非对称加密

1. 对称加密的弊端'

- 秘钥分发困难
- 可以通过非对称加密完成秘钥的分发

> https
>
> Alice 和 Bob通信, Alice给bob发送数据, 使用对称加密的方式
>
> 1. 生成一个非对称的秘钥对, bob生成
> 2. bob将公钥发送给alice
> 3. alice生成一个用于对称加密的秘钥
> 4. alice使用bob的公钥就对称加密的秘钥进行加密, 并且发送给bob
> 5. bob使用私钥就数据解密, 得到对称加密的秘钥
> 6. 通信的双方使用写好的秘钥进行对称加密数据加密

2. 非对称加密的秘钥

- 不存在秘钥分发困难的问题

2.1 场景分析

数据对谁更重要, 谁就拿私钥

- 直观上看: 私钥比公钥长
- 使用第三方工具生成密钥对: 公钥文件xxx.pub xxx

> 1. 通信流程, 信息加密 （A写数据, 发送给B, 信息只允许B读）
>
>   A: 公钥
>
>   B: 私钥
> 2. 登录认证 （客户端要登录, 连接服务器, 向服务器请求个人数据）
>
>   客户端: 私钥
>
>   服务器: 公钥
> 3. 数字签名（表明信息没有受到伪造，确实是信息拥有者发出来的，附在信息原文的后面）
>   - 发送信息的人: 私钥
>   - 收到信息的人: 公钥
> 4. 网银U盾
>   - 个人: 私钥
>   - 银行拿公钥

3. 使用RSA非对称加密通信流程

> 要求: Alice 给 bob发送数据, 保证数据信息只有bob能看到

4. 生成RSA的秘钥对

4.1 一些概念

1. x509证书规范、 pem、 base64
    - pem编码规范 - 数据加密
    - base64 - 对数据编码, 可逆
        - 不管原始数据是什么, 将原始数据使用64个字符来替代
            - a-z A-Z 0-9 + /
2. ASN.1抽象语法标记
3. PKCS1标准

4.2 密钥对生成流程

- 生成私钥操作流程概述

> 1. 使用rsa中的GenerateKey方法生成私钥
>
>   func GenerateKey(random io.Reader, bits int) (priv *PrivateKey, err error)
>
>   - rand.Reader -> import "crypto/rand"
>   - 1024 的整数倍 - 建议
> 2. 通过x509标准将得到的ras私钥序列化为ASN.1 的 DER编码字符串
>
>   func MarshalPKCS1PrivateKey(key *rsa.PrivateKey) []byte
> 3. 将私钥字符串设置到pem格式块中
>
>   初始化一个pem.Block块
>
> ```
> type Block struct {
>     Type    string            // 得自前言的类型 (如"RSA PRIVATE KEY")
>     Headers map[string]string // 可选的头项
>     Bytes   []byte            // 内容解码后的数据, 一般是DER编码的ASN.1结构
> }
> ```
>
> 4. 通过pem将设置好的数据进行编码, 并写入磁盘文件中
>
>   func Encode(out io.Writer, b *Block) error
>
>   - out - 准备一个文件指针

- 生成公钥操作流程

> 1. 从得到的私钥对象中将公钥信息取出

```
type PrivateKey struct {
    PublicKey            // 公钥
    D         *big.Int   // 私有的指数
    Primes    []*big.Int // N的素因子，至少有两个
    // 包含预先计算好的值，可在某些情况下加速私钥的操作
    Precomputed PrecomputedValues
}
```

2. 通过x509标准将得到 的rsa公钥序列化为字符串

```
func MarshalPKIXPublicKey(pub interface{}) ([]byte, error)
```

3. 将公钥字符串设置到pem格式块中

   type Block struct { Type string // 得自前言的类型（如"RSA PRIVATE KEY"） Headers map[string]string // 可选的头项 Bytes []byte // 内容解码后的数据，一般是DER编码的ASN.1结构 }

4. 通过pem将设置好的数据进行编码, 并写入磁盘文件

   func Encode(out io.Writer, b *Block) error

5. RSA加解密

5.1 RSA加密

1. 将公钥文件中的公钥读出, 得到使用pem编码的字符串

   -- 读文件

2. 将得到的字符串解码

   -- pem.Decode

3. 使用x509将编码之后的公钥解析出来

   -- func ParsePKCS1PrivateKey(der []byte) (key *rsa.PrivateKey, err error)

4. 使用得到的公钥通过rsa进行数据加密

5.2 RSA解密

1. 将私钥文件中的私钥读出, 得到使用pem编码的字符串
2. 将得到的字符串解码
3. 使用x509将编码之后的私钥解析出来
4. 使用得到的私钥通过rsa进行数据解密

6. 哈希算法

6.1 概念

称谓: 单向散列函数, 哈希函数, 杂凑函数, 消息摘要函数

接收的输入: 原像

输出: 散列值, 哈希值, 指纹, 摘要

6.2 单向散列函数特性

1. 将任意长度的数据转换成固定长度的数据
2. 很强的抗碰撞性
3. 不可逆

1. MD4/MD5
    - 不安全
    - 散列值长度: 128bit == 16byte
2. sha1
    - 不安全
    - 散列值长度: 160bit == 20byte
3. sha2 - 安全
    - sha224
        - 散列值长度: 224bit == 28byte
    - sha256
        - 散列值长度: 256== 32byte
    - sha384
        - 散列值长度: 384bit == 48byte
    - sha512
        - 散列值长度: 512bit == 64byte

6.3 go中使用单向散列函数

```
// 第一种方式, 直接调用sum
// 适用于数据量比较小的情况
func Sum(data []byte) [Size]byte

// 第二种方式
// 1. 创建哈希接口对象
func New() hash.Hash
type Hash interface {
    // 通过嵌入的匿名io.Writer接口的Write方法向hash中添加更多数据，永远不返回错误
    io.Writer
    // 返回添加b到当前的hash值后的新切片，不会改变底层的hash状态
    Sum(b []byte) []byte
    // 重设hash为无数据输入的状态
    Reset()
    // 返回Sum会返回的切片的长度
    Size() int
    // 返回hash底层的块大小；Write方法可以接受任何大小的数据，
    // 但提供的数据是块大小的倍数时效率更高
    BlockSize() int
}
type Writer interface {
```

```
    Write(p []byte) (n int, err error)
}
// 2. 往创建出的哈希对象中添加数据
hash.Hash.Write([]byte("添加的数据..."))
hash.Hash.Write([]byte("添加的数据..."))
hash.Hash.Write([]byte("添加的数据..."))
hash.Hash.Write([]byte("添加的数据..."))
// 3. 计算结果, md5就是散列值
md5 := hash.Sum(nil);
// 散列值一般是一个二进制的字符串, 有些字符不可见, 需要格式化
// 格式化为16进制的数字串 - 0-9, a-f
func EncodeToString(src []byte) string
// 数据转换完成之后, 长度是原来的2倍

```

1. 计算一个大文件比如1G文件的散列值
2. 使用udp的方式分发秘钥, 进行一个对称加密的通信
    - 服务器
        - 生成密钥对
        - 公钥发送给客户端
    - 客户端
        - 客户端收到了公钥
        - 生成一个秘钥 - 用于对称加密
        - 使用公钥加密, 发送给服务器

复习

1. 概念
    - 加密三要素
        - 明文/密文
        - 秘钥
        - 算法
    - 对称加密和非对称加密
        - 对称加密: 加解密使用同一个秘钥, 1个
            - 效率高
        - 非...: 密钥对
            - 公钥加密, 私钥解密
            - 私钥加密, 公钥解密
    - 对称加密中的公开的加密算法
        - des
            - 分组长度: 8字节
            - 秘钥长度: 8字节
        - 3des

        - 分组长度: 8字节
        - 秘钥长度: 24byte
    - aes
        - 分组长度: 16字节
        - 秘钥长度: 16字节, 24字节, 32字节
            - 在go的api中只能使用16字节
- 对称加密的分组模式
    - EBC - 不推荐使用
    - CBC - 常用的方式
        - 准备的数据:
            - 初始化向量iv - 字符数组
                - 长度 == 明文分组长度
                - 加解密初始化向量值必须相同
            - 秘钥
                - 根据加密算法定

        `=` `=` `=` `=` `=` ==
    - OFB - 不推荐使用
    - CFB - 不推荐使用
    - CTR - 推荐使用, 效率最高