1. 加密通信和加密算法

对称加密算法包括以下五个基本成分：明文X、密文Y、加密算法E、解密算法D、加密密钥和解密密钥. 解密算法和加密算法互为逆运算，即 或

经加密的通讯过程图示如下：

通讯流程中，发信人Alice手持明文X和加密密钥, 将二者作为参数传入加密算法E中计算出密文Y并发送出去，密文在信道上传输以后到达期望收信人Bob的手中，他手持密文Y和解密密钥, 将二者作为参数传入解密算法D中计算出明文X. 于是我们的Alice和Bob就完成了一次加密通信。

加密算法可以分为对称加密和非对称加密两类。二者的区别在于加密密钥和解密密钥是否相同。对于对称加密而言，加密密钥和解密密钥是相同的；而对于非对称加密而言，加密密钥和解密密钥是不同的。

那么这种区别究竟意味着什么呢？我们不禁要问，无论对称还是非对称，难道不是都需要双方通过另一个信道协商密钥吗？其实不是这样的，我们来看一下非对称密钥是怎么操作的：假设Alice需要给Bob发信，这时，应先由**Bob方面生成一对密钥**, 由和算法难以得到. 将用于加密的经过信道发送给Alice, 她收到以后，就可以利用其对信息进行加密，并将密文传给Bob, 而Bob则直接用本地的来进行解密。我们发现在这个过程中，始终被保存在Bob手里，而没有在危机重重的网络环境上传播，因此它可以说是非常安全的。也就是说非对称加密的核心在于由**接收方**生成密钥对，且**由和算法难以得到**. 而则可以无所顾忌地在网络上传输，**甚至**大胆地向所有人公开（快来给我发送RSA加密密信吧！），因此也被称为公钥，也被称为私钥，而非对称加密则也被称为**公开密钥体制**。

以下我们介绍这两类算法的一些具体实现。

**对称加密算法**

对称加密出现很早（最早可以追溯到凯撒密码Caesar cipher），非对称加密直到1970s才出现，目前对称加密仍然是两种算法中使用较为广泛的一种。

由于通常情况下**加密算法是公开的**（被标准化，甚至可能已经设计了专门的硬件），因此该通讯过程的保密性在于密钥能够被妥善保存而不被泄露，双方协商密钥的场所必须是安全的，或者至少比上述信道有更高的安全性，否则整个加密通讯毫无意义又浪费计算资源。

为了设计一个安全的加密方案，我们考虑在密钥未知的情况下对加密通信进行攻击（不懂攻焉知防），一般可以采取两种策略：第一是**密码分析学**，通过对算法特征的分析、一些已被截获明文—密文对来推测密钥；第二是**暴力破解**(brute-force attack)或称**穷举攻击**，尝试所有可能的密钥。要使用后一种方法在时效内完成破解显然极其依赖计算机的计算速度，在理论讨论中[[1]](#footnote-1)，应对暴力破解的唯一方法就是**加长秘钥长度**从而增加可能的密钥个数，使得预期破解时间被延长。

需要注意的是，暴力攻击不仅仅需要把密文转化为明文，还必须从得到的海量明文中把真正的明文识别出来，这在不同情况下的难度差异是很大的：如果明文是英文，则比较容易通过NLP的有关技术识别，但如果明文是二进制文件，如视频或音频等，则很难对分析进行自动化（根本原因还是机器学习发展程度不够），如果文件进行了压缩则难度更大。因此在解密时，知道某些有关可能明文的期望知识是非常重要的。

3. 对称分组加密

分组加密是被最广泛应用的一类对称加密形式。它的工作过程如下：先将明文划分成一系列**定长分组[[2]](#footnote-2)**（因此被称为分组加密），然后将每一组明文通过与密钥相关的某种算法分别转换为等长的密文. DES, 三重DES, AES都属于这类算法，以下分别介绍。

|  |  |  |  |
| --- | --- | --- | --- |
|  | DES | 三重DES | AES |
| 明文分组长度（位） | 64 | 64 | 128 |
| 密文分组长度（位） | 64 | 64 | 128 |
| 密钥长度（位） | 56 | 112/168 | 128/192/256 |

【数据加密标准DES】

DES采用64位的明（密）文分组和56位长度密钥。

上一节提到，对一个密码体制的安全性可以从算法特征和密钥长度两个方面来考虑。自DES提出以来，已经成为了研究最深入的加密标准，至今没有发现它存在致命弱点，但其密钥长度实在太短, 1988年7月电子前沿基金会(EFF)设计了一台“DES破解机”，用三天破解了DES. EFF公布了这台机器的细节使得其他人也可以破解DES.

【三重DES算法】

**非对称加密算法**

非对称加密，又称为公钥密码体制。

**数字签名**

数字签名的目的是为了核实报文确实是由某个个体发出，而不是其他人伪装的。利用公开密钥体制中私钥的保密性很容易做到这一点，假设Alice需要给Bob发送报文，并且Bob提前告知对方我需要验证报文确实由你发出，那么Alice就可以先利用自己的私钥（即解密密钥）对原文进行计算，将计算过后的报文发送给Bob, 对方再利用公钥对报文进行运算即可得到原文。这样通过私钥进行运算得到的报文就被称为**已签名的**报文。

机械地理解加密解密的同学会遇到一个问题：为什么可以先进行“解密”（用私钥）后进行“加密”（用公钥）呢？其实这并没有什么问题，所谓解密和加密只不过是一种变形算法而已，并且根据运算的可逆性我们有

所以这样子操作是完全可行的。

请注意，这里我们不再使用“明文”“密文”等词汇，这是因为数字签名并不是用作加密用途而仅仅是用于验证对方身份的，由于公钥是公开的，因此任何人都可以收到报文并验证发送者身份。

然而数字签名由于需要进行非对称加密算法的运算，一旦报文很长，就需要消耗大量的计算资源，因此一般将数字签名与散列函数(hash function)结合起来应用，即Alice先将报文进行hash运算得到报文的摘要，再对这个较短的摘要进行数字签名，在Bob收到报文+已签名的摘要以后，首先用公钥解开签名得到原摘要，接着对报文进行hash运算得到自己计算的摘要，如果二者相同就可以确保报文没有受到完整性攻击，并且可以验证报文发送者的身份。

**密钥管理**

1. 对称加密体制

在对称加密算法中，密钥必须被严格保护，因此密钥必须使用安全的方式进行协商。目前采取的技术是设立**密钥分配中心**(Key Distribution Center, KDC)进行管理。密钥中心与客户端的通信采用对称加密，事先商议好密钥（被称为用户的主密钥，这里将用户X的主密钥记作），当两台主机A和B需要进行对称加密通信时由密钥中心管理密钥分配，这里记作,流程如下：

主机A以明文方式向KDC表示自己希望和B进行通信，KDC接受请求后生成一个密钥用于此次A与B的通信并使用主密钥加密后发送给A, 在报文中还附带发送一个请A转送给B，并且已经用B的主密钥加密（因此A无法窥视其中的内容）的票据(ticket), 其中包含A和B的身份以及密钥（因为这些内容A都知道所以其实也没有必要去窥视，这里实际上是为了防止第三方攻击），A将ticket转送给B以后，B通过持有的解密以后就得知了A的通信请求，并且收到了此次通信的密钥, 从而完成了一次密钥分配，此后A和B就可以愉快地进行通信了。

请注意实践中的密钥分配和KDC使用的是更复杂的协议，以上只是一个理论中的理想模型。

1. 强调这一点是因为实践中还可以通过验证码等限制方式来防止不受限制的暴力破解。 [↑](#footnote-ref-1)
2. 问题：如果所加密数据不是分组长度的整数倍，如何处理？ [↑](#footnote-ref-2)