# 数字签名数字证书

数字签名，就是私钥拥有者向公钥拥有者发送 message 时，将 message 先进行 SHA Security Hash Algorithm 运算，得到 H，再将 H 用私钥加密，和 message 一起打包发送给公钥拥有者。签名采用的是非对称加密，一般是 RSA，依托于复杂的数学计算，包括大数乘法、大数模等等，耗时较久，安全性较高。message 不是明文传输的，明文传输有窃听风险，因此也要进行加密，通信双方通过非对称加密协商出对称加密密钥，协商密钥用来加密 message。类似于 TLS

在传输无误的情况下，因为 HASH 运算是单向加密的，而且复杂度极高，公钥拥有者接收到后，对 message 用协商密钥解密，将签名用公钥进行解密，然后将解密出来的 message 进行相同类型的 HASH 运算，将得到的结果于 H 比较，如果相同，说明内容没有被中途篡改；如果不相同，说明中途遭受攻击了。

下面来分析程序，

```
n = b'This is a test message'                                    #待加密的报文
h = SHA.new()                                                    #创建一个SHA对象
h.update(n)                                                      #对n进行SHA运算
print('Hash:',h.hexdigest(),'length:',len(h.hexdigest())*4)      #在屏幕上打印Hash结果，以及对应位数

sign_txt = 'sign.txt'                                            #创建文件对象

with open('master-private.pem') as f:                            #打开私钥，进行签名
    key = f.read()
    private_key = RSA.importKey(key)
    hash_obj = SHA.new(n)
    signer = Signature_pkcs1_v1_5.new(private_key)               #用私钥创建一个签名动作
    d = base64.b64encode(signer.sign(hash_obj))                  #签名，并用base64转码

f = open(sign_txt,'wb')                                          #将签名内容写入sign.txt
f.write(d)
f.close()

with open('master-public.pem') as f:
    key = f.read()                                               #打开私钥，进行签名确认
    public_key = RSA.importKey(key)
    sign_file = open(sign_txt,'r')
    sign = base64.b64decode(sign_file.read())                    #base64转码读出签名内容
    h = SHA.new(n)                                               #对正确的报文进行SHA运算
    verifier = Signature_pkcs1_v1_5.new(public_key)              #用公钥创建一个核实动作
    print('result:', verifier.verify(h,sign))
```

证书 pem 格式是不用 base64 转码的，文件格式比较特殊，公钥以“-----BRGIN PUBLIC KEY-----”开头，以“-----END PUBLIC KEY-----”结尾，私钥结构类似

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDkTOuidIcWU4LANMebfGrdxMfE
1qwjW6jEPIJJ1p+NwbkRS6PkbOAJWs99KQjhj0Fh2MtVZ/JqQkcNZTMdHoNReIN6
JAFihhzP3vbJnzE2FBi/e/Lj+wL90+rXxXsVqi25LwrMbbn9SQ3GZ1d35Elostxu
OjsY+d5IDyedCEDDNQIDAQAB
-----END PUBLIC KEY-----
~
```

在这个程序中要使 result 为 false，很简单，只需要等签名完成后，修改 n 的内容就可以了，其实就是模拟攻击模式。在后面加了个感叹号，HASH 运算结果就截然不同

```
n = b'This is a test message'
h = SHA.new()
h.update(n)
print('Hash:',h.hexdigest(),'length:',len(h.hexdigest())*4)

sign_txt = 'sign.txt'

with open('master-private.pem') as f:
    key = f.read()
    private_key = RSA.importKey(key)
    hash_obj = SHA.new(n)
    n = b'This is a test message!'
    signer = Signature_pkcs1_v1_5.new(private_key)
    d = base64.b64encode(signer.sign(hash_obj))

f = open(sign_txt,'wb')
f.write(d)
f.close()

with open('master-public.pem') as f:
    key = f.read()
    public_key = RSA.importKey(key)
    sign_file = open(sign_txt,'r')
    sign = base64.b64decode(sign_file.read())
    h = SHA.new(n)
    print("After change:")
    print('Hash:',h.hexdigest(),'length:',len(h.hexdigest())*4)
    verifier = Signature_pkcs1_v1_5.new(public_key)
    print('result:', verifier.verify(h,sign))
```

输出结果:

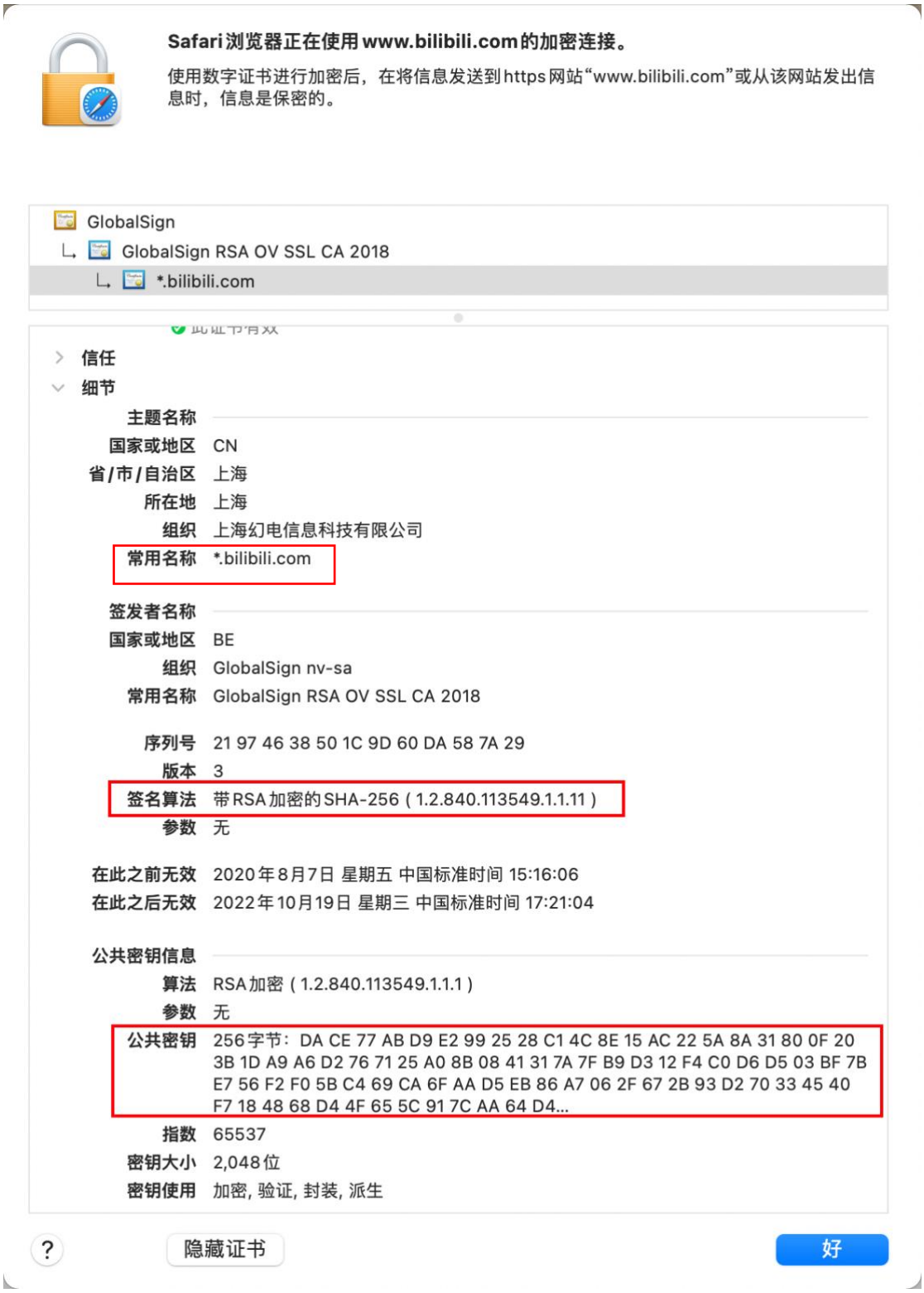
```
Hash: 0f7b9c1779a56cde0deec591da114e59a68db301 length: 160
After change:
Hash: a9d66d4b652597fb32dd1092e7c9cde18f0c7fbc length: 160
result: False
```

下面我们来研究点有趣的东西，数字证书

数字签名只能保证内容不被更改。在非对称加密通信过程中，服务器需要将公钥发送给客户端，在这一过程中，公钥很可能会被第三方拦截并替换，然后这个第三方就可以利用自己的公钥私钥冒充服务器与客户端进行通信，这就是“中间人攻击”(man in the middle attack)。解决此问题的方法是通过受信任的第三方交换公钥，具体做法就是服务器不直接向客户端发送公钥，而是要求受信任的第三方，也就是权威数字证书认证机构（Certificate Authority，简称CA)将公钥合并到数字证书中，然后服务器会把公钥连同证书一起发送给客户端，私钥则由服务器自己保存以确保安全。

数字证书可以用来确保公钥和公钥所有者的对应关系，机构或者公司将域名和公钥拿到 CA 注册，CA 将内容摘要映射到一个哈希值，然后用 CA 用自己的私钥对哈希值进行数字签名，内容再加上 CA 签名就得到了证书，值得一提的是，CA 的公钥、根证书一般都集成到我们的系统或者浏览器中，这时，如果还有攻击者想伪造，证书检测这关是过不了的。在 TLS 握手过程中，Sever Hello 阶段服务器会向客户端发送证书，如果证书过期或者未注册，浏览器都会给我们发起提醒。

我们来简单地看个证书：这是 b 站的 EV 证书


Safari浏览器正在使用 www.bilibili.com 的加密连接。
使用数字证书进行加密后，在将信息发送到https网站“www.bilibili.com”或从该网站发出信息时，信息是保密的。
GlobalSign
GlobalSign RSA OV SSL CA 2018
*.bilibili.com
信任
细节
主题名称
国家或地区 CN
省/市/自治区 上海
所在地 上海
组织 上海幻电信息科技有限公司
常用名称 *.bilibili.com
签发者名称
国家或地区 BE
组织 GlobalSign nv-sa
常用名称 GlobalSign RSA OV SSL CA 2018
序列号 21 97 46 38 50 1C 9D 60 DA 58 7A 29
版本 3
签名算法 带RSA加密的SHA-256 ( 1.2.840.113549.1.1.11 )
参数 无
在此之前无效 2020年8月7日 星期五 中国标准时间 15:16:06
在此之后无效 2022年10月19日 星期三 中国标准时间 17:21:04
公共密钥信息
算法 RSA加密 ( 1.2.840.113549.1.1.1 )
参数 无
公共密钥 256字节：DA CE 77 AB D9 E2 99 25 28 C1 4C 8E 15 AC 22 5A 8A 31 80 0F 20 3B 1D A9 A6 D2 76 71 25 A0 8B 08 41 31 7A 7F B9 D3 12 F4 C0 D6 D5 03 BF 7B E7 56 F2 F0 5B C4 69 CA 6F AA D5 EB 86 A7 06 2F 67 2B 93 D2 70 33 45 40 F7 18 48 68 D4 4F 65 5C 91 7C AA 64 D4...
指数 65537
密钥大小 2,048位
密钥使用 加密, 验证, 封装, 派生
?
隐藏证书
好