

- **What is *Diffie-Hellman* Key Exchange Algorithm**
  - Developed in 1976 by *Whitfield Diffie* and *Martin Hellman.*
  - Only allows exchange of a secret key.
  - *Diffie-Hellman* 密钥交换算法：加密和解密可以使用不同的规则，只要这两种规则之间存在某种对应关系，即可在不直接传递密钥的情况下完成解密，从而回避了密钥直接传递过程的风险。



3. *Diffie-Hellman* Key Exchange Algorithm

- ***Diffie-Hellman* Key Exchange Algorithm**
  - *W. Deffie* 和 *M. Hellman* 于1976年首次提出基于求解离散对数问题困难性的密钥交换体制，简称 *Diffie-Hellman* 密钥交换协议。
  - *Alice* 和 *Bob* 使用 *Diffie-Hellman* 密钥交换协议，在一个不安全的信道上交换密钥，其操作步骤如下：

    (1) *Alice* 和 *Bob* 商定一个素数 p 和 p 的一个素根 g，p, g 可以公开；

    (2) *Alice* 秘密选取正整数 $A<p$，计算 $a = g^A \bmod p$ 并发送给 *Bob*；

    (3) *Bob* 秘密选取正整数 $B<p$，计算 $b = g^B \bmod p$ 并发送给 *Alice*；

    (4) *Alice* 计算 $(b)^A \equiv (g^B)^A \equiv g^{BA} \pmod p$；

    (5) *Bob* 计算 $(a)^B \equiv (g^A)^B \equiv g^{AB} \pmod p$；

    (6) *Alice* 和 *Bob* 双方获得一个共享密钥 $K = g^{AB} \pmod p$。
  - 上述的 a 和 b 称为 *Diffie-Hellman* 公开参数。当 p 取的足够大时，从 a, g 和 p 取得 A，或者从 b, g 和 p 取得 B 是计算不可行的。目前主要采用的三组 DH 公开参数的位数分别是 768、1024 和 1536。
  - 私钥 A 和 B 的取值范围建议在 0 到 p-2 之间。

3. *Diffie-Hellman* Key Exchange Algorithm

- ***Diffie-Hellman* Key Exchange Algorithm**
  - *Alice* 和 *Bob* 使用 *Diffie-Hellman* 密钥交换协议，在一个不安全的信道上交换密钥。

Diffie-Hellman Key Exchange

Step	Alice	Bob
1	Parameters: p, g	
2	$A = \text{random}()$ $a = g^A \pmod p$	$\text{random}() = B$ $g^B \pmod p = b$
3	$a \longrightarrow$ $\longleftarrow b$	
4	$K = g^{BA} \pmod p = b^A \pmod p$	$a^B \pmod p = g^{AB} \pmod p = K$
5	$\longleftarrow E_K(data) \longrightarrow$	

  - 大素数 *p* 可以由 *miller-Robin* 算法生成，在 *p*-2 内随机测试原根 *g*。
  - Ref. to “PKCS#3: *Diffie-Hellman* Key Agreement Standard”, RSA Lab.

3. *Diffie-Hellman* Key Exchange Algorithm

- ***Diffie-Hellman* Key Exchange Algorithm**
  - *Diffie-Hellman* 密钥交换协议的一个多用户应用场景。
    - ✧ 一组用户 $\{A_1, A_2, \ldots, A_n\}$，用户 A_i 有私钥 X_i 和公钥 Y_i。所有公钥 Y_i (i=1, 2, …, n) 连同一个协商好的素数 p 和 p 的一个素根 g 作为 *Diffie-Hellman* 公开值存储在 CA 服务器。A_i 随时可以通过 CA 访问 A_j 存储的公钥 Y_j，计算一个秘密密钥 $K = g^{X_i Y_i} \pmod p$，并使用密钥 K 发送一个加密报文给 A_j。A_j 利用 A_i 存储在 CA 的公钥 Y_i 计算出同一个密钥 K，并使用密钥 K 解密该报文。
    - ✧ 在 CA 可信任的前提下，上述流程提供了保密性和一定程度的认证功能。
      - ○ 保密性：只有 A_i 和 A_j 可以确定对称密钥 K，其它用户无法生成密钥 K 因而无法解读报文；
      - ○ 认证性：根据 CA 提供的保证，接收方 A_j 确信只有用户 A_i 才能使用对称密钥 K 生成收到的加密报文。

3. *Diffie-Hellman* Key Exchange Algorithm

- ***Diffie-Hellman* Key Exchange Algorithm**
  - *Diffie-Hellman* 密钥交换协议的中间人攻击：*Diffie-Hellman* 算法不能验证对方身份，因而在不安全的信道上无法防御中间人攻击，需要增加数字签名。例如 *D* 发起针对 *A* 和 *B* 的中间人攻击：

    (1) *Alice* 和 *Bob* 商定一个素数 p 和 p 的一个素根 g，p, g 可以公开；

    (2) *Alice* 秘密选取正整数 $A<p$，计算 $a = g^A \bmod p$ 并发送给 *Bob*；

    (2)′ *Darth* 选取正整数 $D<p$，计算 $d = g^D \bmod p$；

    (2)″ *Darth* 截获 a，冒充 *Alice* 发送 d 给 *Bob*；

    (3) *Bob* 秘密选取正整数 $B<p$，计算 $b = g^B \bmod p$ 并发送给 *Alice*；

    (3)′ *Darth* 截获 b，冒充 *Bob* 发送 d 给 *Alice*；

    (4) *Alice* 计算 $(d)^A \equiv (g^D)^A \equiv g^{DA} \pmod p$；

    (5) *Bob* 计算 $(d)^B \equiv (g^D)^B \equiv g^{DB} \pmod p$；

    (5)′ *Darth* 计算: $(a)^D \equiv g^{AD} \pmod p$，$(b)^D \equiv g^{BD} \pmod p$；

    (6) *D* 与 *A*、*B* 双方分别建立了共享密钥通信。
  - *D* 必须实时截获 a 和 b 并马上冒充转发，否则会被发现。

3. *Diffie-Hellman* Key Exchange Algorithm

- ***Diffie-Hellman* Key Exchange Algorithm**
  - *Diffie-Hellman* 算法的主要优势：
    - ✧ 仅当需要时才生成密钥，降低了密钥存储的风险；
    - ✧ 事先约定公开值，密钥交换不需要基础设施的支持。
  - *Diffie-Hellman* 算法的主要不足：
    - ✧ 不具备认证能力；
    - ✧ 计算密集型过程需要计算资源的支持，容易遭受阻塞性攻击；
    - ✧ 不能防止重放攻击；
    - ✧ 容易遭受中间人的攻击。
  - 优化后的 Oakley 算法：
    - ✧ 采用 cookie 机制对抗阻塞攻击；
    - ✧ 双方协商一个全局参数集合；
    - ✧ 使用时间戳抵抗重放攻击；
    - ✧ 交换 *Diffie-Hellman* 公开密钥；
    - ✧ 增加数字签名等认证手段以对抗中间人攻击。

4. RSA Algorithm

2.4.4 RSA Algorithm

- **RSA 算法的起源**
  - RSA 算法在1977年由 MIT 的 *Ron Rivest*、*Adi Shamir* 和 *Leonard Adleman* 一起提出，并以他们三人姓氏开头字母命名，是一种获得广泛使用的非对称加密算法。
  - 1983年麻省理工学院在美国为 RSA 算法申请了专利。这个专利2000年9月21日失效。由于该算法在申请专利前就已经发表，在世界上大多数其它地区这个专利权不被承认。

4. RSA Algorithm

- **RSA 算法的安全性概述**
  - 对极大整数进行因数分解的难度 (*The Factoring Problem*) 决定了 RSA 算法的可靠性。换言之，对一个极大整数做因数分解愈困难，RSA 算法就愈可靠。假如有人找到一种快速因数分解的算法的话，那么用 RSA 加密的信息的可靠性就肯定会极度下降。目前看来找到这样的算法的可能性非常小。
  - 目前还没有可靠的攻击 RSA 算法的方式。短的 RSA 钥匙可能被强力方式解破 (比如 768-bit，即232个十进制位以下的整数)。只要其钥匙的长度足够长 (比如 1024-bit 至 15360-bit)，用 RSA 加密的信息看来很难被破解。
  - 在分布式计算技术和量子计算理论日趋成熟的今天，RSA 加密的安全性受到了挑战。

4. RSA Algorithm

- **How RSA Works**
  - *Alice* is generating her Public key and her Private key.
    - ✧ Select two different, big primes p and q. Let $N=pq$.
    - ✧ Compute *Euler*'s φ function $\varphi(N)$ By :

      $$\varphi(N) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1).$$

    - ✧ Select an integer e such that $1< e < \varphi(N)$ and $\gcd(e, \varphi(N)) =1$.
    - ✧ Compute d by $de \equiv 1 \pmod{\varphi(N)}$.
      - ◌ d is an inverse of e modulo $\varphi(N)$.
      - ◌ Apply Ext_*Euclidean*_Algorithm to get d.
      - ◌ d is positive and need to be big enough.
    - ✧ Destroy p and q (when N is big enough, it will be extremely difficult to find p and q from N).
    - ✧ (e, N) is used as the Public key, and (d, N) the Private key. *Alice* will send her (e, N) to *Bob* or anyone, but keep her (d, N) in secret.

4. RSA Algorithm

- **How RSA Works**
  - *Alice* is generating her Public key and her Private key.
    - ✧ Select two different, big primes p and q. Let $N=pq$.
    - ✧ Compute *Euler*'s φ function $\varphi(N)$ By :

      $$\varphi(N) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1).$$

    - ✧ Select an integer e such that $1< e < \varphi(N)$ and $\gcd(e, \varphi(N)) =1$.
    - ✧ Compute d by $de \equiv 1 \pmod{\varphi(N)}$.
      - ◌ d is an inverse of e modulo $\varphi(N)$.
      - ◌ Apply Ext_*Euclidean*_Algorithm to get d.
      - ◌ d is positive and need to be big enough.
    - ✧ Destroy p and q (when N is big enough, it will be extremely difficult to find p and q from N).
    - ✧ (e, N) is used as the Public key, and (d, N) the Private key. *Alice* will send her (e, N) to *Bob* or anyone, but keep her (d, N) in secret.

The moduli is $\varphi(N)$, NOT N.

4. RSA Algorithm

- **How RSA Works**
  - Encryption
    - ✧ Suppose that *Bob* want to send *Alice* a message M. *Bob* has *Alice*'s Public key (e, N) in hand. Firstly *Bob* transforms M to an integer n less than N in a presumed method. By applying the key (e, N), n is encrypted to c:

      $$c = n^e \bmod N.$$

      Then *Bob* can send c to Alice through an unsafe channel.
  - Decryption
    - ✧ By applying her Private key (d, N), *Alice* can decode n' from c by

      $$n' = c^d \bmod N.$$

      and recover the M from n' in the same presumed method.

4. RSA Algorithm

- **How RSA Works**
  - Principle of RSA
    - ✧ By the *Corollary* of *Euler's Theorem* on φ function, given two primes p and q satisfied $N=pq$, one integer n satisfied $0<n<N$, and any positive k, we have $n^{k\varphi(N)+1} \equiv n \pmod{N}$.
    - ✧ In another hand, d was generated from

      $de \equiv 1 \pmod{\varphi(N)}$, that is

      $de = k\varphi(N)+1$ for some positive k.
    - ✧ Then $n^{de} \equiv n \pmod{N}$, or $n^{de} \bmod N = n \bmod N$.
    - ✧ Now we have

      $c = n^e \bmod N$, $n' = c^d \bmod N$.
    - ✧ Applying the rules of modular arithmetic, we have

      $n' = c^d \bmod N = (n^e)^d \bmod N = n \bmod N$.
    - ✧ That is $n' \equiv n \bmod N$.
    - ✧ In practice, RSA encryption is combined with zero padding, salt, and message hash functions to securely transmit messages.

4. RSA Algorithm

- **How RSA Works**
  - Principle of RSA
    - ✧ By the *Corollary* of *Euler's Theorem* on φ function, given two primes p and q satisfied $N=pq$, one integer n satisfied $0<n<N$, and any positive k, we have $n^{k\varphi(N)+1} \equiv n \pmod{N}$.
    - ✧ In another hand, d was generated from
      $de \equiv 1 \pmod{\varphi(N)}$, that is
      $de = k\varphi(N)+1$ for some positive k.
    - ✧ Then $n^{de} \equiv n \pmod{N}$, or $n^{de} \bmod N = n \bmod N$.
    - ✧ Now we have
      $c = n^e \bmod N$, $n' = c^d \bmod N$.
    - ✧ Applying the rules of modular arithmetic, we have
      $n' = c^d \bmod N = (n^e)^d \bmod N = n \bmod N$.
    - ✧ That is $n' \equiv n \bmod N$.
    - ✧ In practice, RSA encryption is combined with zero padding, salt, and message hash functions to securely transmit messages.

The length of plaintext, say n, is limited by N.

4. RSA Algorithm

- **How RSA Works**
  - Principle of RSA
    - ✧ By the *Corollary* of *Euler's Theorem* on φ function, given two primes p and q satisfied $N=pq$, one integer n satisfied $0<n<N$, and any positive k, we have $n^{k\varphi(N)+1} \equiv n \pmod{N}$.
    - ✧ In another hand, d was generated from
      $de \equiv 1 \pmod{\varphi(N)}$, that is
      $de = k\varphi(N)+1$ for some positive k.
    - ✧ Then $n^{de} \equiv n \pmod{N}$, or $n^{de} \bmod N = n \bmod N$.
    - ✧ Now we have
      $c = n^e \bmod N$, $n' = c^d \bmod N$.
    - ✧ Applying the rules of modular arithmetic, we have
      $n' = c^d \bmod N = (n^e)^d \bmod N = n \bmod N$.
    - ✧ That is $n' \equiv n \bmod N$.
    - ✧ In practice, RSA encryption is combined with zero padding, salt, and message hash functions to securely transmit messages.

The length of ciphertext, say c, is also limited by N.

4. RSA Algorithm

- **How RSA Works**
  - Padding of Input Message
    - ✧ Ref. to RFC 8017 (pkcs-1 v2.2): 7.2 RSAES-PKCS1-v1_5
    - ✧ EME-PKCS1-v1_5 encoding with Public Key:
      - ◌ Let *k* be the length in octets (字节数) of the RSA modulus *N*, *M* be the message to be encrypted, an octet string (字节串) of length *mLen*, where *mLen* ≤ *k*-11.
      - ◌ Generate an octet padding string *PS* of length *k*-*mLen*-3 consisting of pseudo-randomly generated nonzero octets. The length of *PS* will be at least eight octets, which is a security condition for public-key operations that makes it difficult for an attacker to recover data by trying all possible encryption blocks.
      - ◌ Concatenate *PS*, the message *M*, and other padding to form an encoded message *EM* of length *k* octets as

$$EM = 0x00 \,||\, 0x02 \,||\, PS \,||\, 0x00 \,||\, M.$$

4. RSA Algorithm

- **How RSA Works**
  - Padding of Input Message
    - ✧ EME-PKCS1-v1_5 decoding with Private Key:
      - ◌ Separate the encoded message *EM* into an octet string *PS* consisting of nonzero octets and a message *M* as

        EM = 0x00 || 0x02 || *PS* || 0x00 || *M*.
      - ◌ If the first octet of *EM* does not have hexadecimal value 0x00, if the second octet of *EM* does not have hexadecimal value 0x02, if there is no octet with hexadecimal value 0x00 to separate *PS* from *M*, or if the length of *PS* is less than 8 octets, output "decryption error" and stop.
      - ◌ Output *M*.
    - ✧ The second octet of *EM* is defined as
      - ◌ 02 - Encryption with Public Key, *PS* consisting of pseudo-randomly generated nonzero octets.
      - ◌ 01 - Encryption with Private Key, *PS* consisting of 0xFF.
      - ◌ 00 - Encryption with Private Key, *PS* consisting of 0x00.

4. RSA Algorithm

- **RSA 算法原理**
  - 公钥和私钥的产生
    - ✧ 选择两个不同的大素数 p 和 q，计算 $N=pq$。
    - ✧ 引用欧拉 φ 函数，小于 N 且与 N 互素的正整数个数为

      $$\varphi(N) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1).$$

    - ✧ 选择一个整数 e，$1< e < \varphi(N)$ 且 $\gcd(e, \varphi(N)) =1$.
    - ✧ 求一个满足下列同余式的 d：

      $$ed \equiv 1 \pmod{\varphi(N)}.$$

      - ◌ d 就是 e 的模 $\varphi(N)$ 逆元，可以应用扩展欧几里德算法得到。
      - ◌ d 必须是一个足够大的正整数。
    - ✧ 将 p 和 q 的记录销毁 (当 N 足够大时，寻找 p, q 将极其困难)
    - ✧ 以 (e, N) 作为公钥，(d, N) 作为私钥。假设 *Alice* 想通过一个不可靠的媒体接收 *Bob* 的一条私人讯息，*Alice* 将她的公钥 (e, N) 公开给 *Bob*，而将她的私钥 (d, N) 严密收藏起来。

4. RSA Algorithm

- **RSA 算法原理**
  - 加密消息
    - ✧ 现在 *Bob* 想给 *Alice* 送一个消息 M，他知道 *Alice* 产生的公钥 (e, N)。
    - ✧ *Bob* 使用事先与 *Alice* 约好的格式将 M 转换为一个小于 N 的整数 n，比如他可以将每一个字符转换为其数字形式的 Unicode 码，然后将这些数字连在一起组成整数 n (假如他的信息非常长的话，他可以将这个信息分为几段，逐段加密)。
    - ✧ *Bob* 引用公钥 (e, N)，利用下面的公式，将 n 加密为 c：

      $$c = n^e \bmod N$$

    - ✧ *Bob* 算出 c 后可以将它经公开媒体传递给 *Alice*。

4. RSA Algorithm

- **RSA 算法原理**
  - 解密消息
    - ✧ *Alice* 得到 *Bob* 的消息 c 后利用她的私钥 (d, N) 解码。
    - ✧ *Alice* 利用以下的公式将 c 转换为 n'：

      $$n' = c^d \bmod N$$

    - ✧ *Alice* 得到的 n' 就是 *Bob* 的 n，因此可以将原来的信息 M 准确复原。

4. RSA Algorithm

- **RSA 算法原理**
  - 解码原理
    - ✧ 欧拉定理
      - ◌ 对互素的 a 和 m，有 $a^{\varphi(m)} \equiv 1 \pmod{m}$
    - ✧ 推论
      - ◌ 给定满足 $N = pq$ 的两个不同素数 p 和 q 以及满足 $0<n<N$ 的整数 n，k 是正整数，有 $n^{k\varphi(N)+1} \equiv n \pmod{N}$.
    - ✧ 由于 $ed \equiv 1 \pmod{\varphi(N)}$，即 $ed = k\varphi(N)+1$，故

      $n^{ed} = n^{k\varphi(N)+1} \equiv n \pmod{N}$，即 $n^{ed} \bmod N = n \bmod N$.

      现在我们有

      $c = n^e \bmod N$, $n' = c^d \bmod N$.

      应用模算术运算规则得到

      $n' = c^d \bmod N = (n^e)^d \bmod N = n \bmod N$.

      即为 $n' \equiv n \bmod N$.

4. RSA Algorithm

- ***Example.***
  - For continent, we select small p, q and e.
  - Suppose *Alice* want to sent a string "key" to *Bob* by RSA encryption.

    (1) Compute (e, N) and (d, N)
    - ✧ Let $p = 3$, $q = 11$, $N = pq = 3 \times 11 = 33$

      $f(N) = (p-1)(q-1) = 2 \times 10 = 20$
    - ✧ We take $e = 3$ (3 is prime to 20) and try to get d such that

      $ed \equiv 1 \bmod f(n)$, that is $3d \equiv 1 \bmod 20$
    - ✧ We get d by trial: (or by the *Extended Euclidean Algorithm*)

4. RSA Algorithm

- ***Example.***
  - Suppose *Alice* want to sent a string "key" to *Bob* by RSA encryption.

    (1) Compute (e, N) and (d, N)
    - From the table, we find that when d=**7**, $ed \equiv 1 \bmod f(n)$ holds.
    - So the public key and the private key are as

      $K_U = (e, N) = (3, 33)$.

      and $K_R = (d, N) = (7, 33)$.

d	$e \times d$	$(e \times d) \bmod f(N)$
1	3	3
2	6	6
3	9	9
4	12	12
5	15	15
6	18	18
7	**21**	**1**
8	24	4
9	27	7

4. RSA Algorithm

- ***Example.***
  - Suppose *Alice* want to sent a string "key" to *Bob* by RSA encryption.
    (2) Digitization

a	b	c	d	e	f	g	h	i	j	k	l	m
01	02	03	04	05	06	07	08	09	10	11	12	13
n	o	p	q	r	d	t	u	v	e	x	y	z
14	15	16	17	18	19	20	21	22	23	24	25	26

Code Table for message transcoding

- By the Code Table, the code values of the string "key" are
  M_1=11, M_2=05 and M_3=25.
- Note that the max value of any code is 26 (it can not over N=33).

4. RSA Algorithm

- ***Example.***
  - Suppose *Alice* want to sent a string "key" to *Bob* by RSA encryption.

    (3) Encryption

    ✧ By $M^e = C \pmod N$ we have $C = M^e \pmod N$

    That is $C_1 = 11^3 \bmod 33 = 11$

    $C_2 = 5^3 \bmod 33 = 26$

    $C_3 = 25^3 \bmod 33 = 16$

    (4) Decryption

    ✧ By $C^d = M \pmod N$ we have $M = C^d \pmod N$

    That is $M_1 = 11^7 \bmod 33 = 11$

    $M_2 = 26^7 \bmod 33 = 5$

    $M_3 = 16^7 \bmod 33 = 25$

    (5) Message recovery

    ✧ By checking the Code Table, the string "key" is recovered.

4. RSA Algorithm

- **RSA 算法的可靠性**
  - 算法的敏感数据分析
    - $N = pq$.
    - $\varphi(N) = (p-1)(q-1)$.
    - p, q 被销毁
    - (e, N) 作为公钥被公开
    - d 是 e 的模 $\varphi(N)$ 逆元：$de \equiv 1 \pmod{\varphi(N)}$.
  - 结论
    - 私钥被破解即 (d, N) 被推算出来的必要条件是 N 被因数分解。
    - 对极大整数做因数分解的难度决定了 RSA 算法的可靠性。对一极大整数做因数分解愈困难，RSA 算法愈可靠。
    - 极大整数的因数分解非常困难。目前除了暴力破解，还没有发现其它有效方法。

4. RSA Algorithm

- **RSA 算法的可靠性**
  - 针对 RSA 算法的攻击
    - ✧ 基于数学的攻击
      - ◌ 分解 $N = pq \Rightarrow \varphi(N) = (p-1)(q-1) \Rightarrow d = e^{-1} \pmod{\varphi(N)}$。
      - ◌ 不求出 p, q，直接求 $\varphi(N) \Rightarrow d = e^{-1} \pmod{\varphi(N)}$。
      - ◌ 不求出 $\varphi(N)$，直接计算 d。
    - ✧ 同模攻击 Common Modulus Attack
      - ◌ 两对密钥的模 N 相同。
    - ✧ 计时攻击 Timing Attack
      - ◌ 利用 CPU 处理某些特殊的模乘比较慢的规律来确定每一位指数。

4. RSA Algorithm

- **RSA 算法中的大素数选择问题**
  - RSA 算法中 p 和 q 选择的一些基本原则
    - ✧ (*D. Coppersmith*) p 和 q 不能离得太近。如果 N 的位数为 k，那么 $|p - q|$ 要同时满足 $\log|p - q| > k/2\text{-}100$ 以及 $\log|p - q| > k/3$。
    - ✧ (*D. Coppersmith*) 较短的 e 可以提高加密算法计算 n^e 的速度，但可能存在计算 d 的快速算法。PKCS#1 建议 $e = 2^{16}+1 = 65537$。
    - ✧ (*M. Wiener*) d 不能太小，否则可以通过快速算法计算得到 d。如果 N 的位数为 k，那么 d 的值要满足 $d > 2^{k/2}$
    - ✧ (*O. Schirokauer*) N 的非相邻形式 (Non-Adjacent Form, *NAF*) 的海明权重 (*Hamming* weight, 非0元总数) 不能太小，否则应用数筛法可能会快速对 N 进行质因数分解。一般要求 N 的 *NAF* 表述权重大于 $N/4$。

4. RSA Algorithm

- **RSA 算法中的大素数选择问题**
  - RSA 算法中 p 和 q 的选择流程
    - (1) 确定 RSA 所要求 N 的位数 k。 k = 1024、2048、3072、4096 ...
    - (2) 随机选择一个位数为 $(k+1)/2$ 的素数 p。
      - 即 $p \in [2^{(k+1)/2-1}, 2^{(k+1)/2} - 1]$.
    - (3) 选择一个位数为 $k - (k+1)/2 = (k-1)/2$ 的素数 q。
    - (4) 求 $|p-q|$；如果 $\log|p-q|$ 过小，则返回 (2)，重新选取 p。
    - (5) 计算 $N = pq$，确认 N 的位数为 k ($N \in [2^{k-1}, 2^k - 1]$)；否则返回 (2)，重新选取 p。
    - (6) 计算 N 的 *NAF* 权重；如果权重过小，则返回 (2)，重新选取 p。
    - (7) 计算 $\varphi(N)$，选择公钥 e， $2^{16} < e < \varphi(N)$ 且 $\gcd(e, \varphi(N)) = 1$。
      - 一般建议选择素数 $e = 2^{16}+1 = 65537$。
    - (8) 求 e 的模 $\varphi(N)$ 逆元 d；如果 d 过小，则返回 (2)，重新选取 p。
    - (9) 返回 RSA 的参数 p、q、e、d。
      - 或者销毁 p、q，返回 N、e、d。

5. Generating Big Primes

2.4.5 大素数的生成

- 素数的存在性
  - 素数理论：在正整数 *N* 附近，每 *ln*(*N*) 个整数中有一个素数。
- 素数的生成过程

(1) 随机选择一个奇数 *n* (比如通过伪随机数发生器)；

(2) 随机选择 *a*, 使 0<*a*<*n*；

(3) 进行素性测试 (例如用 *Miller-Rabin* 算法)，若 *n* 没有通过测试，抛弃 *n*，转到 (1)；

(4) 如果通过了足够次数的测试，概率上可以认为 *n* 是素数，算法结束；否则转 (2)。

5. Generating Big Primes

- ***Miller-Rabin* Primality Test**
  - *Fermat*'s Little Theorem
    - If p is a prime number and a is an integer not divisible by p, then $a^{p-1} \equiv 1 \pmod{p}$.
  - *Fermat* 测试
    - 假设需要测试 n 是否为素数。随机选取 $0<a<n$，计算 $a^{n-1} \bmod n$，如果结果不为1，则可以断定 n 不是素数。
    - 如果结果为1则再随机选取一个新的 a 进行测试。如此反复多次，如果每次结果都是1，就可以假定 n 是素数。
    - *Fermat* 测试的结果不一定准确，有可能出现误判的情况。
    - *Miller* 和 *Rabin* 在 *Fermat* 测试的基础上，建立了 *Miller-Rabin* 素数测试算法 (1976)，增加了一个二次探测定理：
      - 如果 n 是奇素数，则 $a^2 \equiv 1 \bmod n$ 的解为 $a \equiv 1 \bmod n$ 或者 $a \equiv (n-1) \bmod n$。

5. Generating Big Primes

- ***Miller-Rabin* Primality Test**
  - *Miller-Rabin* 素数测试算法
    - ✧ *Fermat* 测试的一轮中，如果 $a^{n-1} \equiv 1 \bmod n$ 成立，*Miller-Rabin* 算法需要判定 n-1 是否偶数。如果 n-1 是偶数，令 $u = (n-1)/2$，检查是否满足二次探测定理，即

      $a^u \bmod n = 1$ 或者 $a^u \bmod n = n - 1$.

      如果上述结果不成立，则二次探测失败，n 不可能是素数。
    - ✧ 例：假设 n=341，选取 a=2。第一次测试时，$2^{340} \bmod 341 = 1$。由于340是偶数，因此检查 2^{170}，得到 $2^{170} \bmod 341 = 1$，满足二次探测定理。同时由于170还是偶数，因此进一步检查 $2^{85} \bmod 341 = 32$。此时不满足二次探测定理，可以判定341不是素数。
    - ✧ *Miller-Rabin* 算法每次测试失误的概率是1/4；进行 s 次测试后，失误的概率是 4^{-s}，计算复杂度 $O(s(\log n)^3)$。

6. RSA for Digital Signature

2.4.6 RSA for Digital Signature

- **Digital Signatures**
  - The RSA algorithm can be repurposed for digitally signing a message m
    - Public key $K_U = (e, N)$, private key $K_R = (d, N)$.
    - Signing: Compute
      $$S = \text{sign}(K_R, m) \equiv m^d \pmod{N}.$$
    - Verification: Compute $m' = S^e \bmod N$.
    - If $m' \equiv m \pmod{N}$, the signature is valid.

6. RSA for Digital Signature

- **Digital Signatures**
  - 传统签名的基本特点
    - ✧ 签名与被签的文件在物理上不可分割
    - ✧ 签名者不能否认自己的签名
    - ✧ 签名不能被伪造
    - ✧ 签名容易被验证
  - 数字签名是传统签名的数字化，基本要求是
    - ✧ 签名与被签的文件具有“绑定”机制
    - ✧ 签名者不能否认自己的签名
    - ✧ 签名不能被伪造
    - ✧ 签名容易被验证

6. RSA for Digital Signature

- **Digital Signatures**
  - RSA 签名原理
    - ✧ 假如 *Alice* 想给 *Bob* 传递一个署名的消息 *M*，她可以为 *M* 计算一个散列值 *h* (采用消息摘要 Message Digest 算法，比如 MD5)，然后用她的私钥 K_{AR} 加密 *h* 并将得到的 *H* 作为“署名”加在 *M* 的后面。*H* 只有用 *Alice* 的公钥 K_{AU} 才能被正确还原成 *h*。*Bob* 获得 *M* 及 *H* 后用 K_{AU} 解密 *H* 得到 *h'*，并采用相同的消息摘要算法从 *M* 得到 *h*。如果 *h* = *h'*，那么 *Bob*就可以认为发信人持有 *Alice* 的私钥 (据此认为发信人就是 *Alice*)，以及这个消息在传播路径上没有被篡改，从而得到完整性和不可抵赖性校验。
  - 问题：
    - ✧ 公钥的管理：如何有效确认公钥与发布者身份的对应关系?
    - ✧ 签名的有效性：能否及时处理私钥丢失导致的假冒?

End of Chapter 2.4