1.4-1密码学基本知识

2021年5月26日 19:10

一些基本概念

加密

密码学中的加密指的是将明文信息改变为难以读取的密文内容，使之不可读的过程。只有拥有解密方法的对象，经由解密过程，才能将密文还原为正常可读的内容。理想情况下，只有经授权的人员能够读取密文所要传达的信息。即便任何加密后的消息都有可能被破解，但经授权的人员可以轻松的获取信息，而未经授权的人员破解加密信息往往需要大量的时间和算力。

加密方式主要有两种：

1.**对称密钥加密**，也称**私钥加密**，加密和解密时使用相同的密钥。常见的对称加密算法有DES、AES、3DES等等。

与另一种加密方式相比，对称密钥加密的计算量通常较小，加密效率较高。但相对不是很安全。（我的理解是，在通过对称加密方式进行加密之前，信息传递双方必须先约定好所使用的的密钥，如果想要避免密钥泄露导致的信息不安全，就不能通过网络约定密钥，这使得这种加密方式不是很方便）

2.**非对称密钥加密**，也称**公钥加密**。加密和解密时分别使用不同的密钥。一把作为公钥，一把作为私钥。公钥加密的信息，只有私钥才能解密。私钥加密的信息，只有公钥才能解密。

非对称加密算法实现机密信息交换的基本过程是：甲方生成一对密钥并将其中的一把作为公用密钥向其它方公开；得到该公用密钥的乙方使用该密钥对机密信息进行加密后再发送给甲方；甲方再用自己保存的另一把专用密钥对加密后的信息进行解密。甲方只能用其专用密钥解密由其公用密钥加密后的任何信息。

非对称加密算法的密匙一般是通过一系列算法获取到的一长串随机数，通常随机数的长度越长，加密信息越安全。通过私钥经过一系列算法是可以推导出公钥的，也就是说，公钥是基于私钥而存在的。但是无法通过公钥反向推倒出私钥。

相较于对称密钥加密，非对称密钥加密速度较慢，但较为安全。

此外还有在加密时不考虑解密的加密方式等等。

数字证书

数字证书，也称公开密钥认证，是用于公开密钥基础建设的电子文件，用来证明公开密钥拥有者的身份。

数字证书的内容通常包含公钥信息、拥有者身份信息（主体）、以及数字证书认证机构（发行者）对这份文件的数字签名，以保证这个文件的整体内容正确无误。拥有者凭着此文件，可向电脑系统或其他用户表明身份，从而对方获得信任并授权访问或使用某些敏感的电脑服务。

证书产生的过程大致如下：

由权威的证书签发机构，称为CA，CA用非对称加密算法产生一对公私钥，然后用自己的私钥对自己的公钥进行**签名**，生成所谓的数字证书。

过程大概如下：

- 首先生成一个文件，文件包含公钥、签发者ID、证书所有者ID、有效期等信息，这些内容记为内容P。
- 然后使用hash算法，对这些内容hash计算，得到一个hash值H。
- 然后使用CA的私钥对H进行RSA加密，得到签名信息S。这个步骤称为签名，就是用私钥对某公开内容的hash值进行加密。
- 最后将P，S连成一个文件，这个文件就是所谓的数字证书了。所以数字证书里，包括证书持有者的身份信息，证书信息，证书持有人的公钥，以及签名信息。

如何确认这个证书的所有者是谁?

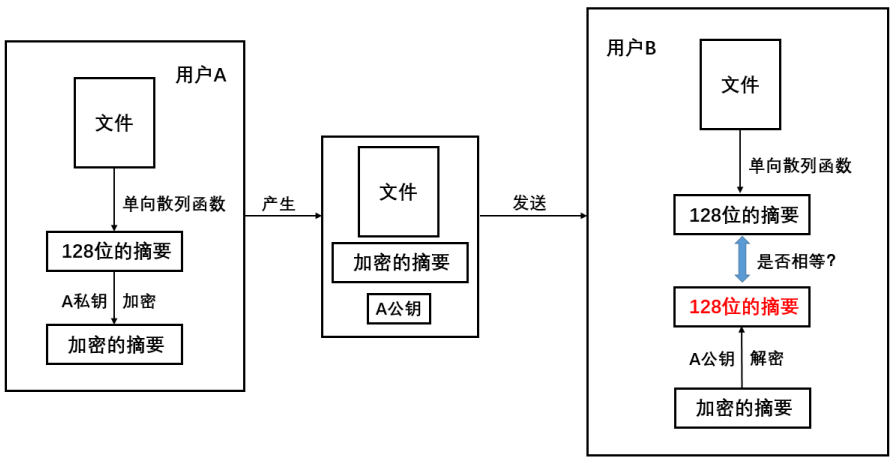
- 在得到数字证书时，数字证书中包含上述的内容P和内容S。
- 首先用与数字证书生成时同样的hash算法对P进行hash计算，得到一个hash值H1.
- 然后用公钥解密S，得到一个值H’ 。
- 最后对比H’ 和H1是否相等，如果相等，那么就证明这个证书是由CA签发给subject的证书，否则就说明：1.内容P被篡改过，或者2.证书不是由CA签发的。

只需要确保CA的权威性，就可以保证证书是可以信任的。全球权威的CA公司的证书，被各软件厂商设置为“可信任的**根**证书”，作为受信任的起始点，随后可以用这个证书来证明其他的证书。

（数字）签名

通过公钥加密的技术，鉴别数字信息。一套数字签名通常会定义两种互补的运算，一个用于签名，另一个用于验证。

在签名时将要加密的信息摘要通过使用者的私钥进行加密，然后公布公钥证者可以使用公钥解密消息并进行对比。在实际使用时往往对要加密信息的散列值进行加密，这样效率会高很多。



数字签名的实现需要三种算法：密码生成算法、验证算法和标记算法。

（数字）摘要

（数字）摘要指的是按照一定算法生成的，唯一对应于一个消息或者文本的固定长度的值，常常由一个hash函数产生。通过对收到的消息计算摘要，可以确定消息是否被篡改。
摘要并非原始数据加密后的密文，因而通过摘要也无法解密得到原始数据。通过hash函数计算产生的摘要值是固定长度的，相同正文产生的摘要必然相同。
常用的摘要算法包括MD5,SHA，MAC。

XML加密、签字规范

XML加密规范（XML Encryption）是W3C规定的加密标准，以XML的方式描述加密后的内容。当对XML元素或元素的内容进行加密时，在加密后的XML文档中，EncryptedData元素将替代该元素或内容。当对任意数据进行加密时，EncryptedData元素可以成为新的XML文档的根元素，或成为应用选定的XML文档的子元素。
XML加密规范定义了加密数据的步骤，解密加密数据的步骤，表示加密数据的XML语法，用于解密数据的信息以及加一些可用的加密算法，例如三重DES，AES和RSA。XML加密可以被应用于XML元素，XML元素内容和任意数据，包括XML文档。

例如：对该XML文档中的<CreditCard>元素进行加密

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
 <Name>John Smith</Name>
 <CreditCard Limit='5,000' Currency='USD'>
  <Number>4019 2445 0277 5567</Number>
  <Issuer>Example Bank</Issuer>
  <Expiration>04/02</Expiration>
 </CreditCard>
</PaymentInfo>
```

对称加密方式加密后的文档如下。

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
 <Name>John Smith</Name>
 <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
  xmlns='http://www.w3.org/2001/04/xmlenc#'>
  <EncryptionMethod
   Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
   <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
    <KeyName>John Smith</KeyName>
   </KeyInfo>
   <CipherData>
    <CipherValue>ydUNqHkMrD...</CipherValue>
   </CipherData>
 </EncryptedData>
</PaymentInfo>
```

<EncryptedData>元素表示加密的<CreditCard>元素。 <EncryptionMethod>元素描述了所应用的加密算法，在此示例中为三重DES。 <KeyInfo>元素包含检索解密密钥所需的信息，在此示例中，该信息是<KeyName>元素。 <CipherValue>元素包含通过对<CreditCard>元素进行序列化和加密而获得的密文。

非对称加密方式加密后的文档如下

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
 <Name>John Smith</Name>
 <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
  xmlns='http://www.w3.org/2001/04/xmlenc#'>
  <EncryptionMethod
   Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
  <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
   <EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <EncryptionMethod
     Algorithm='http://www.w3.org/2001/04/xmlenc#rsa-1_5'/>
    <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
     <KeyName>Sally Doe</KeyName>
    </KeyInfo>
    <CipherData>
     <CipherValue>yMTEyOTA1M...</CipherValue>
    </CipherData>
   </EncryptedKey>
  </KeyInfo>
  <CipherData>
   <CipherValue>ydUNqHkMrD...</CipherValue>
  </CipherData>
 </EncryptedData>
</PaymentInfo>
```

与之前不同的是，在<KeyInfo> 元素中包含了一个 <EncryptedKey>元素，内含加密后的公钥信息。

XML加密过程比较简单，主要分为以下几个步骤：

（1） 选定算法。首先要根据需要选定一个合适的加密算法。

（2） 数据转换。在加密前先将加密数据转换成字符流的格式。

（3） 加密。用选定的加密算法和加密密钥对待加密数据进行加密。

（4） 设置类型。设定加密的类型，是内容类还是元素类。

（5） 数据替换。将生成的EncryptedData元素中的数据替换原来要加密 的数据。

完整的官方文档：https://www.w3.org/TR/xmlenc-core/#sec-eg-Element

XML签字规范

一个定义数字签名的XML语法的W3C推荐标准。XML signature可以用来签名任何类型的数据（称作资源），最常见的是XML文档，但是任何可以通过URL访问的资源都可以被签名。

一个XML签名包含一个Signature元素，其名字空间为*http://www.w3.org/2000/09/xmldsig#。基本结构如下所示:*

```
<Signature>
  <SignedInfo>
    <SignatureMethod />
    <CanonicalizationMethod />
    <Reference>
      <Transforms>
      <DigestMethod>
      <DigestValue>
    </Reference>
    <Reference />
  </SignedInfo>
  <SignatureValue />
  <KeyInfo />
  <Object />
</Signature>
```

SignedInfo元素包含或引用签名后的数据，并指出使用了哪种算法。

SignatureMethod和CanonicalizationMethod元素被SignatureValue元素所使用，并包含在SignedInfo元素中以防止篡改。

一个或多个Reference元素通过URI引用的方式说明被签名的资源；以及在签名前对资源进行的任何转换。转换可以是一个XPath表达式，从文档树中选择一个子集[1]。

DigestMethod元素指定散列算法。

DigestValue元素包含转换后资源经过散列算法的结果。

SignatureValue元素包含一个经过Base64编码的签名结果 - 签名是按照SignedInfo元素中的SignatureMethod元素中指明的参数进行的，签名前要先根据CanonicalizationMethod元素中指定的算法进行规范化。

KeyInfo元素（可选）允许签名者为接收者提供验签该签名的密钥，通常是以一个或多个X.509数字证书的形式。如果没有出现KeyInfo元素，接收方必须从上下文中识别出验签的密钥。

Object元素（可选）包含被签名的数据，如果是enveloping signature（签名的数据在Signature元素内）的情况。

产生签名的步骤包括通过被签名的信息生成 Reference 元素和 SignatureValue元素。

1 . Reference元素的生成
对于每个被签名的数据对象：

- 将一定的转换算法（base64、XSLT变换算法等等）应用于数据对象。
- 计算变换后数据对象的摘要值。
- 创建一个 Reference 元素，包括数据对象的（可选）标识、任何变换元素（可选）、摘要算法和 DigestValue（摘要值）元素。

2 . SignatureValue元素的生成

- 使用 SignatureMethod、CanonicalizationMethod 和 Reference(s) 创建 SignedInfo 元素。
- 根据 SignedInfo 中指定的算法，基于 SignedInfo 中的信息计算 SignatureValue。
- 构造包含 SignedInfo、Object(s)（如果需要，编码可能与用于签名的编码不同）、KeyInfo（如果需要）和 SignatureValue 的 Signature 元素。

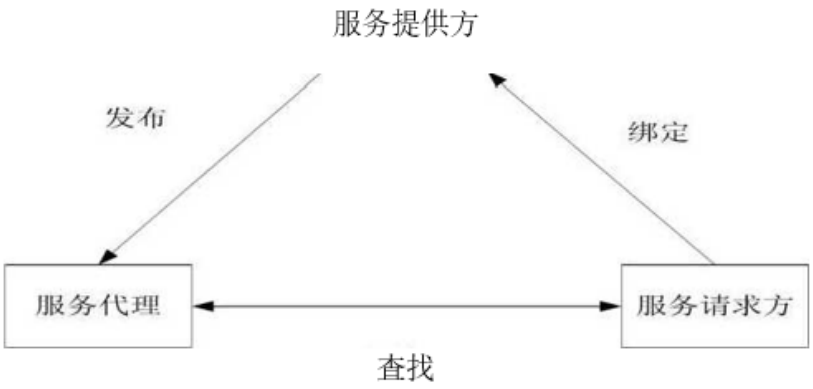
当**验证**一个XML签名时，需要遵守一个称作核心验证（Core Validation）的程序：

引用验证：每一个引用的摘要都通过获取相应的资源，并且按照指定的转换方法和摘要方法进行转换和摘要，然后将结果与DigestValue元素中的内容进行比较，如果不匹配，验证失败。
签名验证：SignedInfo元素使用CanonicalizationMethod元素中指定的XML标准化方法进行处理，密钥或取自KeyInfo元素或通过其他方法取得，然后通过SignatureMethod指定的签名方法进行验签。

详见W3C官网文档：
https://www.w3.org/TR/xmlenc-core/
https://www.w3.org/TR/xmldsig-core/

WS-Security规范

Web Services又称为XML Web Services,是一种借助Web来提供的服务形式,是由XML、WSDL, UDD1和SOAP 等技术组成的一个技术集，Web Services利用SOAP来通信，XML携带消息，WSDL文件对Web Services进行描述.Web Services不仅定义了应用程序如何在Internet上实现互操作，同时也 将紧密耦合的应用程序变成了松散耦合的应用程序，这一转变使得应用程序的逻辑结构更简单，研发者就没有必要浪费精力去定义合作应用程序，以及不同应用程序之间的通信规则，而且松散耦合的维护成本较低，稳定性也有 提高。Web Services 的体系结构模型如下：



该模型包含三个角色，分别为：服务提供者，服务请求者和服务代理。
服务提供者：就是指的是提供服务的服务所有者，发布自己的服务，并且对服务请求进行响应。
服务代理：注册已经发布的Web Services,并且对其进行分类，提供搜 索服务，将获取服务的信息绑定给请求方。
服务请求者：就是有特定需求的企业或个人，是服务的使用者。

。

SOAP (Simple Object Access Protocol)简单对象访问协议，是 XML Web Services的通信协议，描述XML如何调用的一种规范。
SOAP协议包含以下三个部分：

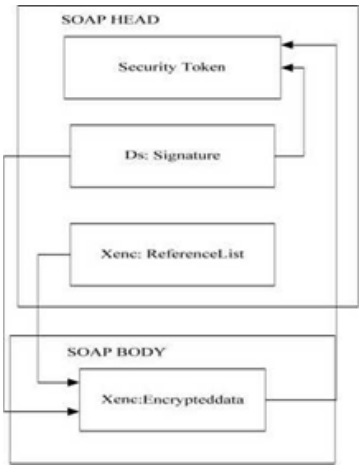
- SOAP封装。在SOAP封装中定义了一个有关消息的框架，该框架规定了消息的具体定义是什么，消息的运行机制，如何处理消息等要求。
- SOAP编码规范。它定义了一种编码机制，表示如何把应用程序的数据类型实例进行编码。
- SOAP RPC表示。SOAP RPC定义了两方面内容，首先是关于远程调用的规范，然后定义了远程过程如何应答的规范。

Web Services 具有松耦合性使得Web Services广泛应用于企业信息集成中，然而又因为其这一优点使得在应用的时候，必须要考虑如何应对安全挑战，利用XML安全机制是一个重要的解决问题的方法，包括XML Signature和XML Encryption等在内XML安全机制在保护消息传递完整性，保密性方面起到重要作用，WS-Security规范就是把这些安全规范如何运用到具体的Web Services中去做了一个详细的描述。

WS-Security（Web服务安全）是一种提供在Web services上应用安全的方法的网络传输协议。
协议包含了关于如何在Web服务消息上保证完整性和机密性的规约。
WS-Security规范本身并没有定义新的安全协议,而是在已存在的安全标准和规范中强调安全性。
WS-Security-提供了3种类型的安全机制：
（1）安全令牌传输；
（2）ＸＭＬ数字签名；
（3）ＸＭＬ加密.

它们定义了如下一些操作： 如何将签名信息附加到SOAP消息；如何加密消息头部；描述加密、签名消息常用的安全令牌如X509证书以及Kerberos票据等．使用XML签名以及安全令牌可以保证消息的完整性，保证消息在中途传输和处理过程中不被修改．加密SOAP消息中的核心部分可以保证整个消息的机密性．在消息中添加的安全性信息都是作为附加的控制信息以消息的形式传递的,不依赖于任何传输协议。因而,WS-Security 规范具有传输中立性,能保证端到端的安全性。
WS-Security定义的这些机制，并不可以不做任何工作直接应用，因为这些机制并不是完整的方案，它们是一些指导性的规范，是可以用来同别的协议联合使用的一些构件。这些机制可以供用户以自己需求的方式来使用，体现了 WS-Security 使用的灵活性，既可以独立使用，也可以以紧密集成的方式使用。

WS-Security规范对于**WEB服务安全的三个方面**：身份认证、完整性和保密性分别使用了三个元素来对应实现：Security Token , XML Signature, XML Encryption Reference List。这三个元素之间的关系可以用图来表示：



对于**身份认证**的方式，在 WS-Security规范中采用了多种身份验证方法，用户可以根据自己的需求选 择相应的方法，在SOAP 的Header中嵌入令牌，然后采取验证令牌是比较普遍的方式，而且各种各样 的令牌都可以嵌入到SOAP消息中,WS-Security定义了少量的令牌发送方案作为推荐使用的传送方式，例如包含Kerberos票据的令牌，包含 username/password系统的令牌，和X.509证书方式。

WS-Security采用**XML签名**方式确保消息完整性。

为确保**消息保密性**，WS-Security提供了 SOAP消息在传递过程中的加密方法，可以对全部数据加密，也可以对局部数据加密，同数字签名一样，WS-Security借鉴了 XML加密方法对文件进行加密，但是在WS-Security规范中又对XML加密规范进行了简化，只取用了其中几个元素，分别是 < EncryptedData > 、 < EncryptedKey > 、 < ReferenceList > ,

在实际应用中,服务提供者可以根据自己的需要制定相关的安全策略，并要求所有服务请求者提供相应的声明和证书,如果服务请求者无法提供,则可以拒绝提供服务。为了得到相应的声明和证书,服务请求者或服务提供 方可以从第三方获取信任令牌,由于第三方的服务提供者也具有自己的安全策略,因而不同安全域的服务提供者之间需要相互信任,为了取得这种相互信任,它们之间

可以制定相互认可的安全策略。这样即使跨多个不同的安全域,WS-Security也能够保证消息传递的安全性。当服务请求者具备相应的声明和证书之后,在请求服务时它们会将声明和证书随同SOAP 报文一 起 发 送 给服务提供者 。

参考链接

https://zh.wikipedia.org/wiki/%E5%8A%A0%E5%AF%86
https://zh.wikipedia.org/wiki/XML_Encryption
https://www.w3.org/TR/xmlenc-core/
https://www.w3.org/TR/xmldsig-core/
https://www.ibm.com/docs/en/was-nd/8.5.5?topic=authentication-xml-encryption
https://zh.wikipedia.org/wiki/%E5%85%AC%E9%96%8B%E9%87%91%E9%91%B0%E8%AA%8D%E8%AD%89
https://www.cnblogs.com/itps/p/12359865.html
https://www.zhihu.com/question/24294477/answer/74783418
https://blog.csdn.net/qq_29689487/article/details/81634057

[1]刘庆明. 基于WS-Security规范的Web Services安全性研究与实现[D].哈尔滨工程大学,2010.
[2]段友祥,包永堂.基于WS-Security规范的安全Web服务性能评估[J].小型微型计算机系统,2009,30(12):2364-2368.
[3]张维勇,程俊,王建新.基于WS-Security安全规范的Web服务设计[J].合肥工业大学学报(自然科学版),2006(08):972-975.

1.4-2常用算法原理简析

2021年5月28日 11:45

📖 RSA

RSA算法，目前使用最广泛的公钥加密算法，以其安全性得以著称。

RSA算法的**加密过程**可以简单的表示为：密文=明文求E次方后对N求余。只要知道E、N，就可以进行RSA加密。因此E和N的组合(E,N）就是公钥。

RSA的**解密**同样可以这样表示：明文 = 密文求D次方后对N求余。知道D和N就能进行解密了，所以D和N的组合（D,N）就是私钥。

那么，在进行RSA加密之前，首先需要**生成密钥对**（E,D,N）。这个密钥对生成的步骤如下：

- 求N：准备两个质数p，q。这两个数不能太小，太小则会容易破解，将p乘以q就是N
- 求L：L 是 p - 1 和 q - 1的最小公倍数，可用如下表达式表示：

  ```
  L=lcm（p - 1，q - 1）
  ```

- 求E：E必须满足两个条件：E是一个比1大比L小的数，E和L的最大公约数为1。用gcd(X,Y)来表示X，Y的最大公约数则E条件如下：

  ```
  1 < E < L
  gcd（E，L）=1
  ```

  之所以需要E和L的最大公约数为1是为了保证一定存在解密时需要使用的数D。现在我们已经求出了E和N也就是说我们已经生成了密钥对中的公钥了。

- 求D：数D是由数E计算出来的。D、E和L之间必须满足以下关系：

  ```
  1 < D < L
  E * D mod L = 1
  ```

以上就是RSA加密算法实现的数学过程。

理论上，通过因数分解是可以暴力破解私钥，从而破解加密信息的。但是对一个很大的证书做因数分解是十分困难的。因此在实际实现时p和q这两个质数选取的很大，这确保了RSA算法的可靠性。

```
12301866845301177551304949
58384962720772853569595334
79219732245215172640050726
36575187452021997864693899
56474942774063845925192557
32630345373154826850791702
61221429134616704292143116
02221240479274737794080665
35141959745985690214341 3
```

上面这个数字是人类已经分解的最大整数，它占了232个十进制位，768个二进制位。因此目前被破解的最长RSA密钥就是768位的。

📖 DES

DES算法是一种（基于56位密钥的）对称加密算法，在加密时将明文分组，每次按顺序取明文一部分。一般以64位作为一组。

加密过程大致如下：

1、明文数据分组，64位一组。

2、对每组数据进行初始置换。

即将输入的64位明文的第1位置换到第40位，第2位置换到第8位，第3位置换到第48位。以此类推，最后一位是原来的第7位。置换规则是规定的。L0(Left)是置换后的数据的前32位，R0(Right)是置换后的数据的后32位；

下表表示了数据置换后的位置，表中的数字代表数据置换之前所在的位置。

```
58,50,42,34,26,18,10,2,
60,52,44,36,28,20,12,4,
62,54,46,38,30,22,14,6,
64,56,48,40,32,24,16,8,
57,49,41,33,25,17, 9,1,
59,51,43,35,27,19,11,3,
61,53,45,37,29,21,13,5,
63,55,47,39,31,23,15,7,
```

3、在初始置换后，明文数据再被分为左右两部分，每部分32位，以L0，R0表示。

4、在秘钥的控制下，经过16轮的**f函数**运算。

函数f的输出经过一个异或运算，和左半部分结合形成新的右半部分，原来的右半部分成为新的左半部分。

函数f由四步运算构成：秘钥置换(Kn的生成，n=0~16)，总共16轮，每轮都会产生一个子密钥；扩展置换（也有一个置换表）；S-盒代替（8个用于代替的数组）；P-盒置换（置换表）。

f函数运算每一步的具体过程参见： https://www.cnblogs.com/jockming/p/12156844.html
https://blog.csdn.net/qq_27570955/article/details/52442092

加密和**解密**可以使用相同的算法。加密和解密唯一不同的是秘钥的次序是相反的。就是说如果每一轮的加密秘钥分别是K1、K2、K3...K16，那么解密秘钥就是K16、K15、K14...K1。为每一轮产生秘钥的算法也是循环的。加密是秘钥循环左移，解密是秘钥循环右移。解密秘钥每次移动的位数是：0、1、2、2、2、2、2、2、1、2、2、2、2、2、2、1。

DES现在已经不是一种安全的加密方法，这与它使用的密钥长度过短有关系。为了保证实际应用中的安全性，最初DES被3DES所替代，如今DES和3DES正在逐渐被高级加密标准AES所替代。

3DES

3DES（或称为Triple DES）是三重数据加密算法（TDEA，Triple Data Encryption Algorithm）块密码的通称。它相当于是对每个数据块应用三次DES加密算法。由于计算机运算能力的增强，原版DES密码的密钥长度变得容易被暴力破解；3DES即是设计用来提供一种相对简单的方法，即通过增加DES的密钥长度来避免类似的攻击，而不是设计一种全新的块密码算法。

3DES短发进行加密时需要三个密钥，均为56位，**加密**算法为：

密文 = EK3（DK2（EK1（平文））） 也就是说，使用K1为密钥进行DES加密，再用K2为密钥进行DES“解密”，最后以K3进行DES加密。

而**解密**则为其反过程：

平文 = DK1（EK2（DK3（密文））） 即以K3解密，以K2“加密”，最后以K1解密。

这三个密钥有三种情况：

密钥选项1：三个密钥是独立的。

密钥选项2：K1和K2是独立的，而K3=K1

密钥选项3：三个密钥均相等，即K1=K2=K3密钥选项1的强度最高，拥有3 x 56 = 168个独立的密钥位。

密钥选项2的安全性稍低，拥有2 x 56 = 112个独立的密钥位。该选项比简单的应用DES两次的强度较高，即使用K1和K2，因为它可以防御中途相遇攻击（英语：meet-in-the-middle attack）。

密钥选项3等同与DES，只有56个密钥位。这个选项提供了与DES的兼容性，因为第1和第2次DES操作相互抵消了。

加密过程有三种模式：

有4种模式，如下图所示：

（1） DES-EEE3 模式：该模式中共使用3个不同密钥，顺序使用3次DES加密算法。

（2） DES-EDE3 模式：该模式中共使用3个不同密钥，依次用加密—解密—加密。

（3） DES-EEE2 模式：该模式中共使用2个不同密钥，顺序使用3次DES加密算法，其中第一次和第三次加密使用的密钥相同。

（4） DES-EDE2 模式：该模式中共使用2个不同密钥，依次用加密—解密—加密，其中加密算法使用的密钥相同

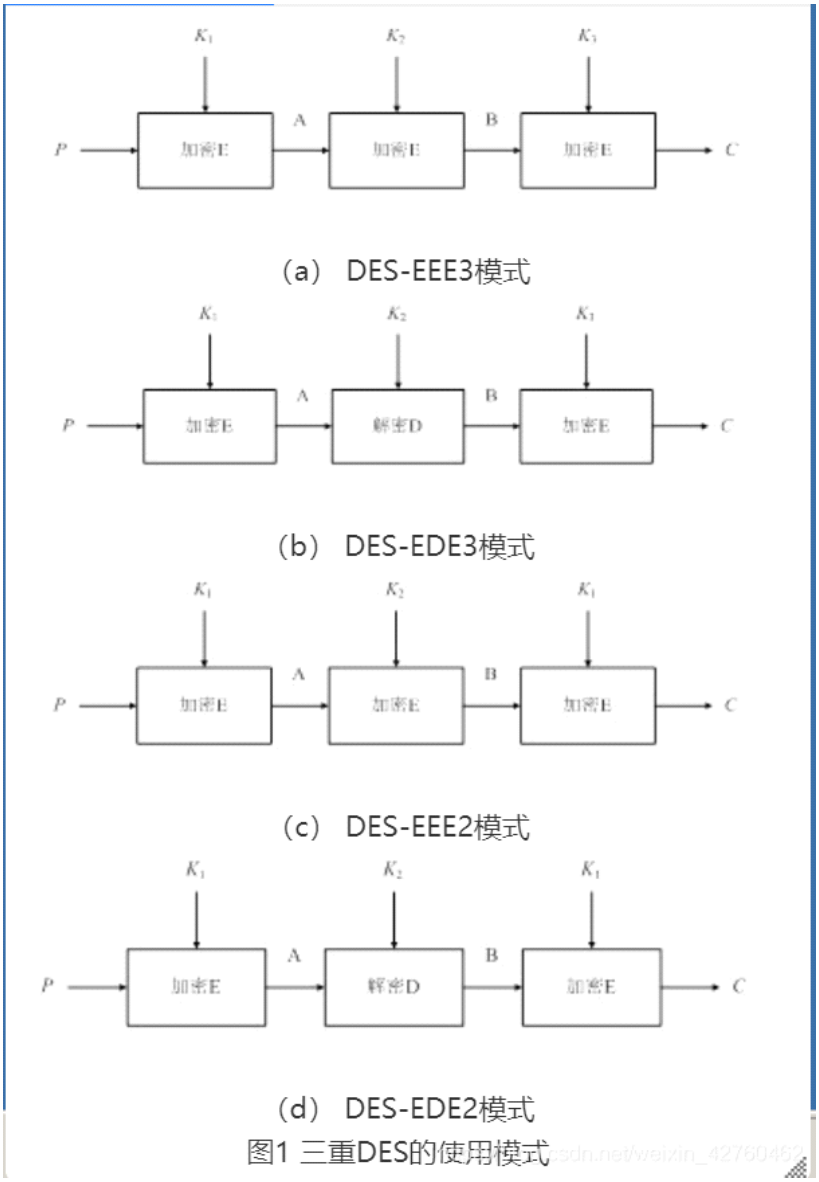


图1 三重DES的使用模式

。

MD5

MD5（消息摘要算法），算法可以产生一个128位的散列值（无论用于产生哈希值的数据长度是多少），用于确保信息的传输完整一致。在前面的基本知识过程中提过，一段信息对应唯一一个哈希值，同时通过哈希值不能倒推出信息的内容。

加密的过程简要描述如下：

1.首先对字符串进行处理，先对字符串进行填充，填充分为三种情况，记字符串长度为n，R=nMOD512。

- R=0时，需要补位，单补上一个512位的分组，因为还要加入最后64个位的字符串长度。
- R<448时，则需要补位到448位，后面添加64位的字符串长度。
- R>448时，除了补满这一分组外，还要再补上一个512位的分组后面添加64位的字符串长度。
  补位的形式是先填充一个1，再接无数个0，直到补足512位。

字符串完成填充之后，以512位为一组将字符串切割。

2. MD5有四个32位的被称作链接变量的整数参数，这是个参数我们定义为A、B、C、D其取值为：A=0x01234567，B=0x89abcdef，C=0xfedcba98，D=0x76543210。但考虑到内存数据存储大小端的问题我们将其赋值为：A=0x67452301，B=0xefcdab89，C=0x98badcfe，D=0x10325476。

同时MD5算法规定了四个非线性操作函数（&是与，|是或，~是非，^是异或）：

- F(X,Y,Z) =(X&Y)|((~X)&Z)
- G(X,Y,Z) =(X&Z)|(Y&(~Z))
- H(X,Y,Z) =X^Y^Z
- I(X,Y,Z)=Y^(X|(~Z))

这些函数是这样设计的：如果X、Y和Z的对应位是独立和均匀的，那么结果的每一位也应是独立和均匀的。利用上面的四种操作，生成四个重要的计算函数。首先我们声明四个中间变量a,b,c,d，赋值：a = A, b = B, c = C, d = D。然后定义这四个计算函数为：

- FF(a, b, c, d, M[j], s, ti)表示 a = b + ((a + F(b, c, d) + Mj + ti) <<< s)
- GG(a, b, c, d, M[j], s, ti)表示 a = b + ((a + G(b, c, d) + Mj + ti) <<< s)
- HH(a, b, c, d, M[j], s, ti)表示 a = b + ((a + H(b, c, d) + Mj + ti) <<< s)
- II(a, b, c, d, M[j], s, ti)表示 a = b + ((a + I(b, c, d) + Mj + ti) <<< s)

3.循环计算
定义好上述的四个计算函数后，就可以实现MD5的真正循环计算了。这个循环的循环次数为512位分组的个数。每次循环执行64次计算，上述4个函数每个16次，具体如下：

```
//第一轮循环计算

    FF(a,b,c,d,M[0],7,0xd76aa478);

    FF(d,a,b,c,M[1],12,0xe8c7b756);

    FF(c,d,a,b,M[2],17,0x242070db);

    FF(b,c,d,a,M[3],22,0xc1bdceee);

    FF(a,b,c,d,M[4],7,0xf57c0faf);

    FF(d,a,b,c,M[5],12,0x4787c62a);

    FF(c,d,a,b,M[6],17,0xa8304613);

    FF(b,c,d,a,M[7],22,0xfd469501) ;

    FF(a,b,c,d,M[8],7,0x698098d8) ;
```

```
FF(d,a,b,c,M[9],12,0x8b44f7af) ;
FF(c,d,a,b,M[10],17,0xffff5bb1) ;
FF(b,c,d,a,M[11],22,0x895cd7be) ;
FF(a,b,c,d,M[12],7,0x6b901122) ;
FF(d,a,b,c,M[13],12,0xfd987193) ;
FF(c,d,a,b,M[14],17,0xa679438e) ;
FF(b,c,d,a,M[15],22,0x49b40821);
//第二轮循环计算
GG(a,b,c,d,M[1],5,0xf61e2562);
GG(d,a,b,c,M[6],9,0xc040b340);
GG(c,d,a,b,M[11],14,0x265e5a51);
GG(b,c,d,a,M[0],20,0xe9b6c7aa) ;
GG(a,b,c,d,M[5],5,0xd62f105d) ;
GG(d,a,b,c,M[10],9,0x02441453) ;
GG(c,d,a,b,M[15],14,0xd8a1e681);
GG(b,c,d,a,M[4],20,0xe7d3fbc8) ;
GG(a,b,c,d,M[9],5,0x21e1cde6) ;
GG(d,a,b,c,M[14],9,0xc33707d6) ;
GG(c,d,a,b,M[3],14,0xf4d50d87) ;
GG(b,c,d,a,M[8],20,0x455a14ed);
GG(a,b,c,d,M[13],5,0xa9e3e905);
GG(d,a,b,c,M[2],9,0xfcefa3f8) ;
GG(c,d,a,b,M[7],14,0x676f02d9) ;
GG(b,c,d,a,M[12],20,0x8d2a4c8a);
//第三轮循环计算
HH(a,b,c,d,M[5],4,0xfffa3942);
HH(d,a,b,c,M[8],11,0x8771f681);
HH(c,d,a,b,M[11],16,0x6d9d6122);
```

```
HH(b,c,d,a,M[14],23,0xfde5380c) ;
HH(a,b,c,d,M[1],4,0xa4beea44) ;
HH(d,a,b,c,M[4],11,0x4bdecfa9) ;
HH(c,d,a,b,M[7],16,0xf6bb4b60) ;
HH(b,c,d,a,M[10],23,0xbebfbc70);
HH(a,b,c,d,M[13],4,0x289b7ec6);
HH(d,a,b,c,M[0],11,0xeaa127fa);
HH(c,d,a,b,M[3],16,0xd4ef3085);
HH(b,c,d,a,M[6],23,0x04881d05);
HH(a,b,c,d,M[9],4,0xd9d4d039);
HH(d,a,b,c,M[12],11,0xe6db99e5);
HH(c,d,a,b,M[15],16,0x1fa27cf8) ;
HH(b,c,d,a,M[2],23,0xc4ac5665);
//第四轮循环计算
II(a,b,c,d,M[0],6,0xf4292244) ;
II(d,a,b,c,M[7],10,0x432aff97) ;
II(c,d,a,b,M[14],15,0xab9423a7);
II(b,c,d,a,M[5],21,0xfc93a039) ;
II(a,b,c,d,M[12],6,0x655b59c3) ;
II(d,a,b,c,M[3],10,0x8f0ccc92) ;
II(c,d,a,b,M[10],15,0xffeff47d);
II(b,c,d,a,M[1],21,0x85845dd1) ;
II(a,b,c,d,M[8],6,0x6fa87e4f) ;
II(d,a,b,c,M[15],10,0xfe2ce6e0);
II(c,d,a,b,M[6],15,0xa3014314) ;
II(b,c,d,a,M[13],21,0x4e0811a1);
II(a,b,c,d,M[4],6,0xf7537e82) ;
II(d,a,b,c,M[11],10,0xbd3af235);
```

II(c,d,a,b,M[2],15,0x2ad7d2bb);

II(b,c,d,a,M[9],21,0xeb86d391);

4.处理完所有分组之后，得到一组新的A,B,C,D的值，将这些值顺序连接，就得到了128位的散列值。

MD5曾被用于文件校验、SSL/TLS、IPsec、SSH，但MD5早已被发现有明显的缺陷。

SHA1

安全哈希算法（Secure Hash Algorithm）主要适用于数字签名标准（Digital Signature Standard DSS）里面定义的数字签名算法（Digital Signature Algorithm DSA）。对于长度小于2^64位的消息，SHA1会产生一个160位的消息摘要。
SHA实际上是一系列算法的统称，分别包括：SHA-1、SHA-224、SHA-256、SHA-384以及SHA-512。后面4中统称为SHA-2，事实上SHA-224是SHA-256的缩减版，SHA-384是SHA-512的缩减版。各中SHA算法的数据比较如下表，其中的长度单位均为位：

类别	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
消息摘要长度	160	224	256	384	512
消息长度	小于2^{64}位	小于2^{64}位	小于2^{64}位	小于2^{128}位	小于2^{128}位
分组长度	512	512	512	1024	1024
计算字长度	32	32	32	64	64
计算步骤数	80	64	64	80	80

一般来说SHA-1算法包括有如下的处理过程：
（1）、对输入信息进行处理
既然SHA-1算法是对给定的信息进行处理得到相应的摘要，那么首先需要按算法的要求对信息进行处理。那么如何处理呢？对输入的信息按512位进行分组并进行填充。如何填充信息报文呢？其实即使填充报文后使其按512进行分组后，最后正好余448位。那填充什么内容呢？就是先在报文后面加一个1，再加很多个0，直到长度满足对512取模结果为448。到这里可能有人会奇怪，为什么非得是448呢？这是因为在最后会附加上一个64位的报文长度信息，而448+64正好是512。
（2）、填充长度信息
前面已经说过了，最后会补充信息报文使其按512位分组后余448位，剩下的64位就是用来填写报文的长度信息的。至次可能大家也明白了前面说过的报文长度不能超过264位了。填充长度值时要注意必须是低位字节优先。
（3）信息分组处理
经过添加位数处理的明文，其长度正好为512位的整数倍，然后按512位的长度进行分组，可以得到一定数量的明文分组，我们用Y0，Y1，……YN-1表示这些明文分组。对于每一个明文分组，都要重复反复的处理，这些与MD5都是相同的。

而对于每个512位的明文分组，SHA1将其再分成16份更小的明文分组，称为子明文分组，每个子明文分组为32位，我们且使用M[t]（t= 0, 1,.....15）来表示这16个子明文分组。然后需要将这16个子明文分组扩充到80个子明文分组，我们将其记为W[t]（t= 0, 1,.....79），扩充的具体方法是：当0≤t≤15时，Wt = Mt；当16≤t≤79时，Wt = (Wt-3 ⊕ Wt-8⊕ Wt-14⊕ Wt-16) <<< 1，从而得到80个子明文分组。

（4）初始化缓存

所谓初始化缓存就是为链接变量赋初值。前面我们实现MD5算法时，说过由于摘要是128位，以32位为计算单位，所以需要4个链接变量。同样SHA-1采用160位的信息摘要，也以32位为计算长度，就需要5个链接变量。我们记为A、B、C、D、E。其初始赋值分别为：A = 0x67452301、B = 0xEFCDAB89、 C = 0x98BADCFE、 D = 0x10325476、 E = 0xC3D2E1F0。

如果我们对比前面说过的MD5算法就会发现，前 4 个链接变量的初始值是一样的，因为它们本来就是同源的。

（5）计算信息摘要

经过前面的准备，接下来就是计算信息摘要了。SHA1有4轮运算，每一轮包括20个步骤，一共80步，最终产生160位的信息摘要，这160位的摘要存放在5个32位的链接变量中。

在SHA1的4论运算中，虽然进行的就具体操作函数不同，但逻辑过程却是一致的。首先，定义5个变量，假设为H0、H1、H2、H3、H4，对其分别进行如下操作：

（A）、将A左移5为与 函数的结果求和，再与对应的子明文分组、E以及计算常数求和后的结果赋予H0。

（B）、将A的值赋予H1。

（C）、将B左移30位，并赋予H2。

（D）、将C的值赋予H3。

（E）、将D的值赋予H4。

（F）、最后将H0、H1、H2、H3、H4的值分别赋予A、B、C、D

在这4轮运算中每轮所使用的函数和计算常数如下。

计算轮次	计算的步数	计算函数	计算常数
第一轮	0≤t≤19步	f_t(B,C,D)=(B&C)|(~B&D)	K_t=0x5A827999
第二轮	20≤t≤39步	f_t(B,C,D)=B⊕C⊕D	K_t=0x6ED9EBA1
第三轮	40≤t≤59步	f_t(B,C,D)=(B&C)|(B&D)|(C&D)	K_t=0x8F188CDC
第四轮	60≤t≤79步	f_t(B,C,D)=B⊕C⊕D	K_t=0xCA62C1D6

经过4轮80步计算后得到的结果，再与各链接变量的初始值求和，就得到了我们最终的信息摘要。而对于有多个明文分组的，则将前面所得到的结果作为初始值进行下一明文分组的计算，最终计算全部的明文分组就得到了最终的结果。

参考链接：

https://zh.wikipedia.org/wiki/%E9%AB%98%E7%BA%A7%E5%8A%A0%E5%AF%86%E6%A0%87%E5%87%86
http://www.360doc.com/content/18/0305/20/41428029_734552305.shtml
https://zh.wikipedia.org/wiki/SHA-1
https://www.cnblogs.com/foxclever/p/8282366.html
https://blog.csdn.net/gulang03/article/details/81175854
https://blog.csdn.net/lwanttowin/article/details/70799146
https://blog.csdn.net/dbs1215/article/details/48953589

https://blog.csdn.net/weixin_42760462/article/details/101379051
https://www.cnblogs.com/lq13035130506/p/10777187.html

1.4-3密码协议

2021年5月31日 18:23

密码协议，又称作安全协议、加密协议，是以密码学为基础的消息交换协议，其目的是在网络环境中提供各种安全服务。通过密码协议可以进行实体之间的认证、在实体之间安全的传输秘密、密钥，进行消息发送和接受的确认等等。网络安全协议建立在密码机制基础上，通过密码算法和协议逻辑实现加密和认证的过程。

按照密码协议实现的**功能**可以将密码协议**分为以下几种**：
认证协议、最小泄密协议、不可否认协议、公平性协议、身份识别协议、密钥管理协议。

按照密码协议**所处于的网络层**可以将密码协议**分为四层**：网络接口层（L2F,PPTP等）、网络层（IPSec等）、传输层（SSL,TLS等）、应用层（SSH,PGP,SET等）。

为了提高协议的安全性，在进行协议的设计和安全性分析时，应当假设安全攻击者除了可以窃听、阻止、截获所有经过网络的消息等之外，还具备以下知识和能力：

- 熟悉加解、解密、散列(hash)等密码运算，拥有自己的加密密钥和解密密钥；
- 熟悉参与协议的主体标识符及其公钥；
- 具有密码分析的知识和能力；
- 具有进行各种攻击，例如重放攻击的知识和能力。

针对安全协议的攻击常见的类型有中间人攻击、平行会话攻击、交错攻击、消息重放攻击。

中间人攻击在密码学和计算机安全领域中是指攻击者与通讯的两端分别创建独立的联系，并交换其所收到的数据，使通讯的两端认为他们正在通过一个私密的连接与对方直接对话，但事实上整个会话都被攻击者完全控制。在中间人攻击中，攻击者可以拦截通讯双方的通话并插入新的内容。
一个中间人攻击能成功的前提条件是攻击者能将自己伪装成每一个参与会话的终端，并且不被其他终端识破。中间人攻击是一个（缺乏）相互认证的攻击。大多数的加密协议都专门加入了一些特殊的认证方法以阻止中间人攻击。

许多**抵御**中间人攻击的技术基于以下认证技术：

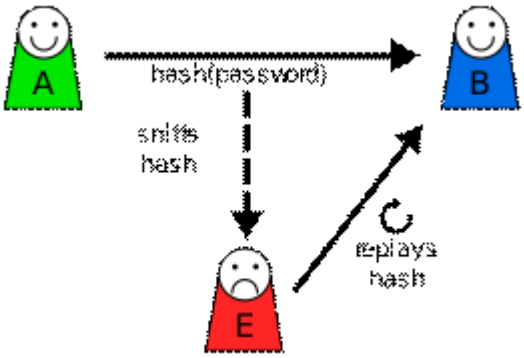
- **强力的相互认证**，例如：密钥（通常是高信息熵的密钥，从而更安全），或密码（通常是低的信息熵的密钥，从而降低安全性）
- **延迟测试**，例如使用复杂加密哈希函数进行计算以造成数十秒的延迟；如果双方通常情

况下都要花费20秒来计算，并且整个通讯花费了60秒计算才到达对方，这就能表明存在第三方中间人。

- **一次性密码本**可以对中间人攻击免疫，这是在对一次密码本的安全性和信任上创建的。公钥体系的完整性通常必须以某种方式得到保障，但不需要进行保密。密码和共享密钥有额外的保密需求。公钥可以由证书颁发机构验证，这些公钥通过安全的渠道（例如，随Web浏览器或操作系统安装）分发。公共密钥也可以经由Web在线信任进行在线验证，可以通过安全的途径分发公钥（例如，通过面对面的途径分发公钥）。

消息重放攻击

攻击者预先记录某个协议先前的某次运行中的某条消息，然后在该协议新的运行中重放(重新发送)记录的消息，导致通信方的错误判断，从而产生不真实的通信，且通信者察觉不到。
例如：Alice 想向Bob 证明自己的身份。 Bob 要求她的密码作为身份证明，爱丽丝应尽全力提供（可能是在经过诸如哈希函数的转换之后）； 与此同时，Eve窃听了对话并保留了密码（或哈希）。交换结束后，Eve（冒充Alice）连接到Bob。当被要求提供身份证明时，Eve发送从Bob接受的最后一个会话中读取的Alice的密码（或哈希），从而授予Eve访问权限。



防范消息重放攻击通常的对策包括，使用会话ID和组件编号标记加密组件（因为会话ID是随机生成的，难以复制）、添加时间戳、使用一次性密码等。

平行会话攻击

在攻击者的安排下，一个协议的多个运行并发执行，在此过程中，攻击者能够从一个运行中得到另外某个运行中的困难问题答案，从而达到攻击的目的。

交错攻击

某个协议的两次或者多次运行在攻击者的特意安排下按交织的方式执行。在这种模式下，攻击者可以合成其需要的特定消息并在某个运行中进行传送，以便得到某主体应答消息，此应答消息又被用于另外运行的协议中，如此交错运行，最终达到其攻击目的。

常见的一些密码协议有:

安全外壳协议SSH

建立在应用层基础上的安全协议。SSH 是目前较可靠，专为远程登录会话和其他网络服务提供安全性的协议。利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。SSH最初是UNIX系统上的一个程序，后来又迅速扩展到其他操作平台。

传统的网络服务程序，如：ftp、pop和telnet在本质上都是不安全的，因为它们在网络上用明文传送口令和数据，别有用心的人非常容易就可以截获这些口令和数据。而且，这些服务程序的安全验证方式也是有其弱点的， 就是很容易受到**中间人攻击**。通过使用SSH，你可以把所有传输的数据进行加密，这样"中间人"这种攻击方式就不可能实现了，而且也能够防止DNS欺骗和IP欺骗。使用SSH，还有一个额外的好处就是传输的数据是经过压缩的，所以可以加快传输的速度。

SSH之所以能够保证安全，原因在于它采用了**公钥加密**。

整个过程是这样的：（1）远程主机收到用户的登录请求，把自己的公钥发给用户。（2）用户使用这个公钥，将登录密码加密后，发送回来。（3）远程主机用自己的私钥，解密登录密码，如果密码正确，就同意用户登录。

安全电子交易协议SET

安全电子交易协议SET是一种应用于因特网（Internet）环境下，以信用卡为基础的安全电子交付协议，它给出了一套电子交易的过程规范。通过SET协议可以实现电子商务交易中的加密、认证、密钥管理机制等，保证了在因特网上使用信用卡进行在线购物的安全。SET协议解决了信用卡电子付款的安全保障性问题，包括：保证信息的机密性，保证信息安全传输，不能被窃听，保证支付信息的完整性，保证传输数据完整接收，在中途不被篡改；认证商家和客户，验证公共网络上进行交易活动包括会计机构的设置、会计人员的配备及其职责权利的履行和会计法规、制度的制定与实施等内容。

SET的工作流程

1）消费者利用自己的PC机通过因特网选定所要购买的物品，并在计算机上输入订货单、订货单上需包括在线商店、购买物品名称及数量、交货时间及地点等相关信息。

2）通过电子商务服务器与有关在线商店联系，在线商店作出应答，告诉消费者所填订货单的货物单价、应付款数、交货方式等信息是否准确，是否有变化。

3）消费者选择付款方式，确认订单签发付款指令。此时SET开始介入。

4）在SET中，消费者必须对订单和付款指令进行数字签名，同时利用双重签名技术保证商家看不到消费者的帐号信息。

5）在线商店接受订单后，向消费者所在银行请求支付认可。信息通过支付网关到收单银行，再到电子货币发行公司确认。批准交易后，返回确认信息给在线商店。

6）在线商店发送订单确认信息给消费者。消费者端软件可记录交易日志，以备将来查询。

7）在线商店发送货物或提供服务并通知收单银行将钱从消费者的帐号转移到商店帐号，或通知发卡银行请求支付。在认证操作和支付操作中间一般会有一个时间间隔，例如，在每天的下班前请求银行结一天的帐。

安全套接层协议SSL

安全套接层协议SSL是传输层安全性协议TLS的前身，它们的目的是为互联网通信提供安全及数据完整性保障。

SSL协议由**SSL记录协议**和**SSL握手协议**两部分组成。

1. **SSL记录协议**：

所有的传输数据都被封装在记录中。记录是由记录头和长度不为0的记录数据组成的。所有的SSL通信包括握手消息、安全空白记录和应用数据都使用SSL记录层。SSL记录协议包括了记录头和记录数据格式的规定。

1) SSL记录头格式：

SSL的记录头可以是两个或三个字节长的编码。SSL记录头的包含的信息包括：记录头的长度、记录数据的长度、记录数据中是否有粘贴数据。其中粘贴数据是在使用块加密算法时，填充实际数据，使其长度恰好是块的整数倍。

当数据头长度是三个字节时，次高位有特殊的含义。次高位为1时，标识所传输的记录是普通的数据记录；次高位为0时，标识所传输的记录是安全空白记录(被保留用于将来协议的扩展）。

2) SSL记录数据的格式：

SSL的记录数据包含三个部分：MAC数据、实际数据和粘贴数据。

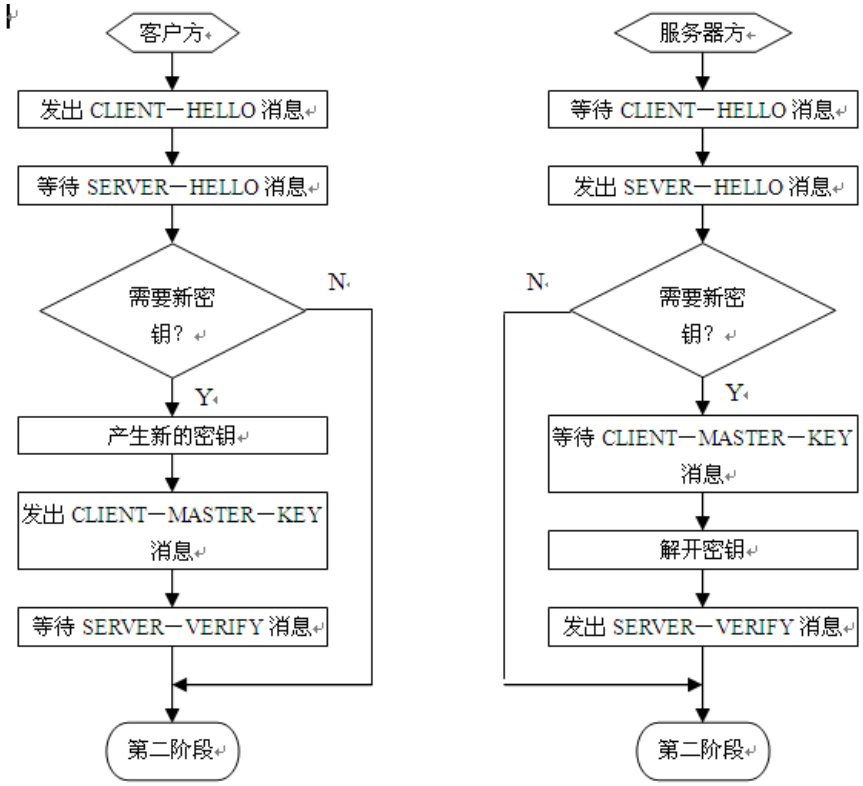
MAC数据用于数据完整性检查。计算MAC所用的散列函数由握手协议中的CIPHER - CHOICE消息确定。

2. **SSL握手协议：**

SSL握手协议包含两个阶段，第一个阶段用于建立私密性通信信道，第二个阶段用于客户认证。

1) 第一阶段：

第一阶段是通信的初始化阶段，通信双方都发出HELLO消息。当双方都接收到HELLO消息时，就有足够的信息确定是否需要一个新的密钥。若不需要新的密钥，双方立即进入握手协议的第二阶段。否则，此时服务器方的SERVER - HELLO消息将包含足够的信息使客户方产生一个新的密钥。这些信息包括服务器所持有的证书、加密规约和连接标识。若密钥产生成功，客户方发出CLIENT - MASTER - KEY消息，否则发出错误消息。最终当密钥确定以后，服务器方向客户方发出SERVER - VERIFY消息。因为只有拥有合适的公钥的服务器才能解开密钥。下图为第一阶段的流程：



2) 第二阶段：

第二阶段的主要任务是对客户进行认证，此时服务器已经被认证了。服务器方向客户发出认证请求消息：REQUEST - CERTIFICATE。当客户收到服务器方的认证请求消息，发出自己的证书，并且监听对方回送的认证结果。而当服务器收到客户的认证，认证成功返回SERVER - FINISH消息，否则返回错误消息。到此为止，握手协议全部结束。

- **网络认证协议kerberos**

协议的设计目标是通过密钥系统为客户机 / 服务器应用程序提供强大的认证服务。该认证过程的实现不依赖于主机操作系统的认证，无需基于主机地址的信任，不要求网络上所有主机的物理安全，并假定网络上传送的数据包可以被任意地读取、修改和插入数据。

认证过程具体如下：

首先，用户使用客户端（用户自己的机器）上的程序进行登录：

- 用户输入用户ID和密码到客户端。
- 客户端程序运行一个单向函数把密码转换成密钥，这个就是客户端（用户）的“用户密钥”(user's secret key)。

随后，客户端认证（客户端(Client)从认证服务器(AS)获取票据授权票据（TGT））：

- Client向AS发送1条明文消息，申请基于该用户所应享有的服务，该AS能够从本地数据库中查询到该申请用户的密码，并通过相同途径转换成相同的“用户密钥”(user's secret key)。
- AS检查该用户ID是否在于本地数据库中，如果用户存在则返回2条消息：
  消息A：Client/TGS会话密钥（该Session Key用在将来Client与TGS的通信（会话）上），通过用户密钥(user's secret key)进行加密
  消息B：票据授权票据(TGT)（TGT包括：消息A中的“Client/TGS会话密钥”

(Client/TGS Session Key)，用户ID，用户网址，TGT有效期），通过TGS密钥(TGS's secret key)进行加密

- 一旦Client收到消息A和消息B，Client首先尝试用自己的“用户密钥”解密消息A，如果用户输入的密码与AS数据库中的密码不符，则不能成功解密消息A。输入正确的密码并通过随之生成的“用户密钥”才能解密消息A，从而得到“Client/TGS会话密钥”(Client/TGS Session Key)。拥有了“Client/TGS会话密钥”(Client/TGS Session Key)，Client就足以通过TGS进行认证了。

然后，服务授权（client从TGS获取票据(client-to-server ticket)）：

- 当client需要申请特定服务时，其向TGS发送以下2条消息：
  消息c：即消息B的内容（TGS's secret key加密后的TGT），和想获取的服务的服务ID（注意：不是用户ID）
  消息d：认证符(Authenticator)（Authenticator包括：用户ID，时间戳），通过Client/TGS会话密钥(Client/TGS Session Key)进行加密
- 收到消息c和消息d后，TGS首先检查KDC数据库中是否存在所需的服务，查找到之后，TGS用自己的“TGS密钥”(TGS's secret key)解密消息c中的消息B（也就是TGT），从而得到之前生成的“Client/TGS会话密钥”(Client/TGS Session Key)。TGS再用这个Session Key解密消息d得到包含用户ID和时间戳的Authenticator，并对TGT和Authenticator进行验证，验证通过之后返回2条消息：
  消息E：client-server票据(client-to-server ticket)（该ticket包括：Client/SS会话密钥(Client/Server Session Key），用户ID，用户网址，有效期），通过提供该服务的服务器密钥(service's secret key)进行加密
  消息F：Client/SS会话密钥(Client/Server Session Key)（该Session Key用在将来Client与Server Service的通信（会话）上），通过Client/TGS会话密钥(Client/TGS Session Key)进行加密
- Client收到这些消息后，用“Client/TGS会话密钥”(Client/TGS Session Key)解密消息F，得到“Client/SS会话密钥”(Client/Server Session Key)。

最后，服务请求（client从SS获取服务）：

- 当获得“Client/SS会话密钥”(Client/Server Session Key)之后，Client就能够使用服务器提供的服务了。Client向指定服务器SS发出2条消息：
  消息e：即上一步中的消息E“client-server票据”(client-to-server ticket)，通过服务器密钥(service's secret key)进行加密
  消息g：新的Authenticator（包括：用户ID，时间戳），通过Client/SS会话密钥(Client/Server Session Key)进行加密
- SS用自己的密钥(service's secret key)解密消息e从而得到TGS提供的Client/SS会话密钥(Client/Server Session Key)。再用这个会话密钥解密消息g得到Authenticator，（同TGS一样）对Ticket和Authenticator进行验证，验证通过则返回1条消息（确认函：确证身份真实，乐于提供服务）：
  消息H：新时间戳（新时间戳是：Client发送的时间戳加1，v5已经取消这一做法），通

过Client/SS会话密钥(Client/Server Session Key)进行加密

- Client通过Client/SS会话密钥(Client/Server Session Key)解密消息H，得到新时间戳并验证其是否正确。验证通过的话则客户端可以信赖服务器，并向服务器（SS）发送服务请求。
- 服务器（SS）向客户端提供相应的服务。

参考链接

https://www.igi-global.com/dictionary/security-protocol/26119
https://www.ruanyifeng.com/blog/2011/12/ssh_remote_login.html
https://www.cnblogs.com/diffx/p/9553587.html
https://zh.wikipedia.org/wiki/%E9%87%8D%E6%94%BE%E6%94%BB%E5%87%BB
https://wiki.mbalib.com/wiki/%E5%AF%86%E7%A0%81%E5%8D%8F%E8%AE%AE
https://zh.wikipedia.org/wiki/%E5%AE%89%E5%85%A8%E5%8D%8F%E8%AE%AE
https://blog.csdn.net/weixin_41924879/article/details/101384544
https://zh.wikipedia.org/wiki/Kerberos
https://blog.csdn.net/kwame211/article/details/77099463