## 数字签名

数字签名的概念也许是现代密码学最基本和最有用的发明之一，每个用户都可以通过一种签名算法给一段消息进行签名，然后其他人就可以通过这个签名来判断消息的真实性。准确来说，就是每一个用户能够创建一对私钥和公钥密钥对，私钥自己保存，这样就只有自己能够通过这个私钥来对消息进行签名，其它任何人都能够通过签名者发布的公钥对这个消息的签名进行验证，验证者能够通过公钥验证这个消息内容自从被签名之后到底有没有被篡改过，因为只有签名者才有自己的私钥，所以签名者自从签名之后也不能够否认自己进行过的签名，

与纸质世界类似，人们可以在信上签名并把它封在信封里，也可以使用自己的私钥在电子邮件上签名，然后用接收者的公钥对结果进行加密。接收方可以分别使用自己的私钥和发送方的公钥执行反操作，即打开信件并验证签名。公共密钥技术在电子邮件中的应用在今天已经相当广泛。

如果公用密钥目录是通过网络访问的，则需要保护用户免受欺诈消息的攻击，而这些消息通常假装自己是目录中的公用密钥。 比较好的解决方案是使用证书，即由公共密钥目录管理员或其他受信方数字签名的用户公共密钥的副本。 如果用户A在本地保留这个目录管理器公用密钥的副本，则他可以验证公钥目录中所有已签名的通信，并避免被假公钥欺骗。 此外，每个用户都可以将其签名的任何消息发送给其公钥证书，从而消除了对中央目录的需要，并允许用户在没有目录管理员公钥的任何信息的情况下验证签名的消息。 这些讲义中有关密钥分发的部分讨论了这种网络组织中涉及的一些协议问题。

## 10.1 数字签名的组成要素

一个公钥框架下的数字签名方法通常被定义为一个由三个算法组成的三元组:

* G代表生成密钥的概率多项式时间算法，通过输入一个加密参数，来产生一个公钥和密钥对（P,S），P代表公钥，S代表密钥，通常表示为这样，其中密钥对（P,S）是由G算法产生的。
* 签名算法也是一个概率多项式时间算法，它通过加密参数、在范围内的密钥S，和一段消息，来产生一个称为消息m的签名的字符串。如果签名算法是概率性的而且，通常表示为这样：。
* 鉴别算法V也是一个概率多项式时间算法，这个算法通过一个公钥P，一段消息m，和对这个消息的签名s，通过这三个参数来进行验证，算法返回1代表签名有效，返回0则代表无效，只有当的时候，那么这个算法，输出为1即验证有效，其它其它任何时候都是为0。通常在上下文很明确的情况下，可以省略公钥并进行缩写为来表示对消息m的签名s进行鉴别。
* 数字签名系统的最后一个特点是它的安全性是由其概率多项式时间保证的。后面的部分会对这个定义进行说明。

既然验证算法V是概率性的，那么就能否放宽验证算法V的限制，具体就是通过验证全部消息m，全部的安全参数k，和所有密钥对，得到的的置信度来判断签名是否有效。同样要注意的是要被加密的消息m可能是纯文本，也可能是已经被加密过的，因为数字签名系统的消息空间可以是任何的子集。

## 10.2 数字签名：陷门函数模型

Diffie 和Hellman[72] 提出一个基于陷门函数的公钥密码体制，用户A通过对消息M添加一个这样的数字签名信息来对一个消息M签名，f是只有用户A知道对应的陷门信息的一个公共陷门函数，在的保证下，任何人都能够通过公钥目录上的A的公钥来核实签名的有效性。值得注意的是如果消息被更改的话那么对应的签名也会失效，所以用户A的消息在被签名之后可以检测出是否被篡改，接收者可以对收到的消息签名进行验证，以此保证消息的真实性和完整性，

因此，在他们最初的提议中，Diffie 和Hellman把加密和数字签名这两项任务联系起来。然而，我们将这两项任务分开。事实证明，就像一些密码方案只适用于加密而不适用于签名一样，许多人也提出了只适用于签名的方案，从而实现更高的安全性。

RSA公钥密码系统符合Diffie 和Hellman范式，它能以一种直接的方式实现数字签名。私有指数d在这里变成了签名指数，消息M的签名变成了，也符合Diffie 和Hellman范式。任何人都能够使用对应的公共鉴别指数e来核实，从而判别签名的有效性，如果这个公式成立，那么签名信息一定是对应签名指数d的持有人对消息M的签名产生的，事实上，也有可能反过来发生，即使用鉴别公式和公共指数e从签名反向计算出一个假消息M’，然而这样计算出来的一个消息很有可能是不通顺的，杂乱无章的。实际上，这个问题可以通过对变换后的消息进行签名，而不直接对消息M本身进行签名这种方式避免，函数f通常是一个标准公共单向函数。

Hellman提出的数字签名方法同样也可以表示为下面的这样的算法三元组：

* 密钥生成算法G：在F里面找出符合的。
* 签名算法：，输出。
* 鉴别算法：，如果输出1，否则输出0。

后面将会讨论这个算法和其他算法的安全性，下面首先将讨论数字签名算法安全性的定义

## 10.3 签名算法安全性的定义个证明

下面是按严重程度列出的三种基本类型的攻击：

* 唯密钥攻击：这种攻击里面，攻击者仅知道签名者的公钥，因此攻击者只能够核实收到的签名信息的有消息。
* 已知签名攻击：攻对手知道签名者的公钥，并看到合法签名者选择和产生的消息/签名对。 实际上，这是对手所能做到的最低限度。
* 选择消息攻击：攻击者可以让签名者对攻击者选择的数字消息进行签名，而这些选择性的消息也许就是跟据此前获得的签名所产生的。例如：人们可能会想到按要求签署文件的公证人。

攻击者更详细的可能的攻击可以参考{105}。

**成功的伪造一个签名意味着什么？**

下面是按伪造的签名成功可能性增加列出的几个常用攻击方式：

* 存在性伪造：攻击者成功的伪造了某一条消息的签名，而这个消息不必须是他指定的哪个消息。
* 选择性伪造：攻击者成功的伪造了他指定的部分消息的签名。
* 通用性伪造：攻击这尽管没有拿到密钥，但是他也能够伪造任何消息的签名。
* 全部破解：对手可以计算出签名者的密钥。

很明显，不同应用的只需要达到符合自己条件的安全等级就行了。有些时候，可以充分证明具有已知签名攻击能力的对手不能成功地进行选择性伪造，然而对于另外其他的应用（例如，当签名者是公证员或纳税申报人时），可能要求即使在存在伪造的情况下，具有选定签名攻击能力的对手也不能成功。

我们想达到的安全性目标是，在存在选择消息攻击的情况下，多项式时间对手无法伪造成功的概率极高，甚至无法生存地伪造。

如果一个可以使用真实签名者作为“圣谕”的敌人不能在多项式大小的时间内伪造任何不是从真实签名者处获得的消息的签名，那我们就可以认为数字签名是安全的。比如，一个对消息m生成签名的黑盒B，对于任何消息m都有，即验证签名有效，现有伪造算法F，接收一个可以进入黑盒B的公钥P作为输入，表示为。伪造算法F分为下面两步：首先发动一个选择消息攻击，然后输出一个新的可以被定义为任何消息签名对的伪造数据。对于全部伪造算法F，对于任何多项式Q，以及任何足够大的k，有：；成立。概率是由密钥对的选择，以及攻击算法F和黑盒B的随机决定的。Dillie和Hellman的原始建议不符合这种严格的安全性定义。可能只利用公共信息就可以伪造：随机选择一个s。用公钥s产生，这样s也变成了消息m的一个签名了。

### 10.3.2 RSA数字签名方法

第一个例子是金鱼RSA的加密系统。

公钥是组成的数值对，n是两个大素数的成绩，e是和互质的数，密钥是d，满足，签名就是计算，鉴别过程就是将签名提升到e次方，并将其与原始消息进行比较。

RSA在选择性信息攻击下通常是可被遗忘的。(或者，在已知消息攻击下存在可遗忘)。

证明：如果我们能够为两条消息生成签名，那么这两条消息的乘积的签名就是签名的乘积。假设m1和m2分别是两条消息，用黑盒；生成他们的签名。而后我们可以为这两个消息的乘积产生签名：。

要生成消息m的签名，首先要选择一个随机数，然后使用上面的方法产生，，然后有这些消息的乘积的签名如下：。

### 10.3.3El Gamal 的方法

这种数字签名系统的安全性依赖于解决一个称为Diffie-Hellman密钥交换（DHKE）问题的与离散对数问题有关的难题。DHKE的问题是在输入一个素数p，一个生成器g，并且，计算输出gxy mod p，目前已知的最好的解决DHKE的方式是，首先解决离散对数问题，而计算离散对数是否像Diffie-Hellman问题那样困难，目前还是一个悬而未决的问题。以下数字签名方案是概率性的。 DSS的近似变体已被认可为国家标准。

该方案的要点如下：

* 公钥：三元组，其中，p是素数，g是生成的生成器。
* 密钥：x ，满足。
* 签名：消息m的签名是（r，s）组成的，满足。
* 验证：验证。

为了产生组成签名的（r，s），签名者以这样一个随机数k开始，k要满足，。其中。然后计算满足。对于指数的关系是，因此，消息m的签名是（r，s）。

很明显，如果一个攻击者可以解决离散对数问题，那么他就能够通过公共文件里面的信息来计算密钥x，从而破解这个签名方法。不仅如此，如果攻击者找到了消息对应的密钥x，那么他就可以解决离散对数问题，因此使用伪随机数生成器来产生k就非常有必要。

在已知的消息攻击存在时，此方案里面可以伪造的。

关于基于离散日志的密钥交换协议的注意事项：有趣的是，两个人可以使用未知的DL问题交换秘密密钥而无需事先秘密会面，而DL问题不会产生陷阱门功能。 可以由人A和B在素数p和生成器g上达成一致。A选择一个秘密数字x，并发送给B，而B选择一个秘密数字y并发送

给A，然后A和B都可以计算，将这个视作共享密钥，如果计算xy像DLP一样困难的话，那么这个共享密钥不会被其他人知道。

### 10.3.4 Rabin的方法

Rabin[170]提出了一个方法，消息M的签名是M的平方根，再模n，然后分解为两个大素数的乘积，因为求平方根的能力与因式分解n的能力是等价的，除非对手可以分解n，否则对手不应伪造任何签名。然后考虑一下他的变体：当，的时候，签名就被唯一的决定了。

该论点假定对手仅能访问包含签名者的模数n的公钥。攻击者可以主动让真正的签名人签名，其中x是随机选择的，这样来主动攻击破坏这个方法，如果签名人同意并产生M的平方根y，则产生n 的非平凡因子的可能性只有一半。签名者因此出卖了自己的秘密！尽管拉宾提出了一些实用的技术来解决此问题，但它们的作用是消除对伪造因子的建设性减少。

现在来看一下部分细节，这种数字签名方法是基于计算一个计算一个合数的平方根模的难度的：

* 公钥：n=pq。
* 密钥：素数p、q。
* 签名：。
* 鉴别：验证。

声名10.3：该系统存在仅使用密钥攻击的情况。

证明：选择一个签名并将其平方以生成相应的消息。

声明10.4：面对选定的消息攻击，系统是完全可以攻破的。

证明：我们知道，如果我们可以找到一条消息的两个不同的平方根，那么就可以因数分解模数。选择一个值s，并使得得m=s2，然后s是消息m得有效签名。将m提交给黑盒。它产生相同签名s的几率是二分之一，如果重复了，就再进行这个过程，否则我们就有m得两个能恢复n得平方根。

安全性等同于因数分解

考虑到Rabin方案在面对选定的消息攻击时的不安全性，人们可能会假设不存在基于分解的安全数字签名系统。也就是说，这样一个方案，其中：

* 破解这个系统就等同于因式分解。
* 签名方法能够抵抗选择消息攻击。

错误证明：假设(1)成立可以证明(2)是不可能的。因为第一个句子是破解这个系统就等同于因式分解，我们知道，对于一个合数n的输入，下面的约简是可能的：

* 产生公钥P。
* 产生消息m。
* 使用攻击算法产生有效签名，重复这三个步骤，直到达到多项式次数。
* 因数n。

总得来说，面对选择的消息攻击，系统必须是不安全的，因为我们可以用CMA代替攻击者得算法。

这个论点有什么问题？首先，公共信息P只有一个模糊的定义，他没必要包含数字n；其次，CMA将始终针对固定的公共信息产生签名，而在上述减少中，可能需要在每次致电攻击者时使用不同的公共信息。

## 10.4 概率签名

概率技术也已应用于数字签名的创建。 这种方法是由Goldwasser, Micali, and Yao [107]提出来的，这种方法基于分解的难易程度和对RSA函数求逆的难易程度，证明对手很难使用已知的特征攻击来伪造。

Goldwasser, Micali, and Rivest [105]通过提出一种签名方案来增强此结果，该签名方案在选定的消息攻击下是不可伪造的。他们的方法基于因素分解得难度，和关于无爪陷门排列的存在性（即陷门无爪排列的f0、f1，在公共域中定义的陷门排列，对于这样的排列，很难找到满足得x和y）。

如最初所描述的，该方案尽管在理论上很有吸引力，但却是不够有效的。 但是，可以对其进行修改，以允许使用更紧凑的签名，除了使用公钥和秘密密钥以外，不使用签名之间的存储空间，甚至不需要对每个新签名进行随机选择。 特别是，Goldreich [95]提出了一些建议，使该方案的基于分解的版本更加实用，同时保留其安全性。

Bellare和Micali在[14]中展示了一种数字签名方案，其安全性可以基于任何陷门排列的存在（比无爪的要求更弱）。 然后，Naor和Yung [152]展示了如何从任何一种单向排列开始，设计一种数字签名方案，该方案可以通过选择的签名攻击来防止存在的伪造。 最后，Rompel [177]展示了给定任何单向函数的签名方法。 这些作品基于Lamport关于如何签名的早期想法[130]。 这个想法如下。如果f是单向函数，并且Alice发布了两个数字f（x0）= y0和f（x1）= y1，那么她可以通过释放x0来签名消息0，并且可以通过释放x0来类似地签名消息1。 消息x1。 默克尔[146]介绍了这一基本思想的一些扩展，包括构建一棵经过身份验证的值，其根存储在签名者的公钥中。下面我们将详细描述其中一些理论发展。

### 10.4.1 无爪陷门排列

介绍了无爪陷门排列的概念，并展示了如何在假设存在无爪置换对的情况下构造签名方案。

定义10.5令f0, f1在公共域D上排列，如果成立，则称（x,y,x）是无爪的。

定义10.6[一组无爪排列]，当满足谢列情况时，一组被称作为一组无爪陷门排列：

1. 存在一个算法G，满足输出两个元组对、，其中是的陷门信息。
2. 存在一种PPT算法，对于给定的和，计算。
3. 存在算法I，存在一些可忽略函数那民族对于范围充分大的k，有。

以下观察结果表明，一对无爪排列的存在并不一定表示无爪排列的存在。举例来说，通过下式定义一组RSA的排列：

，

。

因为这两个函数是交换的，所以很容易通过随机选择一个w和定义和来生成一个爪。

通常来收，具有的问题如下：

开放问题10.7 陷阱门排列族的存在是否暗示无爪陷阱门排列族的存在？

以上情况的对立面显然是正确的：对于一个给定的无爪序列生成器，很容易生成一个陷门排列，如果是在公共域上的一对无爪排列，也就是说在计算上很难找到一个单元组x,y,z是他们满足，而是陷门。否组，给定一个反演算法，z在D上均匀分布，不能忽略这样的概率，I能够产生，因此（x,y,z）是一个有爪的，这就矛盾了）

### 10.4.2 例子：因式分解很难得情况下，无爪排列的存在性

令n=pq，其中p和q是素数，，并且p=1 mod 8，q=7 mod 8。显然大于1/16的奇素数满足这个要求，令表示modn的二次残余集。首先：

1. 
2. 

明确的有一个平方根（x是一个Blum整数），然而又四个平方根y,-y,w,-w。跟w,-w又雅可比符号-1，y和-y又雅可比符号+1。

现在定义一个函数族，并且证明，假设假设分解的是很难得，则它是QRn上的无爪陷阱门排列族。对于，



从上面公式可以看出，这些函数是的排列。

声明：是无爪的。

证明：假设它不是无爪的，假设并且满足。这表明，然而，通过核实两边的雅可比符号可得：



这就是说，x是二次剩余，但是不是，因为会在n产生一个非平凡因子。

### 10.4.3 怎样对1bit进行签名

首先描述签名方案的基本构建块：对一个比特签名。

令D是无爪对的公共域，并且假设x是从D里面随机选择的。



令来对一个比特进行签名。通过验证来验证签名s。

声明10.8 上面的方法都可以防止选择消息攻击。

证明：反证法，假设上面的方法不能防止这种攻击，那就是说存在一个伪造算法可以对与给给定的公共信息伪造一个签名；F向b索要b的签名，并且F能够以大于的概率签下一个正确的签名。为了推导出这个矛盾，我们设计了一个算法，对于给定的，能够生成爪：

输入：。

输出：满足的x，y，z。

1. 随机选择，抛一次硬币，像下面这样并且放到公共文件里面：（其中f0和f1是一个排列，因此z是D上的均匀分布）。
2. 运行算法:
3. 如果F向索要签名，那么返回（1）.
4. 如果F向索要签名，那么回复。
5. 由假设可知，F可以为产生新的签名。

### 10.4.4如何对一个消息进行签名

与之前一样，D是无爪对的公共域，x是D中的一个随机数。



对于，通过下公式对第一个消息进行签名：



然后用下列公式验证签名：



对于：

，

明显是D上的排列，并且容易计算。通过将新的排列应用于之前的签名上来对下一个消息进行签名：



通过下列公式验证：



需要注意的点：

1. 用这个方法，签名的长度会随着被签名的消息的长度线性增长。
2. 很容易伪造我们之前看到的消息的前缀的签名，我们因此假设消息在被签名之前有一个前缀预编码和预消息预处理过程。

声明：这个方法不能抵抗已知消息攻击。

证明：假设存在，对给定的公共信息p和历史，对消息，运行算法，就能够找到一个满足的消息，于是就嫩巩固产生一个满足下列条件的签名：



然后设以一个算法A，这个算法使用F提供一个爪：

输入：。

输出：满足的a，b，c。

1. 选择，令，令，为公共文件。（z也是从D里面随机选择的）
2. 产生记录，表示，全部消息的字符串就被生成了。
3. 运行伪造算法来产生。
4. 极大可能是一个有效的签名，就是说walking back with ,跟据它提供的记录，将会到达x，因此一定要找到从回退的路径，令l为两条路径相交的位置，即是说，m在第一个l-1比特的地方同意，并且表示为，假设m的第l个比特为0，的第l个比特为1，并且令u，v为对应的，输出。

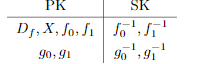
明显是有爪的，因此，将公共的f0，f1应用到伪造算法F的输出结果上，很大可能是矛盾的。

然而，是有爪的，这个方法似乎并不确保能够抵抗选择消息攻击，至少我们不知道如何去证明，下一章节我们将修改实现这个方法。

### 10.4.5 一个基于无爪排列的安全签名方法

令为无爪排列对的公共域，签名的消息，其中，其中是k的多项式。

选择两队无爪排列，选择能求出来的和，选择，令公钥包含，令密钥包含。



令o为链接函数，并设记录。

1. 选择。
2. 令。
3. 令
4. 令签名。
5. 令。

验证一个消息签名对，其中。

1. 令
2. 验证

如果是这样的话，签名就是有效的，并且验证函数，否则，这个方法充分利用了这样一个事实，即一个新的随机元素能够被用于取代每一个消息的X，所以伪造者不能通过请求多项式数量的消息的签名来获得信息，签名和验证程序要求在多项式时间内执行完毕，下面的理论可以证明他的安全性：

定理10.9无爪排列签名方法在无爪排列存在时可以抵抗CMA。

证明：反证法，加假设不成立，然后有一个伪造算法，由下列两部分组成：

1. F选择最多个信息获取他们的签名。
2. F输出，其中满足是和所有步骤1中的要求不同，并且

我们证明，如果存在这样一个F，那么就会有一个PTM a满足下列：

输入：里面的一个均匀分布满足未知。

输出：一个概率大于的h-claw，其中是k的多项式。

下面是通过得到的矛盾：

PTM A 是基于这样一个事实：当F成功地做到了上述的步骤2之后：

类型1：伪造：找到一个g-claw。

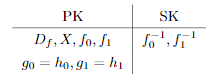
类型2: 伪造：找到一个f-claw。

类型3：伪造，找到，最后一个由签名者提供的记录点。

PTM A 由PTM的A1和A2组成，一个在另一个后面运行，A1基于这样的假设：F会产生一个f-claw，尝试找出有一个h-claw，A1和A2两者都在他们的公钥里面使用h0和h1，为了使用h0和h1对一个消息进行签名，这些PTM会计算，其中，并且将R作为使用，因此，无论是A1和A2在响应F的请求的时候都需要反转，注意因为是一个排列，如果R是随机的话，那么v也会是随机的。

下面是对A1的介绍：

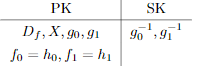
1. 选择一个满足 已知。令公钥包含  令密钥包含



1. 令记录，运行，当F请求消息的签名的时候：
2. 选择一个随机的。
3. 令
4. 令
5. 输出：
6. 令

然后F输出

1. 如果不成立的话，那么A1就是无效的。
2. 令，对于任意的i如果不成立的话，A1就无效，因为F不产生类型1的伪造。
3. 否则，令j满足，现在有，从这里可以轻松的获得一个h-claw。
4. 下面是A2的描述：
5. 选择满足一致的无爪对，令，选择，，令令公钥包含，令密钥包含

，

1. 令记录并运行，当F请求消息的响应的时候：
2. 令
3. 令
4. 输出
5. 令

F输出满足的。

1. 令
2. 有三种可能性：

F进行了类型1的伪造：这意味着存在某些i使得成立，此时A2无效。

F制作了类型2的伪造：中有一些第一个比特和A2的最后记录HN不同，对于一些有成立。可以得到，这个提供h-claw的A2。

F制作了类型3的伪造：对于一些比特b和字符串s 成立，因为A2选择的比特要紧邻着HN，如果另外一个请求时随机请求的，b大约有1/2的概率和d不同，这种情况下，A2会提供一个h-claw的A2.

假设有可能提供一个类型1的伪造，有p2的概率提供类型2的伪造，冰球有p3的概率会提供一个类型3的伪造。因为时无爪排列里面的均匀选择的。A1将会有p1的概率有效，并且A2会有的概率有效。因此，A1和A2将会以至多的概率有效。

注意：

1. 与先前的方案不同，此处的签名不必包含该方案所签名的所有先前消息；只有元素才会被赋予到签名上。
2. 签名的长度不必与签名的消息数成线性关系。可能是以线性方式连接在一起，为了建立树形结构，在这个结构里面，R1鉴别R2和R3，而R2又鉴别R4和R5，直到我们构建一个深度对数为B(k)的完整二叉树，其中B(k)是有待签署的签名总数的上限，然后重新命名叶子节点上的为。
3. 计算的代价是|m|，接下来我们将展示执行基于因式分解的无爪函数，m个因式可以被保存。

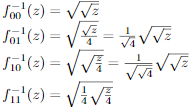
例子：计算的有效方。

如例10.4.2所示，如果分解很难，则特定的陷阱门排列族无爪。设n = pq，其中p和q是质数，p3mod 8, q7 mod 8，对于：

是无爪陷门排列的族。

注意：当才能写。

为了计算，我们首先要计算：



令i(m)为对应字符床m翻转过来的整数，很容易得到，现在，只需要计算一次，并且这个可以通过增加的幂高效的完成，。

### 10.4.6基于陷门函数的安全签名方法

本节包含陷门排列签名方案。 我们首先显示对单个b进行签名的方法：

1. 选择一个我们知道逆函数的陷门置换f。选择，令公钥包含，令密钥包含。



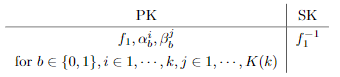
1. 对比特b进行签名，

简单的通过验证（b，s）。

用于签名多条消息的方案将上述方案用作构建块。 签名多个消息的问题是f不能重复使用。 因此，陷门置换签名方案为每个已签名的消息生成并签署新的陷门置换。 然后可以使用新的陷门排列对下一条消息进行签名。

陷门置换签名方案的描述：

1. 选择一个我们知道逆函数的陷门置换f1，选择 ，其中是一个k的多项式，令公钥包含f1和全部的令密钥包含令记录



签名消息

1. 令是使用 和生成的签名。
2. 选择一个新的陷门函数满足其逆已知的。
3. 令，其中是的二进制表示。
4. 的签名是，是使用和生成的签名。
5. 令

注意：我们假设用来表示的比特是够用的。

定义10.10如果存在陷门置换，则陷门置换签名方案对于CMA存在上是安全的。

证明：反证法，然后有一个伪造者F，可以请求其选择的消息，并且伪造一条尚未被请求过的消息，如果我们证明了这样的F切实存在，我们能发现一个PTM A’可以：

输入：逆已知的陷门函数h，并且

输出：以至少的概率输出，其中是一个k的多项式。

A’的结构如下所示：

1. A’ 将尝试使用h作为它的陷门排列之一来响应F的签名请求。因为A’不知道h的逆，它仅产生一个合适的的集合：通过随机和均匀的对所有选择 ，对于相同范围内的b和j，令，然后均匀的选择。第一阶段，A’ 的行为非常类似于陷门置换签名方案，但有一个例外，就是当A‘选择单向排列fn的时候，它将选择h。如果A’可以摆脱没改变的，尽管它不知道h的逆，那么它也可以对的请求进行签名。然而A’不改变。
2. 在随机选择一个，并令之和输入的w相等，令公钥包含。
3. 运行使用当前方案的F，注意到至少的概率，F会产生至少n个消息的请求当F请求一个消息的签名的时候，A’会有1/2的概率可以对进行签名，这是由于1/2的概率A’不会计算的逆。
4. 有的概率，Fhi成功输出一个效果很不错的伪造，为了使得成为一个优秀的伪造，它必须不知能够被验证，还要和之前的请求不同，至少有的概率伪造者将会选择和历史请求不同的，因此F将会使用h作为自己的陷门排列。
5. 如果这种情况发生了，有的概率伪造者会反转。
6. 如果是这样的话，A‘就输出。

A’成功的概率因此至少，因为是一个我们假设有矛盾的k的多项式。

## 10.5 坚固的安全性和基于RSA的实用签名

实际上，用于RSA签名的最广泛使用的范例是 hash签名然后解密：“首先通过 hash”将消息放入RSA的域点，然后解密（使用RSA解密指数进行幂运算）。这种范例的吸引力显而易见：签名仅需进行一次RSA解密，而验证仅需进行一次RSA加密。 此外，它易于实现。 因此，尤其是这是几种现有标准的基础。

在本节中，我们分析了这种范例。 不幸的是，我们将看到，即使假设基础散列函数是理想的，在有关RSA的标准假设下也无法证明标准化方案的安全性。 建议使用具有更好安全性的方案。

我们已经看到，这样的方案确实存在。遗憾的是，没有一种方案能与散列然后解密范式的方案相媲美。见第10.5.13节的比较）。那么，我们能做什么呢？

我们在这里提出了一些在效率上与 "先散列后解密 "的方案相匹配，但假设我们可以获得理想的散列函数，那么这些方案是安全的。正如第7.4.6节所讨论的那样，这意味着在形式上，散列函数被建模为随机的绕口令，而在实现上，散列函数是从密码学的散列函数中得到的。这代表了一种实际的折中方案，在这种情况下，我们可以在合理的安全保证下获得效率。关于这种方法的完整讨论见[15]。

我们提出并分析了两种方案。第一种是[15]的FDH方案。第二种是[26]的PSS方案。此外，我们提出了一种叫做PSS-R的方案，它具有消息恢复的特点。这是一种有效地缩短签名大小的有效方法。

现在让我们扩展以上所有内容。 我们首先看一下当前的做法。 然后，我们考虑[15，26]的全域哈希方案并讨论其安全性。 最后，我们来看看PSS和PSS-R及其确切的安全性。

我们提出了针对RSA的这些方案。 Rabin方案也可以这样做。 本节的材料主要来自[26]。 为了使本节内容完整，我们重复本章前面部分的一些基础知识。 对于具体的安全性，仍然存在不同的观点，因此材料并不是完全多余的。

### 10.5.1 数字签名方案

在公共密钥设置中，用于提供数据完整性的原语是数字签名方案。 就像消息身份验证方案一样，只是密钥结构不对称。 用于生成标签的密钥sk（在此设置中，标签通常称为签名）与用于验证签名的密钥pk不同。 此外，在对手也知道的意义上，pk是公开的。 因此，尽管只有拥有私钥的签名者才能生成签名，但拥有相应公钥的任何人都可以验证签名。

定义10.11数字签名方案DS =（K; S; V）由以下三种算法组成：

* 随机密钥生成算法K（不输入并且）分别返回一对密钥（pk; sk），公共密钥和匹配的秘密密钥，通过执行k算法，并让(pk;sk)为返回的密钥对。
* 签名算法S通过使用密钥sk和消息M来返回签名或者一个标签，该算法可以是随机的或有状态的。通过对S的输入sk和M进行操作，并令为返回的签名。
* 确定性验证算法V接受公钥pk，消息M和候选签名æ，以使M返回一个比特。通过来表示对输入的pk，M，运行V算法，并且令d为返回的比特。

我们要求，涡轮输出是K的输出，任何消息M，以及任何的输出，对于任何公钥密钥对都要满足，如果S是无状态的，则我们将每个M的集合与每个公钥关联一个消息空间Messages（pk）。

令S为想要具有数字签名功能的实体。 第一步是密钥生成：S运行K为其自身生成一对密钥（pk; sk）。 密钥生成算法由S本地运行。S将使用sk生成签名，其他人将使用pk验证这些签名。 后者要求任何希望验证S签名的人都必须拥有S生成的此密钥pk。 此外，必须确保验证者公钥是真实的，这实际上是S的密钥，而不是其他人的密钥。

有多种机制可用来确保预期验证者拥有签名者的真实公钥。 这些通常以密钥管理的名义进行。 很简单，这里有一些选择。 S可能将其公钥“移交给”验证者。更常见的是，S在某个可信任的服务器上以S的名称注册pk，该服务器充当公共电话簿，并且任何希望获得S的公钥的人都可以通过将服务器发送给服务器来请求它。 S的名称，并取回公共密钥必须采取步骤以确保此通信也已通过身份验证，这意味着验证程序确实与合法服务器进行通信，并且注册过程本身是真实的。

实际上，密钥管理本身就是一个主题，需要深入了解。 我们稍后会解决。 目前，要掌握的重要问题是问题之间的分离。 即，密钥管理过程不是数字签名方案本身的一部分。 在构建和分析数字签名方案的安全性时，我们假设任何预期的验证者都拥有签名者公钥的真实副本。 该假设在以下进行。

密钥结构到位后，S可以通过运行Ssk（M）返回签名，在某些文档M上生成数字签名。 然后，对（M; ）是文档的身份验证版本。 接收者B收到文件M 0和据称来自S的标签后，便使用指定的验证过程来验证签名的真实性，该过程取决于消息，签名和公钥，通过计算得到，这个计算出来的值1是一个比特，如果此值为1，则表示该数据是真实的，因此B接受它为来自S的数据。否则，它会将数据丢弃为非真实的。

可行的方案当然需要一些安全属性。 但这不是我们现在关心的。 首先，我们要确定构成方案规范的内容，以便我们知道要评估其安全性的对象的种类。

该定义的最后一部分说，正确生成的标签将通过验证测试。 这只是确保接收方将接受真实数据。

签名算法可能是随机的，意味着内部为通过抛硬币的方式确定其输出。 在这种情况下，可能有许多与单个消息M关联的正确标签。该算法也可能是有状态的，例如，利用发送者维护的计数器。 在那种情况下，签名算法将访问计数器作为全局变量，并根据需要对其进行更新。

与加密方案不同，加密方案的加密算法必须是随机的或有状态的，以确保方案的安全性，确定性的无状态签名算法不仅可能而且很常见。

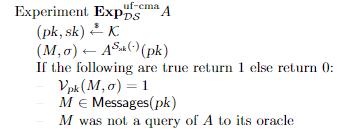
### 10.5.2安全的概念

数字签名旨在提供与消息身份验证方案相同的安全性； 唯一的变化是更加灵活的密钥结构。 因此，我们可以基于过去的工作来理解和确定消息身份验证的安全性概念。 数字签名签名的区别仅在于对手可以访问公钥。

攻击者F的目标是伪造：它想产生满足的文件M和标签，但是M并不是发件人S发起的。在试图制造伪造的过程中，允许对手进行选择消息的攻击，并且即使攻击者在制造伪造的可能性很小的情况下，该方案也是安全的。

令为任意数字签名方案，我们的目标是针对此方案，正式制定一种措施，以防止在选择消息攻击下进行伪造。 对手的行动被视为分为两个阶段：第一个学习阶段是对一个已知的甲骨文，其中是根据算法K优先选择的，要所有查询都是与此键关联的基础消息空间普通文本（pk）中的消息，它就可以以任意方式查询该甲骨文多达q次。 这个阶段结束后，便进入伪造截断，这个阶段输出一个（M，）其中，如果并且M之前从来没被攻击者查询过，那么此次攻击就算成功，因此，与任何对手F相关联都是成功概率。 （概率超过了密钥的选择，S可能做出的任何概率选择以及F做出的任何概率选择。）该方案的不安全性是“最聪明的”可能对手的成功概率。对手的资源受到一定程度的限制。

定义10.12 令为一个数字签名方案，假设A是一个可以访问oracle并返回一对字符串的算法。我们考虑以下实验:：



被定义为：



在消息身份验证方案的情况下，我们不仅为对手提供了一个用于生成标签的预言家，而且还为其提供了一个用于验证它们的预言家。 上面没有验证Oracle。 这是因为数字签名的验证不依赖于对手秘密的任何数量。 由于对手具有公共密钥并且知道算法V，因此可以通过运行后者来尽可能多地验证。

当我们谈论对手的时间复杂性时，我们指的是整个实验的最坏情况下的总执行时间。 这意味着在某种Ø固定RAM计算模型中，对手的复杂度定义为A的最坏情况执行时间加上对手A的代码大小（最坏的情况是指A的硬币或响应A的返回答案的最大值） oracle查询），再加上实验中其他操作的时间，包括生成密钥的时间以及通过执行加密算法计算oracle查询的答案的时间。

作为攻击者的资源，我们将考虑这种时间复杂度，消息长度π和对符号oracle的查询数量q。 我们将π定义为oracle查询的长度加上对手在伪造产品中输出的消息长度的总和。 在实践中，查询对应于由合法发送者签名的消息，并且有意义的是，获得这些示例比仅自己计算要昂贵。 也就是说，我们期望q小于t。 这就是为什么q; π是与t分开的资源。

### 10.5.3RSA参数的生成

RSA陷阱门置换被广泛用作数字签名方案的基础。 让我们看看如何。 我们从一个记号开始：

定义10.13令的整数，带有N，f的RSA函数为，通过进行定义。

与N相关的RSA函数； 因此，f仅是组中指数为f的幂运算，但在当前上下文中为其赋予一个新名称很有用。 以下总结了此功能的基本属性。 回想一下是群的阶。

命题10.14令满足ed=1的整数，然后RSA函数都是的序列，不仅如此，他们还是互逆。。

上面的帕列简单的意味着是的双射，或者是一对一的关系，条件ed=1说明d是组的逆。

命题10.14的证明：对于任意下列的取模的操作：



第三个等式是基于是组的序这一事实成立的，第四个使用的假设e，d的假设条件，能够这个名对于任意的：

，这两个事实就可以满足上卖弄的命题。

用N; e; d如命题10.14所述，我们指出：

* 对于任意的，能过通过已知的有效的计算
* 对于任意的，能通过已知的有效的计算。

现在考虑一个攻击者已知N，e，y并且计算，如果它是d，则可以通过上述方法有效地做到这一点，但是我们没有给出d。 事实证明，当参数为N时； 如果选择正确，此对抗性任务在计算上似乎是不可行的，并且此属性将构成基于RSA的非对称加密方案和数字签名方案的基础。 我们在本节中的目标是通过显示如何选择RSA参数，从而使上述计算困难的主张成为现实，并形式化其正确性，为以后的应用奠定基础。

命题10.15存在一个时间复杂度为的算法对于输入，其中，返回满足

命题10.15的证明：因为d是的逆，算法由和返回值组成，回想一下，模块化反演算法调用扩展的gcd算法作为一个子例程，并且运行时间在其输入的位长上是平方的。要选择RSA参数，需要运行一个生成器。 我们考虑几种类型的生成器：

1. p，q不同，且都为奇素数。
2. N=pq。
3. 

具有相关安全性参数k的RSA生成器是一种随机算法，不接受任何输入并返回一个对（（N; e）;（N， p，q;，d）），因此上述三个条件为真，并且 ，

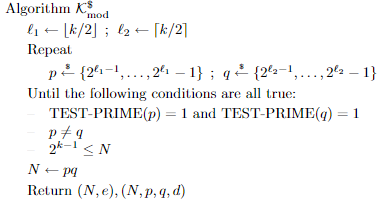
1. 
2. 

我们称N为RSA模数，或仅称为模数。 我们称e为加密指数，d为解密指数。

是群的大小，e，d是相关的素数。如上所述，我们将注意力集中在数字N上，数字N是两个不同的奇数质数的乘积。条件4是为了显示rsa生成器转换到，和

为了使参数生成可行，生成算法必须是高效的。可能有许多不同的肯能的生成器。我们举几个例子：

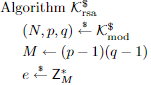
在模数生成中，我们通常选择素数p； q是随机的，每个q大约为k = 2位。对应的模量生成器与相关的安全参数K工作如下:



上面，TEST-PRIME表示接受整数输入并返回1或0的算法。它的设计是，当输入为素数时发生前者的概率很高，而当输入为合数时发生后者的概率很高。

有时，我们可能想要具有特殊形式的素数的模积，例如素数p； q使得（p-1）/2和（q-1）/ 2都是素数。 这对应于一个不同的模量生成器，其工作原理如上所述，只是将条件TEST-PRIME（（p-1）/ 2））= 1和TEST-PRIME（（ q-1）/ 2））=1。还有许多其他可能的模量生成器。

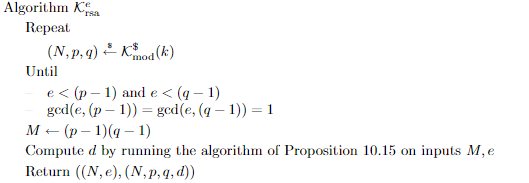
除N外的RSA生成器； ; q，需要产生指数e； d。 有几种选择。 一是首先选择N；二是选择N。 ; q，然后随机选择e满足通过命题10.15的算法计算d。这个随机指数RSA生成器，表示为，详细如下：



在输入M上运行命题10.15的算法计算d;e。

返回（（N,e）,(N,p,q,d)）

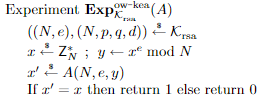
为了加速的计算，经常取e很小。 为此，我们首先将e设置为一些较小的质数（例如3），然后适当选择其他参数。 特别是，我们将以下指数e RSA生成器与任何奇质数e相关联：



### 10.5.4单向性问题

RSA函数的基本假定安全属性是单向的，意味着给定N； e; y很难计算但是，必须谨慎地将其正确形式化。 形式化随机选择y。

定义10.17令为具有相关安全性参数k的RSA生成器，令A为算法。 我们考虑以下实验：

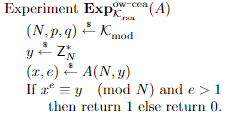


A的被定义为：



在上方，“ kea”代表“已知指数攻击”。 我们还可能允许选择指数的攻击，简称为\cea，在这种攻击中，不是让问题实例指定加密指数，而是让对手选择它。唯一的条件是对手没有选择 e = 1。

定义10.18 令为具有相关安全参数k的模数生成器，令A为算法。 我们考虑以下实验：



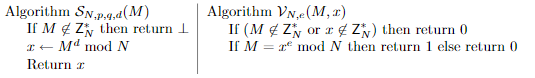
A的被定义为：



### 10.5.5陷门签名

陷门签名代表尝试基于RSA的单向性进行签名的最直接方法。 我们认为签名者拥有秘密密钥N； d，是唯一可以计算RSA逆函数的函数。对于其他任何人，仅知道公钥N； e，此任务在计算上不可行。 因此，签名者通过在其上执行此“困难”操作来对消息进行签名。这需要消息是的一部分。可以通过对所声明的签名执行计算的“轻松”操作来验证签名，并查看我们是否获得了该消息。

更准确的将，让成为一个RSA发生器与相关安全参数k，就像定义10.16定义的那样，假定数字签名方法为，其签名和验证算法如下：



这是确定性的无状态方案，公钥（N;，e）的消息空间为Messages（N;，e）= ，这意味着签名者签名的唯一消息是属于组的元素的消息。 在该方案中，我们用x表示M的签名。 签名算法仅将应用于消息以获取签名，而验证算法将应用于签名并测试结果是否等于消息。

首先要检查的是，签名算法生成的签名通过了验证测试。 由于命题10.14，这是正确的，它告诉我们，如果

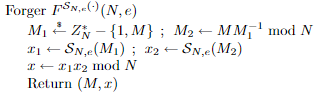
现在，该方案的安全性如何？如前所述，直觉是签名操作应该是签名者只能执行的事情，因为在没有d知识的情况下很难计算。 但是，应该记住的是，如定义10.17中所规定的那样，RSA的正式假定硬度属性（即已知指数攻击下的单向性（此后称为单向性））在一个非常不同的模型和设置下， 签名的安全性。 单向告诉我们，如果我们随机选择M，然后将其提供给对手（谁知道N； e但不知道d），然后要求后者，那么对手 成功将很难。 但是，签名方案中的对手没有获得在其上伪造签名的随机消息M。 而是，其目标是创建一个对（M; x），使。它不必尝试模仿签名算法，而是必须要做一些什么来满足验证算法。特别地，允许选择M而不是必须对给定的或随机的M进行签名。还允许在除消息最终通过签名Oracle输出的M之外的任何消息上获得有效签名，在这种情况下，该消息对应于具有 的oracle。 这些功能使对手很容易伪造签名。

下面说明了几种简单的锻造策略。 首先是简单地输出将消息和签名都设置为1的伪造品。第二步是随机选择一个将充当签名的值，然后根据该值计算消息：



这些伪造者对他们的签名神谕毫无疑问。我们注意到，因此F1的uf-cma-advantage是1。同理，值(M;x)返回的第二个满足,因此它也uf-cma-advantage 1。两种情况下的时间复杂度都很低。(在第二种情况下，伪造者使用O(k3)时间来做取幂模n)所以这些攻击表明这个方案是完全不安全的。

上面的伪造者设法伪造的签名的消息M是随机的。 根据我们对安全性的定义，这足以破坏该方案，因为我们对安全性做了非常强的定义。 实际上，对于这种方案，甚至可以伪造给定消息M的签名，但是这次必须使用签名预言。 攻击依赖于RSA函数的乘法性。



给定M，伪造者想要为M计算一个有效的签名x。 如图所示，M2，并获得其签名x1; 2倍 然后设置。现在验证算法将检查。但是请注意：



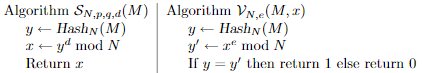
在这里，我们使用了RSA函数的乘法性，并且对于i = 1; 2，xi是Mi的有效签名。 这意味着x是M的有效签名。由于选择的M1不是1或M，所以M2也是如此，因此M不是F的预言查询。因此F以概率1成功。

这些攻击表明，签名不仅是基础功能的单向性。

### 10.5.6散列反转范式

现实世界中基于RSA的签名方案需要克服上述攻击，同时也要注意到陷阱门设置的其他缺陷。特别是，消息通常不是组元素;它们可能是长文件，意味着任意长度的字符串。这两个问题通常都是通过一个散列函数对给定的消息M进行预处理，以在范围内生成一个点y，然后应用到y来获得签名。哈希函数是公共的，这意味着它的描述是已知的，任何人都可以计算它。

为了更精确，让成为一个RSA生成器，并与相关的安全参数k相关联，让密钥成为输出的所有具有正概率的模N的集合。假设哈希是一组函数，其键空间是键，并且每个N 2个键对应Z§N。令为数字签名方案，其签名和验证算法如下:



让我们看看为什么这可能有助于解决陷阱门签名的弱点，以及安全性对哈希函数施加了哪些要求。

让我们回到上面提到的针对活板门签名方案的攻击。提出的伪造者,只是输出(1;1)。这是对我们新计划的攻击吗?告诉,我们看看会发生什么当上述鉴别算法调用输入1;1。我们看到，只有当时，它才返回1。第二个伪造者，我们之前已经把随机的。在井式反转模式下，这个策略的成功概率是多少?如果xe mod N = Hash(M)(而不是像以前那样仅仅对，那么伪造者就赢了。希望通过一个好的哈希函数，x e mod N = HashN(M)是不太可能的。现在考虑我们上面提出的第三种攻击，它依赖于RSA函数的可乘性。要使这种攻击在散列然后反转模式下工作，它必须为真。



同样，使用“好”散列函数，我们希望这不大可能是真的。

因此，哈希函数应该“破坏”使上述攻击成为可能的代数结构。我们如何找到做到这一点的方法，我们尚未解决。

虽然哈希函数可以防止在陷门方案上工作的一些攻击，但它的使用导致了基于哈希函数中的冲突的新攻击路线。如果敌人可以M1Ønd两个不同的消息;散列到相同值的M2，即，则很容易伪造签名，如下所示:



这是因为M1； M2具有相同的签名。 即因为x1是M1的有效签名，并且因为M1； M2具有相同的哈希值，我们有



这意味着验证过程将接受x1作为M2的签名。 因此，对散列函数Hash的必要要求是它是，这意味着在给定N的情况下，对于第n个不同的值M应该不可行； M 0使得。

下面，我们将继续介绍散列反转范式的更具体的实例。 但是在我们这样做之前，重要的是尝试评估我们到目前为止所做的事情。 上面，我们指出了哈希函数的某些特征，这些特征对于签名方案的安全性是必需的。 耐碰撞性是其中之一。 另一个要求的表述方式不是很好，但是大致上我们希望以这样的方式破坏代数结构，例如，公式（10.1）应该很可能失败。 经典设计着眼于这些攻击以及哈希函数的关联功能，旨在实现合适的哈希函数。 但是，如果您一直在理解本节课和笔记中我们一直在努力开发的方法和观点，那么您应该有一个更为批判的观点。 需要注意的关键点是，我们真正需要的并不是真正确定哈希函数的必要功能以防止某些攻击，而是真正确定哈希函数的足够的功能，即足以防止的功能。 所有攻击，甚至尚未构思出来的攻击。 而且我们还没有做到这一点。 当然，固定散列函数的必要特征对于收集关于可能具有足够特征的直觉很有用，但是仅此而已，我们必须小心，不要被引诱认为这已经足够了， 确定了所有问题。 实践证明，这种自满是一次又一次的错误。

我们如何希望做得更好？ 回到可证明安全性的基本理念。 我们要确保签名方案在其基础原语是安全的假设下是安全的。 因此，我们必须尝试将签名方案的安全性与作为单向函数的RSA的安全性以及哈希函数的某些安全性条件联系起来。 考虑到这一点，让我们继续研究一些建议的解决方案。

### 10.5.7PKCS #1 方案

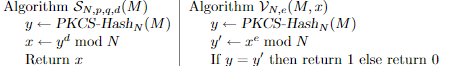
RSA公司一直是基于RSA的加密技术的软件和标准的主要来源之一。 RSA 实验室（现在是动态安全公司的一部分）已经创建了一套称为PKCS （公钥密码术标准）的标准。 PKCS＃1与基于RSA功能的签名（和加密）方案有关。 该标准得到了广泛的使用，因此可以看到它们的作用。

该标准使用散列-然后反转范式，通过我们现在描述的特定散列函数PKCS-Hash实例化散列。回想一下，我们已经讨论了抗碰撞哈希函数。让我们填充函数，其中，其“抗腐蚀性”在某种意义上,没有人知道如何找到任何一对不同的点M，使得，目前角色倾向于SHA-1，所以l = 160。之前是MD5，它的l = 128。RSA PKCS # 1标准如下定义：



在此，|| 表示串联，并且插入了足够的FF字节，以使PKCS-HashN（M）的长度等于k位。 请注意，哈希输出的第四个位为零，这意味着作为整数，它肯定最多为N，因此很有可能在中，因为1和N之间的大多数数字都在中。 还要注意，在PKCS-Hash中的查找冲突并不比在h中的查找冲突容易，因此，如果后者是耐碰撞的，则前者也是如此。

回想一下，签名方案与散列然后反转范式完全相同。具体来说，我们重写一下签名和验证算法:



现在该签名方案的安全性如何？我们最关心的是我们在陷门签名上看到的各种代数攻击。如10.5.6节所述，我们希望像公式（10.1）这样的关系失败。 我们似乎得到了； 很难想象如何具有使其看起来像某些消息的PKCS哈希所需的特定结构。 当然，这并不是不可能进行攻击的证据，但至少还不是显而易见的。

这就是我们的方法与经典的基于攻击的设计方法不同的地方。 在后者的情况下，上述方案是可以接受的，因为已知的攻击会失败。 但是更深入地看是值得关注的。 我们要采用的方法是查看签名方案的所需安全性如何与基础原语（在本例中为RSA函数）的假定或理解的安全性相关。

我们假设RSA是单向的，这意味着为随机选择的点计算在计算上是不可行的。 另一方面，在签名方案中应用; e的点是集合中的点。 SN的大小最大为，因为h输出1位，而的其他位则固定。 对于SHA-1，这意味着。 这似乎是一个很大的集合，但是在RSA域中，它很小。 例如，当k = 1024（这是目前安全参数的建议值）时，我们有：



这是从中随机选择的点落入SN的概率。 实际上，它为零。 因此，RSA很可能是单向的，并且仍然很容易在SN上反转，因为随机点降落在SN中的机会非常小。 因此，不能仅在RSA的标准单向假设下才能保证PKCS方案的安全性。 请注意，无论底层的哈希函数h（在本例中为SHA-1）多么“好”，这都是正确的，它构成了PKCS-Hash的基础。问题在于PKCS-Hash本身的设计，尤其是填充。

PKCS签名方案的安全性要求假设RSA很难在集合SN上反转，而集合SN只是其全部范围的极小一部分。(即使这样，这也只是签名方案安全性的必要条件，而不是先决条件。)

让我们尝试澄清和强调这里采取的观点。 我们并不是说我们知道如何攻击PKCS方案。 但是我们要说的是，缺少已知攻击不应被认为是对该方案感到满意的充分理由。 我们可以确定“设计粘度”，例如该方案使用RSA的方式，这不符合我们对RSA作为单向函数的安全性的理解。这令人关注。

### 10.5.8 FDH方案

从上面我们可以看到，如果哈希-然后-反转范式要产生一个签名方案，其安全性可以基于RSA函数的单路性，那么方案中应用的点y必须是随机的。换句话说，哈希函数的输出必须看起来总是随机的。然而，即使这样，也只是强调了一个必要条件，而不是(据我们所知)一个条件。

我们现在问自己以下问题。 假设我们有一个“完美的”哈希函数Hash。在这种情况下，至少，哈希-然后-反转签名方案是安全的吗？要解决这个问题，我们必须首先确定什么是“完美的”哈希函数。 答案是很自然的：一个随机的答案，即对任何查询返回随机答案，但与过去的查询保持一致。（稍后，我们将解释“随机甲骨文”的工作方式，但现在让我们继续。）因此，我们的问题变成了：在Hash完美的模型中，如果RSA是单一的，我们可以证明签名方案是安全的吗？ 道路？

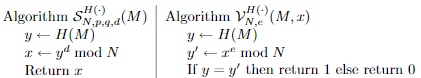
确实，这是一个基本问题。 如果“哈希-然后-反转”范式无论如何可行，那么在哈希函数完美的情况下，我们确实必须能够证明安全性。 如果无法在此模型中证明安全性，那么采用“哈希-然后-反转”范式将是极其不明智的； 如果它不适用于完美的哈希函数，那么我们如何期望它在任何现实环境中都能正常工作？

因此，我们现在集中于“思想实验”，涉及使用具有完美哈希函数的签名方案。这是一个思想实验，因为没有特定的哈希函数是完美的。我们的“哈希函数”不再是固定的，只是 一个∞ips硬币的盒子。 然而，这种思想实验对于我们的签名范式的安全性有重要的说法。 这不仅是我们理解的关键步骤，而且将引导我们制定更好的具体方案，如我们稍后所见。

现在让我们更多地谈谈完美的哈希函数。 我们假定Hash每次被调用时都返回的随机成员，除了如果对同一消息两次调用时，它两次都返回相同的东西。 换句话说，它是具有域和范围的随机函数的一个实例。 在研究伪随机性之前，我们已经看到过这样的对象：请记住，我们通过考虑涉及随机函数的实验来定义伪随机函数。 因此，这个概念并不新鲜。 我们将Hash称为随机预言，在这种情况下用H表示。 所有各方，签名者，验证者和对手都可以使用它，但是它是一个预言家。 这意味着只能通过指定的接口访问它。 要计算H（M），一方必须进行甲骨文呼叫。 这意味着它输出M以及一些指示它希望返回H（M）的指示，并返回适当的值。 具体来说，它可以输出一对（哈希； M），Ørst组件仅仅是用于表示这是一个哈希-甲骨文查询的形式符号。 输出此内容后，调用算法将等待答案。 返回值H（M）后，它将继续执行。

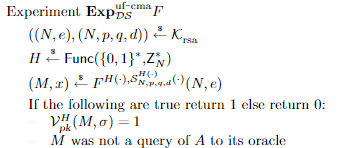
考虑H的最佳方法是作为一个动态过程，该过程维护一个输入输出对表。 每次查询（hash; M）时，处理第一个验证的过程都会检查其表是否包含一对y形式的（M; y）对，如果是，则返回y。 否则，它在中选择一个随机y，将（M; y）放入表中，并返回y作为甲骨文查询的答案。

我们在模型中考虑了上面的哈希然后反转签名方案，其中哈希函数Hash是随机预言H。这称为全域哈希（FDH）方案。 更准确地说，让Krsa是具有关联安全性参数k的RSA生成器。 与Krsa相关的FDH-RSA签名方案是数字签名方案，其签名和验证算法如下：



唯一的变化对我们写的算法的通用哈希反转方案部分10.5.6符号:我们写H作为上标,以表明它是一个甲骨文只能通过特殊的甲骨文接口。指令是通过查询(散列;M)，让y表示返回的答案，如上所述。

现在我们问自己，在RSA是单向的假设下，上述签名方案是否安全。 为了考虑这个问题，我们首先需要扩展我们的定义以包含新模型。 关键区别在于，除了先前考虑的随机选择之外，对手的成功概率还取决于H的随机选择。 伪造者F像以前一样可以访问签名预言，但现在也可以访问H。此外，S和V现在可以访问H。让我们首先介绍测量伪造者F成功的实验，然后再进行讨论。



请注意，除了通常对模拟选定消息攻击的符号甲骨文的访问之外，还向伪造者授予甲骨文对H的访问权限。 在查询其预言数次后，伪造者为其输出消息M和候选签名x。 我们说，如果验证过程接受M，则F成功。 x，但F从未要求签名预言家对M进行签名。（当然可以允许F进行哈希查询M，并且确实很难想象它可能希望如何成功地进行伪造，但是不允许进行签名查询。 M.）A的优势定义为：



我们将要考虑时间复杂度最高为t的对手，最多进行符号甲骨文询问和最多 哈希甲骨文询问，并且查询消息的总长度为π。 资源再次引用整个实验的资源。 我们将执行时间定义为整个实验所花费的时间。这意味着它包括计算对甲骨文查询的答案，生成密钥甚至验证伪造品的时间。然后，时间复杂度t被认为是执行时间的上限加上F代码的大小。在对哈希查询进行计数时，我们再次查看整个实验，并要求对H的查询总数最多为。 计数中包括F的直接哈希查询，由签名oracle进行的间接哈希查询，甚至是在最后一步中由鉴别算法进行的哈希查询。 后者意味着始终至少是验证所需的哈希查询数，而对于FDH-RSA而言，这是一个。 实际上，对于FDH-RSA，我们将具有，在解释以后的结果时要牢记一些。 最后，π是符号查询中所有消息的长度加上最终输出消息M的长度之和。

然而,这里有一个点需要辨别清楚,即如果时间复杂度是指整个的实验,我们如何衡量时间随机选择H ?它实际上是一个无限对象,因此不能在有限时间选择。答案是，尽管我们在实验中把H写成是随机选择的，但这并不是它的实现方式。相反，假设H是动态选择的。请考虑实现我们所描述的表的过程，以便只在调用H ora时进行随机选择，而成本是维护和更新到目前为止查询的输入中包含H值的表。即当一个查询M H,我们收取的费用表,检查H (M)是否已经定义并返回它如果是这样,其他的选择一个随机点从,把它的表和索引,并返回它。

在这种情况下，我们声称FDH-RSA方案是安全的。 以下定理上限仅根据底层RSA生成器的低优先级优势限制了其优势。

理论10.19 令Krsa为具有相关安全性参数k的RSA生成器，令DS为与Krsa相关联的FDH-RSA方案。 假设F是一个攻击者，最多对它的哈希甲骨文进行qhash查询，并且对它的签名Oracle进行最多qsig查询，其中。 然后有一个攻击者满足：



并且I，F是可以进行比较的资源。

定理说，在FDH-RSA方案中伪造签名的唯一方法是尝试在随机点上反转RSA函数。 安全性方面有所损失：在伪造实验中进行哈希查询的次数的一个因素可能是，破坏签名方案的机会大于在可比的时间内反转RSA的机会。 但是，通过选择较大的模量k，可以使足够小，甚至也很小。

必须记住警告:这是在散列函数是随机的模型中。然而，即使这样也告诉了我们一些东西，即散列然后反转范式本身是合理的，至少对于完美的“散列函数”来说是如此。这使我们能够更好地探索范例的具体实例。

现在让我们继续进行定理10.19的证明。 请记住，逆变器I用作输入（N; e），描述了，还有点。 它的工作是尝试输出，其中d是与加密指数e对应的解密指数。 当然，d或N的因式分解都不可用于I.I的成功是根据Krsa给出的（（N; e）;（N; p; q; d））的随机选择来衡量的 从随机选择y。 为了完成其任务，我将在输入的公共密钥（N; e）上将F作为子例程运行，希望以某种方式利用F的能力将签名伪造为。 在讨论如何希望使用伪造函数确定点y的倒数之前，我们需要仔细研究将F作为子例程运行的含义。

回想一下，F可以访问两个神谕，并调用它们。在执行的任何时候，它都可能输出(散列;然后，它将等待一个返回值，并将其解释为H(M)。一旦接收到，它将继续执行。类似地，它可能输出(符号;然后等待接收一个它解释为的值。需要理解的重要一点是，F作为一种算法，仅仅是通过接口与神谕进行通信。它无法控制这些神谕会返回什么。您可以将甲骨文查询视为系统调用。认为F是编写一个甲骨文查询M在某些特殊规定地方在内存中。某些过程被期望在另一个指定的地方放置一个F作为答案的值。F读到这里，然后继续。

当我执行F时，实际上没有甲骨文。 F不知道。 假设存在预言，它会在某个时候进行预言查询，例如查询（哈希； M）。 然后，它等待答案。 如果我想运行F来完成，则由我来提供F的答案作为此甲骨文查询的答案。 F将接受给出的所有内容并继续执行。 如果我无法提供答案，F将不会继续运行； 它会坐在那里，等待。 我们已经在一些证明中看到了“模拟”的思想：我正在创建一个“虚拟现实”，在这种情况下F可以相信自己处于其通常的环境中。

我的策略是利用其对甲骨文查询响应的控制。 它将以奇怪的方式选择它们，而不完全是在实验中选择它们的方式。由于F只是一种算法，它会处理收到的所有内容，最终会因某些输出而中止，声称是伪造的（M ; X）。 通过明智地选择对oracle查询的答复，我将确保F被欺骗，以至于不知道在中它并不是真的，而且x将是y的期望逆。 并非总是如此；我一定很幸运 但是经常会很幸运。

我们首先考虑一个非常简单的伪造F的情况。它不进行符号查询，而仅进行一个哈希查询（哈希； M）。 然后，它输出一对（M; x）作为要求的伪造品，消息M在哈希查询和伪造品中是相同的。 （在这种情况下，我们有qsig = 0和qhash = 2，最后一个是由于F的哈希查询和实验中的查询。）现在，如果F成功，则x是M的有效签名，表示，或者等价为。不知何故，F找到了H（M）的逆数，返回的值作为对甲骨文查询M的响应。现在记住 我的目标是计算，其中y是它的给定输入。 一个自然的想法表明了自己：如果F可以将转换为H（M），那么我将“ H（M）设置为y，从而在下获得y的倒数；e。我可以将H（M）设置为 ），因为它控制着甲骨文查询的答案。当F发出查询（hash，M）时，反转I将会简单地返回y作为响应，如果F输出一个有效的（M，x），我们有，，并且I可以输出x，然后他的工作就完成了。

但是，为什么F在得到y作为对哈希查询M的响应时会返回有效的伪造？ 也许它会拒绝这样做，说它不能在逆变器I提供的点上工作。但这不会发生。 F只是一个算法，并且可以根据给出的结果进行运算。 重要的是仅响应的分布。 在实验中，对（hash; M）的响应是的随机元素。 但是y具有完全相同的分布，因为这是在实验中选择它的方式，从而确定了I成功打破RSA作为单向函数的成功。 因此，F在这个虚拟现实中的行为与现实世界中的行为不同。 它返回有效伪造品的概率仍然是。因此，对于这个简单的F，逆变器在中的成功概率与F在伪造签名中的成功概率完全相同。 式（10.2）要求较少，因此我们当然满足。

但是，大多数伪造者都不愿意进行任何符号查询，而只进行一个包含他们伪造的消息的散列查询。我必须能对付任何伪造者。

逆变器我将定义一对子例程H-Sim（称为哈希甲骨文模拟器）和S-Sim（称为符号Oracle模拟器），分别发挥哈希和符号oracle的作用。 即，每当F进行查询（哈希； M）时，逆变器我将H-Sim（M）返回给F作为答案；每当F进行查询（符号； M）时，逆变器我将返回S-Sim（M） ）以F作为答案。S-Sim例程将另外调用H-Sim。）在执行时，我将建立定义 H的各种表（数组）。对于，第j个字符串 在实验中调用哪个H（直接由于F的哈希查询，间接由于F的符号查询或最终验证查询）将被记录为Msg [j]；哈希返回的响应 甲骨文模拟器对Msg [j]的存储为Y [j]；如果Msg [j]是一个符号查询，则以“签名”形式返回给F的响应为X [j]。 现在的问题是我如何定义所有这些价值。

假设实验中的第j个哈希查询是F进行的符号查询（sign; Msg [j]）的结果。在实验中，伪造者将返回H（Msg [j] ）d modN。如果我想让F继续运行，它必须返回合理的值。 我能做什么？ 它可以尝试直接模仿签名过程，将Y [j]设置为随机值（记住Y [j]扮演H（Msg [j]）的角色）并返回（Y [j]）d modN。 由于它不属于秘密签名指数d，因此它将无法计算后者。 技巧是，I首先在中随机选择一个值X [j]并将Y [j] =（X [j]）e mod N设置为零。现在它可以返回X [j]作为答案 到符号查询，这个答案在某种意义上是正确的，因为检验关系（F可能会检查）成立：我们有。

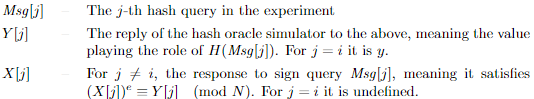
这留下了一些松散的末端。 一种是，我们上面假设，在进行符号查询时，我有权拒绝Y [j]。 但是，由于过去存在涉及该相同消息的哈希查询，因此对于某些<<j来说，Msg [j] = Msg [l]。 然后，哈希值Y [j]已被定义为Y [l]，并且无法更改。 然而，这可以很简单地解决：对于任何哈希查询Msg [l]，哈希模拟器可以遵循上述策略，即在哈希查询为时设置回复。 这意味着它为Msg [l]之后成为符号查询的可能性提前做好了准备。 也许不会，但是什么也不会丢失。

好吧，差不多。 实际上有些东西丢失了。 迄今为止一直保持清醒状态的读者可能会注意到我们已经解决了两个问题：如何使用F到，其中y是I的输入，以及如何模拟对F的符号和哈希查询的答案，但是 这些过程是矛盾的。 我们得到的方法是返回y作为查询的答案（哈希； M），其中M是伪造中的消息。 但是，我们事先不知道哈希查询中的哪个消息将是伪造的消息。 因此，很难知道如何回答哈希查询Msg [j]； 我们返回y，还是为某些X [j]返回（ 如果执行了第一步，我们将无法通过消息Msg [j]来回答符号查询； 如果我们进行第二次处理，并且如果Msg [j]等于伪造中的消息，则不会发现y的倒数。 答案是猜测该怎么做。 这种猜测很有可能是正确的，在这种情况下，我会成功。

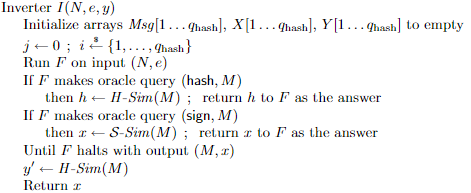
特别要注意的是，由于是最终验证查询中的消息，因此Msg [qhash] = M是伪造的消息。 消息M在列表中可能会出现多次，但是至少会出现一次。 现在，我将在范围内选择一个随机i，并通过y响应哈希查询（hash; Msg [i]）。 对于所有其他查询j，它将首先通过在中随机选择X [j]并将设置为响应。伪造的消息M将等于Msg [i ]的概率至少为，这将意味着公式（10.2）。下面我们总结这些思想作为定理10.19的证明。

从上面的描述中，很容易建议我们总是选择i = qhash，因为按照定义，Msg [qhash] =M。 为什么不行呢？ 因为对于某些j <qhash，M可能也等于Msg [j]，如果我们将i = qhash设置为零，那么在我们想返回y作为M的答案时，我们已经确定H（M） 作为其他事情，现在改变我们的想法为时已晚。

理论10.19的证明：我们首先用两个子例程来描述I：一个哈希预言模拟器和一个符号预言模拟器。 它需要输入N， e，y，其中并维护三个表Msg，X和Y，每个表的索引范围为1到qhash。 它选择一个随机索引i。 所有这些都是全局变量，也将用作子例程。 数组项的预期含义如下，其中



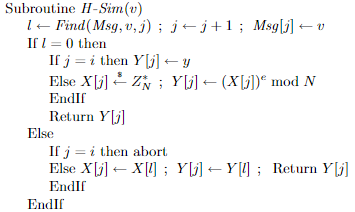
翻转器的代码如下：



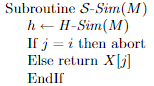
逆变器通过使用适当的子例程来响应甲骨文查询。 一旦声明了伪造，它将进行相应的哈希查询，然后返回签名x。

现在，我们描述哈希甲骨文模拟器。它引用在I的主代码中实例化的全局变量。它以值v作为参数，该值只是一些消息，其哈希直接由F或由下面的符号模拟器（由F调用后者时）请求。

我们将利用一个子程序发现，给定一个数组a，一个值v和索引m，如果就返回0，否则返回最小的索引l，使得v = A[l]。



回答散列查询的方式支持以下符号模拟器。



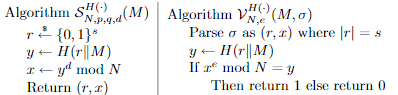
由于任一子例程中的“ abort”指令，我都可能会中止执行。最先出现这种情况是哈希甲骨文仿真器无法返回y作为对第i个哈希查询的响应，因为该查询等于先前对查询的答复 第二种情况是F要求消息的签名是第i个哈希查询，但我无法提供该信息，因为它希望第i条消息是伪造的消息，并已将y作为哈希返回 甲骨文回应。

现在我们需要相对于Krsa下限I的低优势。 验证方程式（10.2）中的界限时涉及一些观察。 首先，在我没有中止的任何时候，F的“view“都是”与实验中的“相同”。这意味着我返回给F的答案将完全按照它们的分布 在真正的实验中。 其次，F没有获得有关我随机选择的值i的信息。 现在请记住，我所做的最后一个哈希模拟器查询是伪造中的消息M，因此在执行I的末尾M肯定在数组Msg中。令为第一个出现M的索引，表示Msg [l] = M，但没有先前的消息为M。则i的随机选择意味着i = l的概率为1 = qhash，这反过来意味着Y [i] = y并且哈希甲骨文模拟器不会中止。 如果x是M的正确签名，因为从F的角度来看Y [i]是H（M），所以我们将得到（mod N）。因此，只要发生这种情况，我就成功了。

### 10.5.9 PSS0:在安全性的提升

FDH-RSA签名方案具有有吸引力的安全属性，即使在随机预言模型中，在RSA是单向函数的前提下，也具有安全性证明。 然而，定理10.19给出的定量安全性可能更好。 该定理使人们有可能伪造签名，其概率是qhash乘以能够在随机点处反转RSA功能的概率的概率，这两个动作是针对具有可比执行时间的对手进行衡量的。 由于qhash可能非常大，例如，所以这里的安全性有相当大的损失。 现在，我们提出一种安全关系更为严格的方案：签名伪造的可能性并不比能够在可比的时间内反转RSA的可能性高得多。

该方案称为“概率签名方案版本0”的PSS0，以强调它的一个关键方面，即它是随机的：签名算法每次被调用时都会选择一个新的随机值，并使用该随机值来计算签名。 像FDH-RSA一样，方案利用公共哈希函数建模为随机预言机，此外，它的参数s为 签名算法选择的随机值的长度，我们编写签名和验证算法如下：



明显“范围检查”中为简单起见没有写明确验证代码;例如，在实际的实现中，应检查和。

除了通过签名算法选择的值将哈希随机化之外，该方案仍可以视为哈希反转范式。如果您两次签名同一条消息，则可能会获得不同的签名。注意随机值r一定要被包含在签名里面，因为否则的话它将不能够验证签名。因此，与以前的方案不同，签名不是的成员； 它是一对，其中一个是s位字符串，另一个是的成员。 签名的长度为s + k位，比确定性散列然后反转签名方案的签名长一些。 通常将l设置为160，并且假设k可以为1024，则长度增加是可以容忍的。

通过实验在随机预言模型中测量了伪造F攻击DS的成功概率。即，该实验与FDH-RSA情况下的实验相同； 现在，我们插入的方案DS只有上面的方案。 因此，我们具有与该方案相关联的不安全功能。 现在我们可以总结PSS0方案的安全性属性。

理论10.20令DS为具有安全参数k和s的PSS0方案。 令F为进行qsig签名查询和哈希甲骨文查询的对手。 然后有一个对手I满足下列条件：



运行时间I时F加上。

有，当l = 160时，上述加法项约为，这非常小。 因此，出于所有实际目的，可以忽略加法项，并且PSS0签名方案的安全性与RSA的安全性紧密相关。

我们进行定理10.20的证明。 I的设计遵循定理10.19证明中使用的相同框架。 即I，在输入N上； e， y，将在输入N，e上执行F，并回答F的甲骨文查询，以便F可以完成其执行。 通过伪造，我将以某种方式。我将通过称为Hash 甲骨文模拟器的子例程H-Sim来响应F的哈希甲骨文查询，并将通过称为S的子例程S-Sim来响应F的符号查询。 甲骨文模拟器。 设计的很大一部分是这些子例程的设计。 为了获得一些直觉，回到定理10.19的证明是有帮助的。

我们看到，在该证明中，方程式（10.2）中qhash的乘法因子来自于我随机猜测的值，并希望其中M是伪造邮件。 也就是说，它必须猜测从哈希预言中首先查询伪造消息的时间。 关于正确猜对的机会，我们能说的最好的是，它至少为1 = qhash。 但是，如果现在我们希望我的成功概率如公式（10.3）所示，我们就无法猜测哈希散列查询伪造消息的时间。 但是，我们当然不提前知道。 无论如何，我仍然必须能够利用伪造来返回。

一个简单的想法是返回y作为所有哈希查询的答案。 然后，对查询消息的伪造肯定会产生所需的值。考虑FDH的这种策略。在那种情况下，出现两个问题。 首先，这些答案将不会像哈希查询答案所要求的那样是随机的和独立的。 其次，如果哈希查询中的消息以后是符号查询，那么我将无法回答符号查询。 （请记住，我为计算了其对哈希查询Msg [j]的答复为，以便以后在Msg [j]显示为X时返回X [j]。 符号查询。但是这里有一个矛盾：我可以执行此操作，或者返回y，但不能同时返回两者。它必须选择，在FDH情况下，它是随机选择的。）

实际上，第一个问题很容易通过一个小的代数技巧来解决，即利用RSA的自约性。 当我想返回y作为哈希甲骨文查询Msg [j]的答案时，它将在中选择一个随机X [j]并返回。 每次随机且独立地选择值X [j]。 现在，是一个置换这一事实意味着，所有不同的Y [j]值都是随机且独立分布的。 此外，假设（M;（r; x））是一个伪造品，对其进行了哈希预言查询并获得了响应。 ，因此y的逆是。

但是，FDH无法解决第二个问题。 这就是为什么PSS0在哈希之前将随机值r附加到消息之前。 这有效地“分离”了两种哈希查询：F对哈希oracle的直接查询，以及对由符号oracle引起的对哈希oracle的间接查询。直接哈希oracle查询的形式对于某些L位为。 字符串r和一些消息M.符号查询只是一条消息M，为了回答它，首先随机选择了一个值，然后值成为先前的哈希查询的可能性很小。 进行了新的直接哈希查询，我可以假设它永远不会是间接哈希查询，因此可以通过上述技巧进行回复。下面是全部的证明：

理论10.20的证明：我们首先用两个子例程来描述I：一个哈希预言模拟器和一个符号预言模拟器。 它需要输入N，e， y其中，并维护四个表R，V，X和Y，每个表的索引范围为1到qhash。 所有这些都是全局变量，也将用作子例程。 数组项的预期含义如下，其中

V [ j ] 实验中的第j和hash查询，形式是。

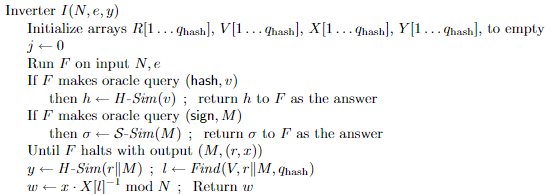
R [ j ] V [ j ]的前l个比特。

Y [ j ] 的值，要么是通过哈希模拟器选择的，要么是符号模拟器选择的。

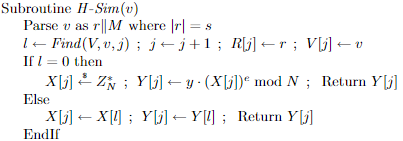
X [ j ] 如果V [ j ]是满足的直接哈希甲骨文查询F。如果V [ j ]是满足的非直接哈希甲骨文查询，这就表示他是Msg[ j ]的签名。

注意，我们实际上不需要存储Msg数组。 它仅在术语解释中在上文中提及。

我们将使用一个子例程Find：给定一个数组A，一个值v和一个索引m，如果 则返回0，否则返回最小索引l，以使v = A [l]。

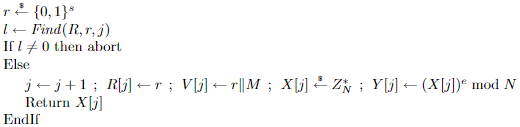


现在，我们描述哈希甲骨文模拟器。 它引用了在I的主代码中实例化的全局变量。它以一个值v作为参数，该值假定至少为s位长，这意味着的形式为s位强r。 （由于哈希查询与签名方案无关，因此无需考虑这种形式的哈希查询。）



该例程将查询的哈希甲骨文的每个字符串v放入表V中，因此V [j]是执行F时的第j个哈希甲骨文查询。以下符号模拟器不会调用哈希模拟器，但是 如有必要，请在必要的表中填写。

子例程S-Sim (M)

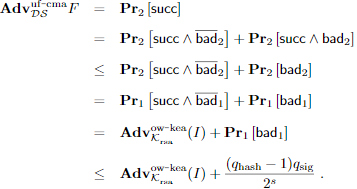


现在我们需要建立方程（10.3）

首先考虑，让表示这个实验中的概率函数。假设bad1是由于信号oracle模拟器中的\abort”指令而终止的事件。

现在考虑，并让表示此实验中的概率函数。 假设bad2是一个事件，即符号甲骨文选择了一个值r，以使F先前对某个M进行了哈希查询。

让succ为F成功进行伪造的事件（在任何一个实验中）。 现在我们有：



这就建立了方程(10.3)。现在让我们对上述问题做一些解释。

首先，等式（10.6）如下。 如果在符号预言机模拟器中选择的随机值r已经存在于集合fR [1]中，则发生所讨论的事件。 在查询符号时，此集合的大小最大为，因此r落入其中的概率最大为。 符号oracle模拟器最多被调用qsig次，因此界限随之而来。

很容易想到，在我没有中止的任何时候，F的“视图”与实验中的F的视图“相同”。这是不正确的，因为它可以测试是否 还是不错的发生。 因此，我们在两款游戏中都考虑了不良事件，并注意：



理由如下。 请记住，I所做的最后一个哈希模拟器查询是，其中M是伪造中的消息，因此肯定在执行I的末尾位于数组V中。所以。我们知道符号模拟器未将放入V中，因为不允许F进行符号查询M。这意味着哈希Oracle模拟器已在上调用。 这意味着，因为这是哈希预言机模拟器选择其答复的方式。 伪造的正确性意味着，并且此处H值的作用由Y [l]扮演，因此我们得到。 解决这个问题得到，因此逆变器在返回时是正确的。

对于上述内容，可能值得增加一些警告。 很容易想到:



这意味着(10.3)式，但实际上更强。这不过是不正确的,因为不良事件和成功事件如不是独立的。

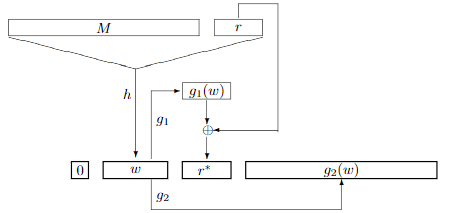
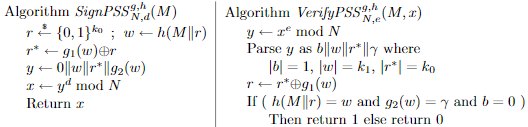


图10.1：PSS：图像的成分变暗。 M的签名为y d modN。

### 10.5.10概率签名方案——PSS

PSS0比FDH-RSA获得了更好的安全性，但代价是增加了签名大小。该方案减小了签名的大小，使其具有较高的安全性和与FDH-RSA相同的签名大小。这是[26]的概率签名方案(PSS)。

签名方案由k0和k1参数化，k0和k1是介于1和k之间的数字，满足。 具体来说，读者可能想像k = 1024，k0 = k1 =128。算法Krsa是RSA密钥生成算法，如10.5.3节所定义。 签名和验证算法使用两个哈希函数。 第一个h称为压缩器，映射为，第二个g称为生成器，映射为。 （分析认为这些是理想的。实际上，可以通过诸如MD5的加密哈希函数之外的简单方法来实现它们，如附录10.5.12所述。）令g1为输入返回的函数 g（w）的第一个k0位，令g2为在输入上的函数； 1g k1返回g（w）的剩余位。 现在，我们描述如何签名和验证。 有关图片，请参见图10.1。 我们编写签名和验证算法如下：



为简单起见，显然没有在检验代码中明确写明“范围检查”；例如，在实际实现中，后者应检查。

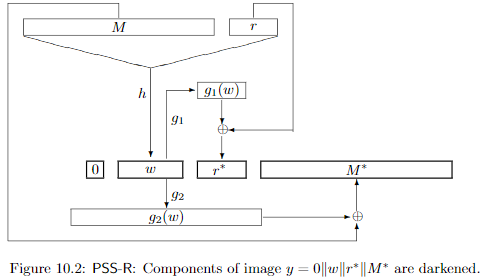
步骤指示签名者随机选择k0位的种子r。 然后，他将该种子连接到消息M，有效地“随机化”该消息，并通过“压缩”功能将此种子散列为k1位字符串w。 然后将生成器g应用于w，以产生k0位串和。 第一个用于对“ k0位种子r”进行“掩码”，得到被掩码的种子。现在前面加0位并附加g2（w）以创建图像点y，该点被解密 在RSA函数中用于拒绝签名（0位是为了确保y在中）。

注意，为每个消息选择了一个新的种子。特别是，给定的消息有许多可能的签名，这取决于签名者选择的r的值。

给定（M; x），验证计算mod N并恢复r§; w; 河 这些用于检查y是否正确构造，并且验证程序仅在所有检查成功的情况下接受。

注意，该方案的效率如所要求的。 签名采用h的一种应用，一种g的一种应用和一种RSA解密，而验证则采用一种h的应用，一种g的一种应用和一种RSA加密。

下面的定理证明了基于RSA单路性的PSS的安全性。两种安全性和PSS0相同。实质上是严格的，比我们在FDH方案中看到的要严格得多。 但是，这次是在不增加签名大小的情况下实现的。



理论10.21[26] 设DS为安全参数为k0和k1的PSS方案。假设F是一个对手，正在进行qsig签名查询和哈希甲骨文查询。那么就有了一个对手I满足：



I的运行时间是F加。

证明在[26]中。 它扩展了上面给出的定理10.20的证明。

### 10.5.11信息恢复签名——PSS-R

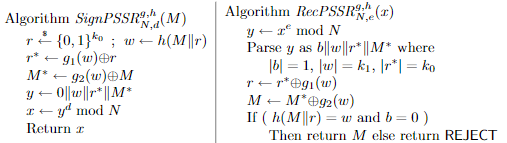
消息的恢复。在标准的签名方案的签名者传送消息M清晰,将签名x附在上面。在一个恢复方案提供消息,只有一个增强签名t传播。我们的目标是节省签名消息的带宽:我们希望这个增强签名的长度小于。(尤其是当M是短暂的,我们希望的长度øk,签名长度)。验证恢复·的消息M增强真实性同时签名和检查。

我们通过将消息的“一部分”“折叠”到签名中，以使验证者可以“恢复”。 当M的长度n小时，实际上我们可以将整个消息折叠到签名中，以便仅发送k位数。 在以下方案中，如果安全性参数为k = 1024，则我们最多可以将767个消息位折叠到签名中。

定义：在形式上，密钥生成和签名算法与以前一样，但是V被Recover代替，它接受pk和x，并返回。签收人拒收的标志是签收人拒收的标志;的返回值表明验证接受消息是真实的。安全性的公式是相同的，除了它对于伪造者的成功意味着什么:它应该提供一个x，使，其中M不是先前的签名查询。我们要求，如果x是通过生产的，那么回收率pk(x) = M。

一个简单的PSS变体实现了消息恢复。我们现在描述这个方案及其安全性。

方案：方案如前所述，由k0和k1参数化，密钥生成算法为Krsa，与以前相同。 与PSS一样，签名和验证算法取决于哈希函数和我们使用相同的g1和g2表示。为了简化说明，我们假设要签名的消息的长度为。 （建议的参数选择为k = 1024，k0 = k1 = 128和n =767​​。）在这种情况下，我们生成仅k位的“增强签名”，验证者可以从中恢复n位消息并同时检查真实性。签名生成和验证的过程如下：有关图片，请参见下图：



在SignPSSR中，与SignPSS的不同之处在于y的最后一部分不是g2（w）。 取而代之的是，使用g2（w）来“屏蔽”该消息，而被屏蔽的消息是图像点y的最后一部分。

上面的内容很容易处理任意长度的消息。 一个完全指定的方案将使用大约min位。

安全性：

PSS-R的安全性和PSS一样的。

理论10.22【26】设DS为具有安全参数k0和k1的恢复方案的PSS。假设F是一个对手，正在进行qsig签名查询和哈希甲骨文查询。那么就存在这样一个对手：



运行I的时间为F的时间乘以。

这个定理的证明与定理10.21的证明非常相似。

### 10.5.12如何执行哈希函数

我们已经讨论了PKCS标准[179，180]和ISO标准[1]，并发现基于RSA是单向陷阱的假设，无法证明其安全性。 其他标准，例如[9]，与[179]相似，并且适用相同的声明。

我们在本节的其余部分中讨论的模式不使用散列然后解密的范式。

基于RSA假设，其安全性可证明的签名方案包括[105，14，152，177，78]。这些作品的主要优点是，它们没有使用理想的哈希函数（随机奥数）模型|可证明的安全性是标准意义上的。另一方面，这些方案中的每个方案的安全性降低都相当宽松。在效率方面，[105, 14, 152, 177]的方案的效率太差了，无法认真考虑它们的实践。另一方面，Dwork-Naor方案[78]在计算上是相当的公平的，需要2到6次RSA计算，尽管有一些存储开销，而且签名的时间比单个RSA模数长。如果人们愿意允许一些额外的计算和存储，并且希望在不假设理想的散列函数的情况下获得合理的安全性，那么这种方案是目前最好的选择。

在假设理想散列的签名方案中，有许多基于因果关系的硬度或其他假设的签名方案已经被提出。这些方案大多是从 验证方案中衍生出来的，就像 第一个所做的那样[83]。这些方法中有些是可证明的（在理想的散列模型中），有些则不然。在一些被证明的方案中，有的方案的安全性是可以分析的，而通常情况下是不可以的。在我们所知道的情况下，没有一个方案是安全的。效率是不同的。计算要求通常比哈希然后解密的RSA签名要低，尽管密钥大小通常要大一些。

最后，我们注意到相关的新工作。 Pointcheval和Stern [165]考虑了随机预言模型中签名的可证明安全性，并表明El Gamal方案[90]和Schnorr [184]方案的改进版本可以被证明是安全的。 （[83]的方案可以证明对没有签名查询的攻击是安全的。）但是他们没有考虑确切的安全性。 一个有趣的问题是考虑并可能提高减排量的确切安全性（必要时对方案进行修改）。

最近，出现了一些基于RSA的非常简单的签名方案，这些签名方案基于关于RSA的更强和更少标准的假设而具有安全性证明，但它们不依赖于随机预言[92，65]。

## 10.6门限签名方案

使用阈值签名方案，数字签名可以由一组玩家而不是由一方产生。 与常规签名方案不同，在常规签名方案中，签名者是持有密钥的单个实体，在阈值签名方案中，密钥由一组n个玩家共享。 为了在给定消息m上产生有效签名，各个玩家在该消息上产生其部分签名，然后将它们组合为m上的完整签名。 如果没有t（或更少）玩家的联盟可以产生新的有效签名，即使系统在不同消息上产生了很多签名之后，分布式签名方案也会达到阈值t <n。 由阈值签名方案产生的签名与由拥有完整秘密签名密钥的单个签名者产生的签名相同。 特别是，具有相应唯一公共验证密钥的任何人都可以验证此签名的有效性。 换句话说，以分布式方式产生签名的事实对于签名的接收者是透明的。

阈值签名的动机是出于某些组织的需要，即要求一组员工在给定消息（或文档）签名之前就给定的消息（或文档）达成一致，以及需要保护签名密钥不受内部和外部的攻击 对手。 随着实践中公共密钥系统的实际部署，后者变得越来越重要。 某些实体（例如，政府机构，银行，证书颁发机构）的签名权不可避免地会邀请攻击者尝试并“窃取”该权限。阈值签名方案的目标是双重的：增加签名的可用性 代理，同时通过使对手更难学习秘密签名密钥来增强防伪保护，特别是，阈值方法排除了基于传统秘密共享的幼稚解决方案（请参阅第12章） ，其中密钥是成组共享的，但每次要生成签名时都由单个玩家重构，这种协议将与t个（或更少）玩家不能产生新的有效签名的要求相矛盾。 方案中，产生了多个签名，而没有暴露或显式重建密钥。

阈值签名是称为阈值加密的一般方法的一部分。 这种方法在文献中受到了相当多的关注。 我们请读者参考[70]对该领域的工作进行调查。 阈值签名方案的解决方案的特定示例可以在RSA的[69，183]和ElGamal类型的签名的[109]中找到。

如果不仅t个或更少的玩家不能产生有效的签名，而且还不能阻止其余的玩家自己计算签名，则将阈值签名方案称为鲁棒性。 健壮的方案基本上可以防止损坏的服务器部分遭受拒绝服务攻击。 上面提到的解决方案并不可靠。 在本章中，我们将重点介绍健壮的方案。 我们将不讨论技术细节。 本部分的目的是向读者介绍相关概念，并指向文献中的出处。

下面我们将使用字母P1来表示签名服务器。

### 10.6.1门限哎签名方案的密钥商生成

为阈值签名方案生成密钥的任务比在单个签名者在场的情况下要复杂得多。 实际上，我们必须生成一个公共密钥PK，其匹配的秘密密钥SK以某种形式在服务器之间共享

一种方法是让某个受信任的交易商为给定的签名方案生成密钥对（PK; SK），将PK公开并使用秘密共享协议在Pi的用户之间共享SK（请参阅第12章）。但是请注意， 这样的密钥生成机制与任何实体都不能签名的要求相矛盾，因为现在交易商知道秘密密钥SK并且他可以自己签名。 这就是为什么人们一直试图避免在密钥生成阶段使用这种发牌人的原因。

对于基于离散日志的签名方案，此任务已成功完成。 El Gamal，Schnorr和DSS签名方案的稳健阈值签名方案（请参见[90，184，85]）可以在[53，159，94]中找到，全部使用Feldman和Pedersen的基本结果[82，160，161] ]。

但是，在某些情况下，经销商解决方案是我们能做的最好的事情。 例如，如果基础签名方案是RSA，那么我们不知道如何在不使用交易商的情况下以共享形式生成密钥。

### 10.6.2签名协议

一旦密钥生成并以某种方式在服务器之间共享；我们需要一个签名协议。 这个想法是，在输入消息M时，服务器将进行某种形式的通信，这将使它们能够为M计算一个签名，而不会泄露密钥。 除了签名以外，该协议不应泄漏任何信息。 另外，为了获得鲁棒性，即使在协议期间多达t台服务器Pi被破坏并且以任何方式表现，这种协议也应该正确地计算签名。 如果可能的话，服务器Pi以这种分布式方式签名所需的计算应该与Pi自己签名时所需的计算相当。 互动应降到最低。

对于类似El Gamal的方案，可以在[53，159]中找到鲁棒的阈值签名方案。 事实证明，DSS的特殊情况很难处理。 最好的解决方案是在[94]中。

事实证明，RSA甚至不适合构建健壮的方案。 可以在[86]中找到一种效率不高的解决方案（需要更多的计算量以及服务器之间的大量交互）。 在[93]中独立提出了一种非常有效且非交互的解决方案。