组织： PKI 论坛 （http://www.pki.com.cn）

PKCS/PKIX 中文翻译计划

论坛 E-mail：pki@pki.com.cn

译者： Cece

更改记录

日期	修改章节	类型	修改描述	修改人
2004/01/15		C	创建文档	Cece

* 修改类型分为 **C**-CREATE **A** - ADDED **M** - MODIFIED **D** - DELETED

PKCS #1 v2.1 RSA 算法标准

RSA 实验室

2002 年 6 月 14 日

目录

1 介绍

这篇文档是介绍基于 RSA 算法[42]的公钥密码系统的实现方法的，它包括以下几个方面：

- 密码原语
- 加密方案
- 带附属的签名方案
- 密钥和方案的 ASN.1 描述

本文档是为计算机和通信系统的一般应用以及具有一定灵活性的系统中的一般应用所编写的。希望基于这些规范的应用标准可以适用于其它的规范。本篇文档确定要与 IEEE-1363-2000 [26]标准以及 ANSI X9F1 [1] 和 IEEE P1363 [27]工作组当前正在开发的草拟标准兼容。

本文档是 PKCS #1 V2.0[44]的后续版本，但是包含了兼容技术。

下面所列为本篇文档的组织结构：

- 第一部分是介绍。
- 第二部分是对文档中使用到的符号的定义。
- 第三部分详细说明了 RSA 公钥和私钥的类型。
- 第四部分和第五部分详细说明了几个原语，或者说是基本数学操作。第四部分详细说明了数据转换原语，第五部分详细说明了密码系统原语（加密——解密、签名——验证）。
- 本文档中的第六、七和八部分涉及加密和签名的方案。第六部分是概述。连同在 PKCS #1 V1.5 中介绍的方法一起，第七部分定义了基于 OAEP[3]的加密方案和第八部分定义了基于 PSS[4][5]的带附属的签名方案。.
- 第九部分详细说明了在第八部分中定义的签名方案的编码方法。
- 附录 A 详细说明了关于在第三部分中定义的密钥以及第七、八章中的方案的 ASN.1 描述。
- 附录 B 详细说明了本文档用到的散列函数和掩模生成函数，包括这些技术的 ASN.1 描述。
- 附录 C 给出一个 ASN.1 模块。
- 附录 D、E、F 和 G 涉及知识产权问题，概述了 PKCS #1 的修订历史，列出了其它参考出版物和标准，提供了关于公钥密码系统标准的一般信息。

2 符号

c	密文代表，是一个界于 0~n-1 之间的整数。
C	密文，是一个八位组串
d	RSA 私有幂
d_i	其它因子 r_i 的 CRT 幂，是一个满足下式的正整数： $e \cdot d_i \equiv 1 \pmod{(r_i - 1)}, i = 3, \ldots, u$
dP	*p* 的 CRT 幂，是一个满足下式的正整数 $e \cdot dP \equiv 1 \pmod{(p - 1)}$
dQ	*q* 的 CRT 幂，是一个满足下式的正整数 $e \cdot dQ \equiv 1 \pmod{(q - 1)}$
e	RSA 公开幂
EM	编码后的消息，是一个八位组串
emBits	（期望的） 编码消息 *EM* 的以比特为计量单位的长度
emLen	（期望的）编码消息 *EM* 的以八位组为计量单位的长度
GCD (. , .)	两个非负整数的最大公约数
Hash	哈希函数
hLen	散列函数 Hash 的输出的以八位组为计量单位的长度
k	RSA 合数模 *n* 的以八位组为计量单位的长度
K	RSA 私钥
L	可选的 RSAES-OAEP 标签，是一个八位组串
LCM (. , … , .)	一系列非负整数的最小公倍数
m	消息代表，是一个界于 0 到 $n - 1$ 的整数
M	消息，是一个八位组串
mask	MGF 的输出，是一个字节串
maskLen	（期望的）掩模的以八位组为计量单位的长度
MGF	掩模生成函数
mgfSeed	生成掩模的种子因数（seed），是一个八位组串

mLen	消息 M 的以八位组为计量单位的长度
n	RSA 合数模， $n = r_1 \cdot r_2 \cdot \ldots \cdot r_u$, $u \geq 2$
(n, e)	RSA 公钥
p, q	RSA 合数模 n 的前两个素数因子
qInv	CRT 系数，是个满足下式且小于 p 的正整数 $q \cdot qInv \equiv 1 \pmod{p}$
r_i	RSA 合数模 n 的素数因子， 包括 $r_1 = p$, $r_2 = q$, 以及任何另外的因子
s	签名代表，是一个位于 0 到 $n - 1$ 之间的整数
S	签名，是一个八位组串
sLen	EMSA-PSS 盐（salt）的以八位组为计量单位的长度
t_i	其它素数因数 r_i 的 CRT 系数， 是一个比 r_i 小的正整数，满足下式 $r_1 \cdot r_2 \cdot \ldots \cdot r_{i-1} \cdot t_i \equiv 1 \pmod{r_i}$, $i = 3, \ldots, u$
u	RSA 合数模的素数因子的个数, $u \geq 2$
x	一个非负整数
X	与 x 对应的一个八位组串
xLen	（指定的）八位组串 X 的长度
0x	一个八位组或八位组串的十六进制表示法的标志；“0x48” 表示十六进制值为 48 的字节; “(0x)48 09 0e” 表示三个连续的字节，它们的十六进制值分别为 48, 09, 和 0e
$\lambda(n)$	LCM $(r_1 - 1, r_2 - 1, \ldots, r_u - 1)$
$\oplus$	两个八位组串的位异或操作
$\lceil \cdot \rceil$	取整函数；$\lceil x \rceil$ 是一个大于或等于实数 x 的最小整数。
$\|\|$	或操作
$\equiv$	同余符号；$a \equiv b \pmod{n}$ 表示能用整数 n 整除整数 $a - b$

注释：CRT 既可以用于递归方式，也可以用于非递归方式。在本篇文档中使用了在 Garner 算法[22]之后的一种递归方式。请参看 0 节中的注释 1。

3 密钥类型

在本篇文档定义的原语和方案中，使用了两种密钥类型：RSA 公钥和 RSA 私钥。RSA 公钥和 RSA 私钥一起构成一个 RSA 密钥对。

本篇规范支持所谓的“多素数”RSA，这种 RSA 的合数模可能由两个以上的素数因子构成。多素数 RSA 的优点在于，当使用 CRT（中国余数定理），它就能减少解密和签名原语的计算开销。这在单处理器平台上能获得更好的性能，但是在多处理器平台上不一定，在多处理器平台上可以并行处理合数模的幂运算。

至于多素数如何影响 RSA 密码系统的安全性，请读者参考[49]。

3.1 RSA 公钥

为了这篇文档起见，一个 RSA 公钥由两部分构成：

n RSA 合数模，是一个正整数

e RSA 公开幂，是一个正整数

在一个有效的 RSA 公钥中，RSA 合数模 n 是由 u 个不同的奇素数 r_i 生成的，$i = 1, 2, \ldots, u$，其中 $u \geq 2$，而 RSA 公开幂 e 是一个位于 $3 \sim n - 1$ 之间的整数，满足 $\text{GCD}(e, \lambda(n)) = 1$，其中 $\lambda(n) = \text{LCM}(r_1 - 1, \ldots, r_u - 1)$。按照惯例，通常用 p 和 q 分别合数模的前两个素数因子 r_1 和 r_2。

在设备之间互换 RSA 公钥的一个推荐描述在附录 0 中给出；设备中的内部表示可能不同。

3.2 RSA 私钥

为了本篇文档起见，一个 RSA 私钥可以采取两种表示法中的任何一个。

1. 第一种表示法由一对整数(n, d)构成，各部分的意义如下：

n RSA 合数模，是一个正整数

d RSA 私有幂，是一个正整数

2. 第二种表示法由一个五元组$(p, q, dP, dQ, qInv)$和一系列（可能为空）三元组 (r_i, d_i, t_i) $i = 3, \ldots, u$ 构成，三元组的每个素数不出现在五元组中，各部分的意义如下：

p 第一个因子，是一个正整数

q 第二个因子，是一个正整数

dP 第一个因子的 CRT 幂，是一个正整数

dQ 第二个因子的 CRT 幂，是一个正整数

qInv （第一个）CRT 系数，是一个正整数

r_i 第 i 个因子,，是一个正整数

d_i 第 i 个因子的 CRT 幂，是一个正整数

t_i 第 i 个因子的 CRT 系数，是一个正整数

当采用第一种表示法表示时，有效的RSA私钥的RSA合数模 *n*与对应的RSA公钥的RSA合数模*n*一样，是由*u*个不同的奇素数r_i 产生的，$i = 1, 2, \ldots, u,$ 其中$u \geq 2$。RSA私有幂 *d*是一个小于n的正整数，满足

$$e \cdot d \equiv 1 \pmod{\lambda(n)} \quad ,$$

其中 *e* 是对应的 RSA 公开幂，λ(*n*)和第 3.1 中定义的一样。

当用第二种表示法表示时，RSA 私钥的两个因素 *p* 和 *q* 是 RSA 合数模 *n* 前两个素数（也就是 r_1 和 r_2），CRT 幂 dP 和 dQ 是小于 *p* 和 *q* 的正整数，分别满足

$$e \cdot dP \equiv 1 \pmod{(p-1)}$$

$$e \cdot dQ \equiv 1 \pmod{(q-1)} \quad ,$$

CRT 系数 *qInv* 是一个小于 *p* 的正整数，满足

$$q \cdot qInv \equiv 1 \pmod{p} \quad 。$$

如果 $u > 2$，表示法中将包括一个或多个三元组 $(r_i, d_i, t_i), i = 3, \ldots, u$。因子 r_i 是 RSA 合数模 *n* 的一个其它素数因子。每一个 CRT 幂 d_i ($i = 3, \ldots, u$)，满足

$$e \cdot d_i \equiv 1 \pmod{(r_i - 1)} \quad 。$$

每个 CRT 系数 t_i ($i = 3, \ldots, u$)是一个小于 r_i 的正整数，满足

$$R_i \cdot t_i \equiv 1 \pmod{r_i} \quad ,$$

其中 $R_i = r_1 \cdot r_2 \cdot \ldots \cdot r_{i-1}$。

在设备之间互换的 RSA 私钥的推荐描述（包括两种表示法的组成部分）在附录 0 中给出；设备的内部表示可能不同。

注释：

1. 这里 CRT 系数的定义以及第 5 部分的原语中使用到的程式遵循 Garner 算法[22]（也可参见[37]的算法 14.71）。然而，为了与 PKCS #1 v2.0 及之前版本中的 RSA 私钥表示法兼容，*p* 和 *q* 的角色被保留了，这与其它素数不一样。因此，第一个 CRT 系数 *qInv* 被定义成是 q mod p 的倒数（inverse），而不是 $r_1 \bmod r_2$ 的倒数（也就是 $p \bmod q$ 的倒数）。
2. Quisquater 和 Couvreur [40] 注意到在 RSA 运算中使用中国剩余定理的优点。

4 数据转换原语

本篇文档所定义的模式中使用了两个数据转换原语：

- I2OSP – 整数到字节串的转换原语
- OS2IP – 字节串到整数的转换原语

为了本篇文档起见，也为了与 ASN.1 语法一致，一个八位组串是指一个有顺序的八位组（八位比特构成一个字节）序列。整个序列从第一位（通常是最左边的一位）到最后一位（最右边的一位）编入索引。为了转换为整数以及转换整数，在接下来的转换原语中第一个八位组被认为是最重要的。

4.1 I2OSP

I2OSP 把一个非负整数转换为一个长度指定的字节串。

I2OSP (*x*, *xLen*)

输入： *x* 待转换的非负整数

*xLen*转换后的八位组串的期望长度

输出： *X* 对应的长度为 *xLen* 的八位组串

错误信息： “ 整数太大 ”

步骤：

1. 如果 $x \geq 256^{xLen}$，输出“ 整数太大 ”然后终止。
2. 用以 256 为基数的 *xLen* 位数表示整数 *x* ：

   $$x = x_{xLen-1}\, 256^{xLen-1} + x_{xLen-2}\, 256^{xLen-2} + \ldots + x_1\, 256 + x_0 \ ,$$

   其中 $0 \leq x_i < 256$ (注意如果 *x* 小于 256^{xLen-1}，一个或多个高位将为零)。
3. 使字节 X_i 的整数值为 x_{xLen-I} ， $1 \leq I \leq xLen$。输出八位组串

   $$X = X_1\, X_2\, \ldots\, X_{xLen}。$$

4.2 OS2IP

OS2IP 将一个八位组串转换成一个非负整数。

OS2IP (*X*)

输入： *X* 待转换的八位组串

输出： *x* 相应的非负整数

步骤：

1. 使 $X_1 X_2 \ldots X_{xLen}$ 分别为 X 的第一个至最后一个八位组，使 x_{xLen-i} 的值为八位组 X_i 的整数值，$1 \le i \le xLen$。
2. 让 $x = x_{xLen-1} 256^{xLen-1} + x_{xLen-2} 256^{xLen-2} + \ldots + x_1 256 + x_0$。
3. 输出 x。

5 密码原语

密码原语是基本的数学运算，在此基础上形成密码方案。人们打算以硬件或软件模块的形式实现它们，而且不打算提供撇开方案的安全。

在本篇文档中定义了四类原语，以配对的方式组织：加密和解密；签名和验证。

原语规范假定输入满足一定的条件，特别地假设 RSA 公钥和私钥有效。

5.1 加密和解密原语

加密原语在公钥的控制下从消息代表产生出密文代表，解密原语在对应私钥的控制下从密文代表中恢复消息代表。

在本篇文档定义的加密方案中使用了一对加密和解密原语，被描述为：RSAEP/RSADP 。RSAEP 和 RSAEP 涉及相同的数学运算，只是输入的密钥不同。

这里定义的原语和在 IEEE Std 1363-2000 [26]中定义的 IFEP-RSA/IFDP-RSA 一样（除了增加了对多原语 RSA 的支持之外），而且与 PKCS #1 v1.5 兼容。

在每个原语中主要的数学运算是幂运算。

5.1.1 RSAEP

RSAEP ((*n*, *e*), *m*)

输入： (*n*, *e*) RSA 公钥

m 消息代表， 是一个位于 0~ *n* – 1 之间的整数

输出： *c* 密文代表， 是一个位于 0~ *n* – 1 之间的整数

错误提示： “消息代表超出范围”

假设： RSA 公钥(*n*, *e*)有效

步骤：

1. 如果消息代表 m 不在 0 ~n – 1 之间， 输出“ 消息代表超出范围 ”并终止。
2. 让 $c = m^e \bmod n$。
3. 输出 c。

5.1.2 RSADP

RSADP (K, c)

输入： K RSA 私钥，其中 K 采用以下形式中的一种：

— 一对(n, d)

— 一个五元组 (p, q, dP, dQ, $qInv$)和 一系列可能为空的三元组 (r_i, d_i, t_i)， i = 3, …, u

c 密文代表，是一个位于 0 ~n – 1 之间的整数

输出： m 消息代表，是一个位于 0 ~n – 1 之间的整数

出错提示： “ 密文代表超出范围 ”

假设： RSA 私钥 K 有效

步骤：

1. 如果密文代表 c 不在 0 ~n – 1 的范围之内， 则输出“ 密文代表超出范围 ”然后终止。
2. 消息代表 m 按照以下步骤计算。
   a. 如果 K 采用第一种形式(n, d), 使 $m = c^d \bmod n$。
   b. 如果 K 采用第二种形式(p, q, dP, dQ, $qInv$)和 (r_i, d_i, t_i), 则按照以下步骤进行：
      i. 使 $m_1 = c^{dP} \bmod p$ ， $m_2 = c^{dQ} \bmod q$。
      ii. 如果 $u > 2$， 使 $m_i = c^{d_i} \bmod r_i$, $i = 3, \ldots, u$。
      iii. 使 $h = (m_1 – m_2) \cdot qInv \bmod p$。
      iv. 使 $m = m_2 + q \cdot h$。
      v. 如果 $u > 2$，使 $R = r_1$ ， 令 $i = 3$ 然后循环做以下各步骤，直至 $i = u$
         1. 使 $R = R \cdot r_{i–1}$
         2. 使 $h = (m_i – m) \cdot t_i \pmod{r_i}$
         3. 使 $m = m + R \cdot h$
         4. 使 $i=i+1$

3. 输出 m。

注释：如果保留了 p 和 q 的定义，步骤 2.a 可以写成单个循环。然而，为了与 PKCS #1 v2.0 兼容，前两个素数 p 和 q 与其它素数分开处理。

5.2 签名和验证原语

签名原语在私钥的控制下从消息代表产生一个签名代表，而验证原语是在对应公钥的控制下从签名代表恢复出消息代表。本篇文档定义的签名方案中使用了一对签名和验证原语，被描述为：RSASP1/RSAVP1。

这里定义的原语与 IEEE 1363-2000 [26]中定义的 IFSP-RSA1/IFVP-RSA1 是一样的（所不同的是这里的原语增加了对多原语 RSA 的支持），而且与 PKCS #1 v1.5 兼容。

在每个原语中的主要数学操作是幂操作，这一点和 0 部分中的加密和解密原语一样。RSASP1 和 RSAVP1 与 RSADP 和 RSAEP 除了输入和输出参数的名称不一样之外，其它各方面都一样；它们的区别在于它们是为不同的目的而编写的。

5.2.1 RSASP1

RSASP1 (K, m)

输入： K RSA 私钥, 这里 K 具有以下形式之一：

— 一对(n, d)

— 一个五元组(p, q, dP, dQ, $qInv$)和一系列可能为空的三元组(r_i, d_i, t_i)， $i = 3, \ldots, u$

m 消息代表，是一个位于 0~ $n - 1$ 之间的整数

输出： s 签名代表，是一个位于 0~ $n - 1$ 之间的整数

出错提示： “消息代表超出范围”

假设： RSA 私钥 K 有效的

步骤：

1. 如果消息代表 m 不在 0 ~$n - 1$ 之间，输出“消息代表超出范围” 然后终止运算。
2. 签名代表 s 由以下步骤计算得出。
   a. 如果 K 采用第一种形式(n, d)， 使 $s = m^d \bmod n$。
   b. 如果 K 采用第二种形式(p, q, dP, dQ, $qInv$)和 (r_i, d_i, t_i)，则按照以下步骤进行：
      i. 使 $s_1 = m^{dP} \bmod p$ ， $s_2 = m^{dQ} \bmod q$。

ii. 如果 $u > 2$，让 $s_i = m^{d_i} \bmod r_i$, $i = 3, \ldots, u$。

iii. *让* $h = (s_1 - s_2) \cdot qInv \bmod p$.

iv. *让* $s = s_2 + q \cdot h$.

v. *如果* $u > 2$, 让 $R = r_1$ ，$i = 3$ ，然后循环进行以下各步骤直至 $i = u$

1. *让* $R = R \cdot r_{i-1}$.
2. *让* $h = (s_i - s) \cdot t_i \pmod{r_i}$.
3. *让* $s = s + R \cdot h$
4. 让 $i = i+1$

3. 输出 s。

注释：如果保留了 p 和 q 的定义，步骤 2.a 还可以写成单个循环。然而，为了与 PKCS #1 v2.0 兼容，前两个素数 p 和 q 与其它素数分开处理。

5.2.1 RSAVP1

RSAVP1 ((*n*, *e*), *s*)

输入： (*n*, *e*) RSA 公钥

s 签名代表， 是一个位于 0~ $n - 1$ 之间的整数

输出： *m* 消息代表， 是一个位于 0~ $n - 1$ 之间的整数

出错提示： “签名代表超出范围”

假设: RSA 公钥(*n*, *e*) 有效

步骤：

1. 如果签名代表 *s* 不在范围 0 ~$n - 1$ 之间,，输出 “ 签名代表超出范围 ” 然后终止运算。
2. 让 $m = s^e \bmod n$。
3. 输出 *m*。

6 方案概述

方案结合了密码原语和其它技术以获得特定的安全目标。在本篇文档种定义了两类方案：加密方案和带附属的签名方案。

本篇文档种定义的方案适用于有限的范围，它们的操作只是由几个使用 RSA 公钥或私钥处理数据的步骤构成，不包括获得密钥或者证实密钥的步骤。因此，一个典型的应用除了包括方案中的操作之外，还应包括密钥管理操作（通过该操作双方可以为一次方案操作选择 RSA 公钥和私钥）。这些具体的额外操作以及其它细节超出了本篇文档的讨论范围。

根据密码原语（见第 0 部分），方案操作规范假定输入满足一定的条件，特别是满足 RSA 公钥和私钥均有效。因此，如果密钥无效，则不规定设备行为。这种不规定行为的影响依赖于实际应用。声明密钥有效的包括通过申请明确密钥有效；在公钥基础架构内密钥的有效性；使密钥生成方承担使用有效密钥进行操作的责任。

一个大体上好的密码实现是将一对给定的 RSA 密钥对只用于一个方案。这样就避免使一个方案的漏洞连累其它方案的安全性，这一点对维持可证实的安全性可能是至关重要的。尽管 RSAES-PKCS1-v1_5（见 0 部分)和 RSASSA-PKCS1-v1_5 (见 0 部分)一贯被结合在一起使用，且相互之间没有任何已知的不好的影响（实际上，这就是 PKCS #1 v1.5 介绍的方案），在新应用中不推崇这种结合使用一个 RSA 密钥对的做法。

为了说明在多个方案中使用一个 RSA 密钥对的风险，我们假定 RSAES-OAEP (见 0 部分) 和 RSAES-PKCS1-v1_5 中使用同一个 RSA 密钥对。尽管 RSAES-OAEP 自身具备抗攻击性，但对手可以利用 RSAES-PKCS1-v1_5 实现中的一个漏洞恢复出用两个方案中任何一个方案加密的消息。再举另外一个例子，假设 RSASSA-PSS (见 0 部分)和 RSASSA-PKCS1-v1_5 中使用同一个 RSA 密钥对。如果对 RSASSA-PSS 的安全坚固性没有考虑到签名可以由另一个方案生成的可能性，那么这种坚固性将不再充分。如果一个 RSA 密钥对被用于这里定义的方案中的任何一个，而且也被用于其它地方定义的方案，那么可能需要作出同样的考虑。

7 加密方案

就本篇文档而言，一个加密方案由一个加密操作和一个解密操作构成，其中加密操作使用接受方的 RSA 公钥把消息转化成密文，而解密操作使用接受方对应的 RSA 私钥将密文恢复成消息。

一个加密方案能用于各种应用中。一个典型的应用是密钥建立协议，在该协议的消息中包含将被秘密地从一方递送到另一方的密钥。举例来说，PKCS #7 [45]使用这种协议将一个内容的加密密钥从发送方递送到接收方；这里定义的加密方案将会是适合于上述情况的密钥加密算法。

在本篇文档中定义了两个加密方案：RSAES-OAEP 和 RSAES-PKCS1-v1_5。 RSAES-OAEP 是新应用的推荐标准；加入 RSAES-PKCS1-v1_5 只是为了与已存在的应用兼容，并且不建议用于新应用。

这里给出的加密方案遵循一般的模型，这个模型类似于在 IEEE Std 1363-2000 [26]中使用到的模型，就是将加密和解密原语与针对加密的编码方法结合起来。这种加密运算对消息进行消息编码运算，以产生一个编码消息，然后将编码后的消息转换成一个整数消息代表。一个加密原语作用于这个消息代表从而产生密文。反之，解密运算将解密原语作用于密文，从而恢复出消息代表，然后将这个消息代表转换成一个以八位组串为形式的编码消息。一个消息解码操作作用于这个编码消息，从而恢复出消息并验证解密运算的正确性。

为了避免与这种方式相关的实现漏洞，在解码运算中会处理错误（参见[6]和[36]），RSAES-OAEP 和 RSAES-PKCS1-v1_5 的编码和解码运算嵌入在各自的加密方案规范中，而不是定义在单独的规范中。两个加密方案均与 PKCS #1 v2.0 中的相应方案兼容。

7.1 RSAES-OAEP

RSAES-OAEP 将 RSAEP 和 RSADP 原语（参见 0 和 0 部分）与 EME-OAEP 编码方法结合起来（参见 0 中的步骤 1.b 和**错误！未找到引用源。**中的步骤 3）。EME-OAEP 基于 Bellare 和 Rogaway 的最佳非对称加密 scheme[3]。（OAEP 代表 “ 最佳非对称加密填充 ”）。它与在 IEEE Std 1363-2000 [26]中定义的 IFES 方案兼容，在 IFES 中加密和解密原语是 IFEP-RSA 和 IFDP-RSA，消息编码方法是 EME-OAEP。RSAES-OAEP 能够操作长度超过 $k - 2hLen - 2$ 字节的消息，这里 *hLen* 是基础散列函数输出的长度，而 *k* 是接收方 RSA 合数模的以八位组为计量单位的长度。

假设计算 e^{th} roots modulo *n* 是不可行的，且 RSAES-OAEP 中的掩模生成函数具有固有的属性；那么 RSAES-OAEP 语义上能抵抗适合的选择密文攻击。如果掩模生成函数被看作是一个黑盒子或者是一个随机的启示程序，那么在攻破 RSAES-OAEP 的难度能直接与使 RSA 函数反向的难度相关的意义上，这个保证是可证实的；进一步的讨论参见[21]和下面的注释。

RSAES-OAEP 加密和解密运算都将标签 *L* 作为输入。在 PKCS #1 的这个版本中，*L* 是一个空串，这个标签的其它使用超出了本篇文档的范围。相关的 ASN.1 语法描述参见附录 A.2.1。

通过散列函数和掩模生成函数的选择以确定 RSAES-OAEP 的参数。这一选择过程对给定的 RSA 密钥是固定不变的。在附录 B 中给出了建议的散列函数和掩模生成函数。

注释：近来的结果对澄清 OAEP 编码方式[3]的安全性[3]有帮助（0 部分中的步骤 1.b 对这个过程作了粗略的描述）。其背景如下所述。1994 年，Bellare 和 Rogaway[3]引入了一个安全概念，他们表述为明文意识（PA94）。他们证明如果一个公钥

加密原语（例如 RSAEP）没有私钥就难以倒转，那么对应的基于 OAEP 的加密方案就是有明文意识的(in the random oracle model)，大致的意思是对手在没有真正知道基础明文情况下是不能产生出合法的密文的。一个加密方案的明文意识与反密文选择攻击的方案的限制有密切联系。在这种攻击中，对手有机会发送询问给一个模仿解密原语的启示程序。使用这些询问的结果，对手尝试解密一个密文。

然而，存在两类密文选择攻击，而 PA94 只包含了针对其中一种攻击的安全性。不同之处在于对手在获得 challenge 密文之后，可以做些什么事情。在对手获得 challenge 密文后，*indifferent* 攻击方案（CCA1 所表示的）不允许向解密启示程序提出任何询问，而 *adaptive* 方案（CCA2 中有表示）允许（除了解密启示程序在其被发布之后拒绝解密 challenge 密文的情况）。1998 年 Bellare 和 Rogaway 与 Desai 和 Pointcheval[2]一道提出了一个新的更健壮的明文意识观念(PA98)，它提出了针对 CCA2 的安全性。

总的说来，存在两个错误观念的潜在来源：PA94 和 PA98 是相同的概念; 或者 CCA1 和 CCA2 是相同的概念。任何一个假设都会得出一个结论，那就是 Bellare-Rogaway 的论文提出针对 CCA2 的 OAEP 安全性，而实际上它并没有提出针对 CCA2 的 OAEP 安全性。[1] OAEP 从未被证明对 CCA2 来说是安全的；事实上，Victor Shoup[48]已经证实这种验证并不存在于一般的情况中。简单地说，Shoup 指出 CCA2 方案中的对手知道如何使加密原语部分反向，但不知道如何使它完全反向，这个对手也能攻破这个方案。举例来说，一个人可以想象如果攻击者知道如何恢复用 RSAEP 加密的一个随机整数的所有子节而不是前 20 个字节，那么她就能够攻破 RSAES-OAEP。这样的攻击者不需要将 RSAEP 完全反向，因为在她的攻击过程中不需要使用前 20 个八位组。

同样地，RSAES-OAEP 对于 CCA2 来说是安全的，就在 Shoup 宣布他的结果之后不久，这一点就被 Fujisaki、 Okamoto、Pointcheval 和 Stern [21]证明了。假如能知道 pre-image 的足够大的一部分，使用聪明的网点缩小技术（clever lattice reduction techniques），他们能设法说明如何使 RSAEP 完全反向。这个观察结合了一个验证，那就是如果基础的加密原语难以被部分反向，那么 OAEP 对 CCA2 来说是安全的；这个观察填补了 Bellare 和 Rogaway 所证明的关于 RSAES-OAEP 的事实和某人认为他们证实的事实之间的缺口。我们就这样被 RSAEP 的公开弱点挽救了（也就是说，全部反向可以从部分反向中演绎出来），这有些自相矛盾。

然而不幸的是，降低安全对于具体的参数并不有效。尽管这个验证成功地将 RSAES-OAEP 的 CCA2 安全性的对手 A 与反向 RSA 的算法 I 关联起来，I 成功的概率仅近似于$\varepsilon^2 / 2^{18}$，其中ε是A成功的概率。另外，I 的运行时间大约是 t^2，其中 t 是对手的操作时间。结果是我们不能排除攻击 RSAES-OAEP 比用具体参数反向 RSA 容易得多的可能性。安全验证的存在仍然提供了一些保证，那就是 RSAES-OAEP 结构比 *ad hoc* 结构（诸如 RSAES-PKCS1-v1_5）合理。

Hybrid 加密方案基于 RSA-KEM 密钥封装范例，并提供严密的安全性验证，能直接应用于具体参数；详情见[30]。PKCS #1 的后续版本可以定义基于这个范例的方案。

[1] PKCS #1 v2.0 引用了[3]并且声称“在没有明确明文意识或选择密文攻击类型的情况下，选择密文攻击对于有明文意识的加密方案，像 RSAES-OAEP，来说是无效的”。

[2] 在 [21]中，反向成功的概率是$\varepsilon^2 / 4$。其它因子 $1 / 2^{16}$ 取决于编码消息 *EM* 的开头八个固定的零比特，这在[21]中考虑的 OAEP 变化中没有出现。（我必须两次应用 A 来反向 RSA，且每个应用程序与因子 $1 / 2^8$ 相应）

7.1.1 加密运算

RSAES-OAEP-ENCRYPT ((*n*, *e*), *M*, *L*)

可选项： Hash 哈希函数（*hLen* 表示散列函数输出的以八位组为计量单位的长度）

MGF 掩模生成函数

输入： (*n*, *e*) 接收方的 RSA 公钥（*k* 表示 RSA 合数模 *n* 的以八位组为计量单位的长度）

M 待加密的消息，是一个长度为 *mLen* 的八位组串，其中 $mLen \le k - 2hLen - 2$

L 消息的可选附加标签；如果没有提供 *L*，那么 *L* 的默认值是空串

输出: *C* 密文，一个长度为 *k* 的八位组串

出错提示： “消息太长”；“标签太长”

假设： RSA 公钥(*n*, *e*) 是有效的

步骤：

1. 长度检查：
   a. 如果 *L* 的长度超出哈希函数的输入限制 （SHA-1 的限制是 $2^{61} - 1$ 个八位组）， 输出“标签太长”然后终止运算。
   b. 如果 $mLen > k - 2hLen - 2$，输出“消息太长”然后终止运算。
2. *EME-OAEP* 编码（见 **错误！未找到引用源。**）：
   a. 如果没有提供标签 *L*， 则让 *L* 为空串。 让 *lHash* = Hash (*L*)， 这是一个长度为 *hLen* 的八位组串（见下面的注释）。
   b. 生成一个由 $k - mLen - 2hLen - 2$ 个零值八元组构成的串 *PS*。 *PS* 的长度可能是零。
   c. 连接 *lHash*，*PS*，十六进制值为 0x01 的八元组和消息 *M* ，形成一个长度为 $k - hLen - 1$ 个八位组的数据块 *DB* ：

$$DB = lHash \,||\, PS \,||\, 0x01 \,||\, M \text{ 。}$$

   d. 生成一个长度为 *hLen* 的随机八位组串 *seed* 。
   e. 使 *dbMask* = MGF (*seed*, $k - hLen - 1$)
   f. 使 $maskedDB = DB \oplus dbMask$.

g. 使 *seedMask* = MGF (*maskedDB*, *hLen*).

h. 使 *maskedSeed* = *seed* ⊕ *seedMask*.

i. 连接一个十六进制值为 0x00 的八位组， *maskedSeed* 和 *maskedDB* ，形成一个长度为 *k* 个八位组的编码消息 *EM*

$$EM = 0x00 \| maskedSeed \| maskedDB。$$

3. *RSA* 加密：

a. 将编码消息 *EM* 转换成一个整数消息代表（见 4.2 部分）：

$$m = OS2IP (EM) 。$$

b. 将 RSA 公钥(*n*, *e*) 和消息代表 *m* 代入 RSAEP 加密原语（5.1.1 部分），产生一个整数的密文代表 *c* ：

$$c = RSAEP ((n, e), m) 。$$

c. 将密文代表 *c* 转换为一个长度为 *k* 个八元组的密文 *C* （见 4.1 部分）：

$$C = I2OSP (c, k) 。$$

4. 输出密文 *C*。

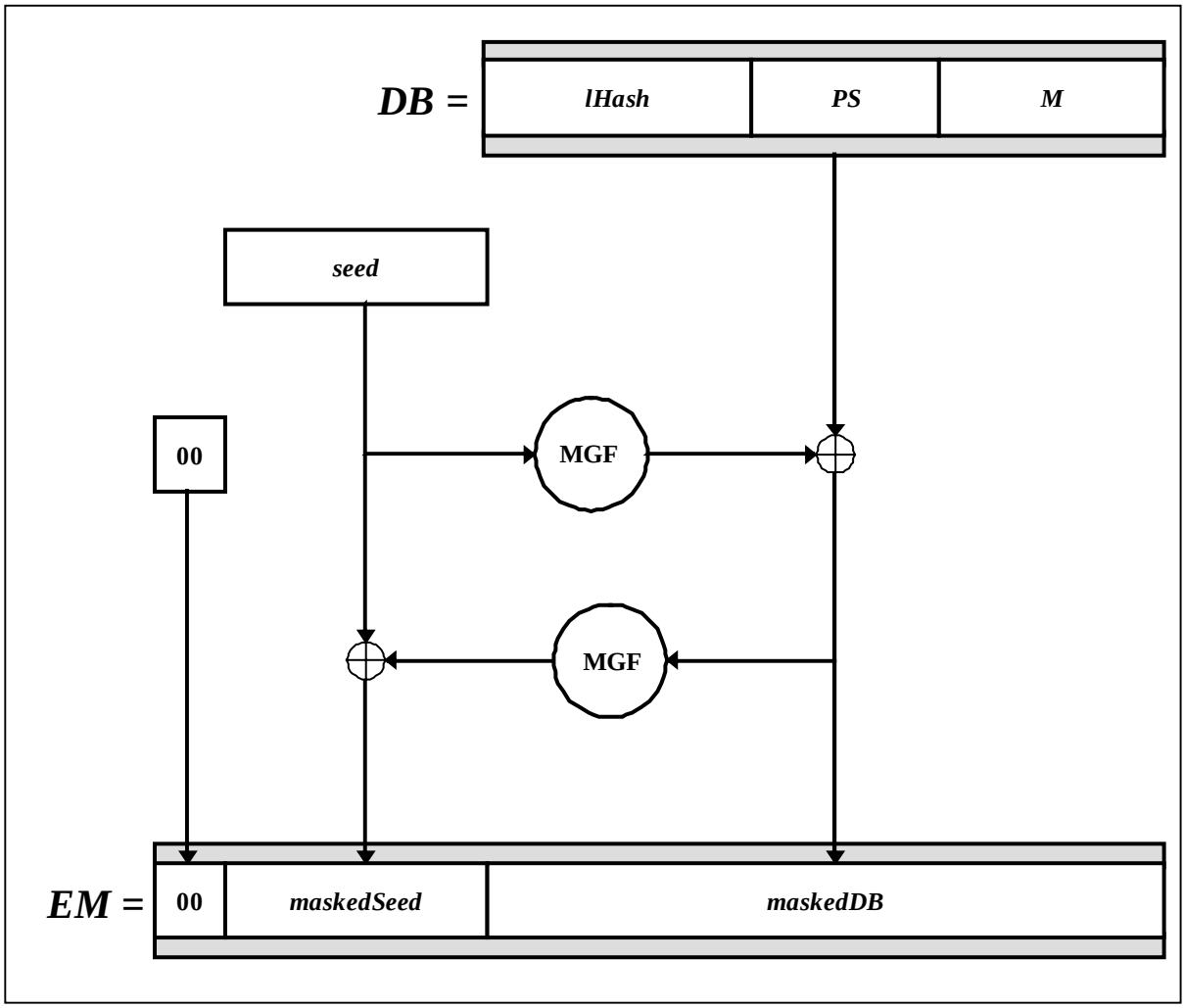


图 1：EME-OAEP 编码运算。*lHash* 是可选标签 *L* 的散列。解码运算按照相反的步骤进行，从而恢复出 *M* 并且验证 *lHash* 和 *PS*。

注释：如果 L 是空串，相应的散列值 lHash 具有下列十六进制值代表哈希函数的不同选择。

SHA-1： **(0x)da39a3ee 5e6b4b0d 3255bfef 95601890 afd80709**

SHA-256： **(0x)e3b0c442 98fc1c14 9afbf4c8 996fb924 27ae41e4 649b934c a495991b 7852b855**

SHA-384： **(0x)38b060a7 51ac9638 4cd9327e b1b1e36a 21fdb711 14be0743 4c0cc7bf 63f6e1da**
274edebf e76f65fb d51ad2f1 4898b95b

SHA-512： **(0x)cf83e135 7eefb8bd f1542850 d66d8007 d620e405 0b5715dc 83f4a921 d36ce9ce**
47d0d13c 5d85f2b0 ff8318d2 877eec2f 63b931bd 47417a81 a538327a f927da3e

7.1.2 解密运算

RSAES-OAEP-DECRYPT (*K*, *C*, *L*)

选项： Hash 散列函数哈希（*hLen* 表示散列函数的输出的以八位组为计量单位的长度）

MGF 掩模生成函数

输入： *K* 接受方的 RSA 私钥（*k* 表示 RSA 合数模 *n* 的以八位组为计量单位的长度）

C 待解密的密文，使一个长度为 *k* 的八位组串，其中 $k \geq 2hLen + 2$

L 可选标签，其与消息的联系将得到验证；如果没有提供 L 值，则 L 的默认值为空串。

输出： *M* 消息，是一个长度为 *mLen* 的八位组串，其中 $mLen \leq k - 2hLen - 2$

错误提示： “解密出错”

步骤：

1. 长度检查：
   a. 如果 *L* 的长度大于散列函数的输入限制（SHA-1 的限制是 $2^{61} - 1$ 个八位组），输出“解密出错”并中止运算。
   b. 如果密文 *C* 的长度不是 *k* 个八位组，则输出“解密出错”并中止运算。
   c. 如果 $k < 2hLen + 2$，则输出“解密出错”并中止运算。
2. *RSA* 解密：
   a. 将密文 *C* 转换成一个整数密文代表 *c*（见 0 部分）：

      c = OS2IP (C) 。

   b. 将 RSA 私钥 *K* 和密文代表 *c* 代入 RSADP 解密原语（见 0 部分），从而产生一个整数消息代表 *m*：

      m = RSADP (K, c) 。

      如果 RSADP 输出“密文代表超出范围”（意思是 $c \geq n$），则输出“解密出错”并且中止运算。
   c. 将消息代表 *m* 转换成一个长度为 *k* 个八位组的编码消息 EM（见 0 部分）：

      EM = I2OSP (m, k) 。

3. *EME-OAEP* 编码：
   a. 如果未提供标签 *L* 的值，则使 *L* 的值为空串。使 *lHash* = Hash (*L*)，这是一个长度为 *hLen* 的八位组串（见 0 部分的注释）。

b. 将编码消息 *EM* 分解为一个八位组 *Y*，一个长度为 *hLen* 的八位组串 *maskedSeed*，以及一个长度为 *k – hLen* – 1 的八位组串 *maskedDB*，使得

$$EM = Y \| maskedSeed \| maskedDB$$ 。

c. *使 seedMask* = MGF (*maskedDB*, *hLen*).

d. *使 seed* = *maskedSeed* ⊕ *seedMask*.

e. *使 dbMask* = MGF (*seed*, *k – hLen* – 1).

f. *使 DB* = *maskedDB* ⊕ *dbMask*.

g. 将 *DB* 分解成一个长度为 *hLen* 的八位组串 *lHash'*，一个（可能为空的）由十六进制值为 0x00 的八位组构成的填充 *PS*，以及一个消息 *M*，使得

$$DB = lHash' \| PS \| 0x01 \| M .$$

如果没有可以从 *M* 中分离出 *PS* 的十六进制值为 0x01 的八位组，如果 *lHash* 没有等同的 *lHash'*，或者如果 *Y* 是非零的，则输出 " 解密出错 " 并中止运算。（见下面的注释）

4. 输出消息 *M*。

注释：必须确保对手无法在步骤 3.f 中分辨出不同的出错条件，防止对手了解关于编码消息 EM 的部分信息，无论是通过出错消息或是定时，或者更一般的。否则对手可能能够获得关于密文 C 的解密的有用信息，进而导致像 Manger 发现的攻击手法一样的选择密文攻击[36]。

7.2 RSAES-PKCS1-v1_5

RSAES-PKCS1-v1_5 将 RSAEP 和 RSADP 原语（见 0 部分和 0 部分）与 EME-PKCS1-v1_5 编码方法（见 0 部分中的步骤 1 和 0 部分中的步骤 3）结合起来。它在数学上等同于 PKCS #1 v1.5 中的加密方案。尽管当加密长消息（参见下面注释的第三条以及[10]，[14]中有一个改良的攻击）时，应该避免起因于 Coppersmith、Franklin、 Patarin 和 Reiter 的低幂 RSA 的攻击；但 RSAES-PKCS1-v1_5 能够对长度大于 *k – 11*（k 是 RSA 合数模的以八位组为计量单位的长度）个八位组的消息进行运算。总的来看，由于这个方案反对随机生成密钥，因此不推荐使用这个方案加密一个任意的消息。考虑到一个合理的成功可能性，即使不知道对应的明文也可能生成有效的 RSAES-PKCS1-v1_5 密文。这种可能性可以用于[6]中所描述的选择密文攻击。因此，如果要使用 RSAES-PKCS1-v1_5，应采取某些容易实现的应对措施以阻挠[6]中描述的攻击。典型的例子包括增加待编码数据的结构，严密地检查解密后消息的 PKCS #1 v1.5 的一致性（以及其它冗余性），统一基于 PKCS #1 v1.5 的一个客户机\服务器协议中的错误消息。这些都可以作为有效的应对

措施，不需要修改一个基于 PKCS #1 v1.5 的协议。如需获得关于上述措施和其它应对措施的进一步讨论，请参见 See [7] 。近来研究表明 SSL/TLS 握手协议（使用 RSAES-PKCS1-v1_5 和某些应对措施）[17]的安全性与不同的 RSA 问题有关；进一步讨论请参见[32] 。

注释：一下段落描述一些使用 RSAES-PKCS1-v1_5 所固有的安全建议。除了这几年间密码改进中提出的新建议之外，也包括了本篇文档的版本 1.5 中的建议。

- 建议为每个加密过程独立生成 0 部分中的步骤 2 中用到的伪随机八位组，特别是在同样的数据被输入多个加密过程时。Håstad 的结果[24]是提出这个建议的一个动机。
- 0 部分步骤 2 中的填充串 PS 的长度至少为八个八位组，这是公钥运算的安全条件，它使得攻击者难以通过尝试所有可能的加密块来恢复数据。
- 当待加密的消息的大小较小，伪随机八位组也能够帮助阻碍起因于 Coppersmith et al[10]的攻击（关于这个攻击的改进，参见[14]）。当相似的消息用同样的 RSA 公钥加密，这个攻击就会作用于低幂 RSA。
- 当待加密的消息的长度保持较小，则伪随机八位组能帮助一次起因于 Coppersmith et al. [10]的攻击（[14]中有攻击的改进）。当使用同样的 RSA 公钥加密类似的消息，这个攻击对低幂 RSA 就奏效了。特别地，这个攻击的一个特色是，当 RSAEP 的两个输入构成比特的一个大小数（8/9），且使用低幂 RSA（$e = 3$）加密它们，用这个攻击恢复这两个输入是可能的。这个攻击的另一个特色是当已知 RSAEP 输入的一个大分数（2/3），就能成功解密单个密文。对于典型 1 应用程序，待加密的消息很短（例如，一个 128 位的对称密钥），所以既不是需要知道足够的信息，也不是两个消息的相同点，使这次攻击得手。然而，如果加密长消息，或者如果消息的某部分已知，那么这个攻击可能是一个小玩意儿。不管怎样，RSAES-OAEP 方案克服了这个攻击。

7.2.1 加密运算

RSAES-PKCS1-V1_5-ENCRYPT $((n, e), M)$

输入： (n, e) 接收方的 RSA 公钥（k 表示合数模 n 的以八位组为计量单位的长度）

M 待加密的消息，是一个长度为 $mLen$ 的八位组串，其中 $mLen \le k - 11$

输出： C 密文，是一个长度为 k 的八位组串

出错提示： “消息太长”

步骤：

1. 长度检查： 如果 $mLen > k - 11$，输出“消息太长”然后中止运算。
2. EME-PKCS1-v1_5 编码：
   a. 生成一个长度为 $k - mLen - 3$ 且由伪随机生成的非零八位组构成的八位组串 PS 。PS 的长度至少为八个八位组。
   b. 连接 PS、消息 M 和其它填充，从而形成一个长度为 k 个八位组的编码消息 EM

$$EM = 0x00 \| 0x02 \| PS \| 0x00 \| M \text{ 。}$$

3. RSA 加密：

   a. 将编码消息 *EM* 转换成一个整数消息代表 *m*（参见 4.2 部分）：

   $$m = \text{OS2IP } (EM) \text{ 。}$$

   b. 将 RSA 公钥(n, e) 和消息代表 *m* 代入 RSAEP 加密原语（参见 5.1.1 部分），从而产生一个密文代表 *c*：

   $$c = \text{RSAEP } ((n, e), m) \text{ 。}$$

   c. 将密文代表 *c* 转换成一个长度为 *k* 个八位组的密文代表 c（参见 4.1 部分）：

   $$C = \text{I2OSP } (c, k) \text{ 。}$$

4. 输出密文 *C*。

7.2.2 解密运算

RSAES-PKCS1-V1_5-DECRYPT (*K*, *C*)

输入： *K* 接收方的 RSA 私钥

C 待解密的密文，是一个长度为 k 的八位组串，其中 k 是 RSA 合数模 n 的以八位组为计量单位的长度。

输出： *M* 消息，是一个长度至少为 $k - 11$ 的八位组串

出错提示： “解密出错”

步骤：

1. 长度检查：如果密文 C 的长度不是 k 个八位组（或者如果 $k < 11$），则输出“解密出错”并且中止运算。

2. RSA 解密：

   a. 将密文 *C* 转换成一个整数密文代表 *c*（参见 4.2 部分）：

   $$c = \text{OS2IP } (C) \text{ 。}$$

b. 将 RSA 私钥(n, d) 和密文代表 c 代入 RSADP 解密原语（参见 5.1.2 部分），进而产生一个整数消息代表 m：

$$m = \text{RSADP} \quad ((n, d), c) \text{ 。}$$

如果 RSADP 输出“密文代表超出范围”（意思是 $c \geq n$），则输出“解密出错”然后中止运算。

c. 将消息代表 m 转换为一个长度为 k 个八位组串的编码消息 EM ：

$$EM = \text{I2OSP } (m, k) \text{ 。}$$

3. EME-PKCS1-v1_5 解码：将编码消息 EM 分离成一个八位组串 PS （由非零八位组构成）和一个消息 M ，使满足

$$EM = 0\text{x}00 \parallel 0\text{x}02 \parallel PS \parallel 0\text{x}00 \parallel M \text{ 。}$$

如果 EM 的第一个八位组的十六进制值不为 0x00，如果 EM 的第二个八位组的十六进制值不为 0x02，如果没有十六进制值为 0x00 的八位组可以从 M 分离出 PS，或者如果 PS 的长度小于 8 个八位组，输出“解密出错”并且中止运算。

4. 输出 M。

注释：必须确保对手无法在步骤 3 中分辨出不同的出错条件，无论是通过出错消息或是定时。否则对手可能能够获得关于解密密文 C 的有用信息，进而导致 Bleichenbacher 攻击[6]的增强版；比得上（compare to） Manger 的攻击 [36]。

8 带附属的签名方案

就本篇文档而言，一个带附属的签名方案包括一个签名生成运算和一个签名验证运算，其中签名生成运算利用签名者的 RSA 私钥产生一个签名，而签名验证运算利用签名者对应的 RSA 公钥验证消息上的签名。为了验证用这种方案产生的签名，验证者必须拥有消息本身。所以说，带附属的签名方案与带消息恢复的签名方案不同，本篇文档不讨论带消息恢复的签名方案。

一个带附属的签名方案可以用于各种应用程序中。举例来说，这里定义的带附属的签名方案是适合于

X.509 认证[28]的签名算法。尽管由于技术原因，PKCS #7 当前版本将散列函数从签名方案（这个方案与这里提到的不一样）中分离出来，但相关的签名方案可以用在 PKCS #7 中；更多的讨论，请参见附录 A.2.3 的注释。

本篇文档种定义了两种带附属的签名方案： RSASSA-PSS 和 RSASSA-PKCS1-v1_5。尽管还不知道有什么攻击是针对 RSASSA-PKCS1-v1_5 的，但是为了提高健壮性，在新应用程序中推荐采纳 RSASSA-PSS。 包含 RSASSA-PKCS1-v1_5 是为了与已存在的应用程序兼容，而且尽管 RSASSA-PKCS1-v1_5 仍然适用于新的应用程序，但是鼓励逐步向 RSASSA-PSS 转换。

这里给定的带附属的签名方案遵循一般的模型（与 IEEE Std 1363-2000 [26]中使用的一样），即将签名和验证原语与针对签名的编码方法结合起来。签名生成运算对消息进行消息编码运算以产生一个编码消息，随后该编码消息将被转换成一个整数消息代表。签名原语作用于消息代表，从而产生一个签名。与此相反，签名验证运算将签名验证原语应用于签名，以恢复出消息代表，然后消息代表被转换成一个编码后的八位组串消息。验证运算作用于这个消息和编码后的消息，以判断它们俩是否一致。

如果编码方法是确定性的（例如，EMSA-PKCS1-v1_5），验证运算可以对消息进行消息编码运算，并将运算结果——编码消息与先前获得的编码消息对比。如果匹配，则签名被认为是有效的。如果编码方法是随机的（如 EMSA-PSS），那么验证运算会更加复杂。举例来说，EMSA-PSS 验证操作从编码后的消息中提取随即 salt 和一个散列输出，并检查该散列输出、salt 和消息是否一致；从消息和 salt 方面来说，散列输出是一个确定的函数。

对于在本篇文档中定义的带附属的签名方案来说，如果签名放在消息的后面，签名生成运算和签名验证运算就像‘ 单方传送 ’运算一样容易实现。至于在 RSASSA-PKCS1-v1_5 情况下的示例格式，请参见 PKCS #7 [45]。

8.1 RSASSA-PSS

RSASSA-PSS 将 RSASP1 和 RSAVP1 原语与 EMSA-PSS 编码方法结合起来。它与在 IEEE P1363a 草案中修订过的 IFSSA 方案兼容 IFSSA 方案，IFSSA 方案中的签名和验证原语是 IEEE Std 1363-2000 [26]中定义的 IFSP-RSA1 和 IFVP-RSA1，而消息编码方法是 EMSA4。由于 EMSA4 是作用于比特串而不是八位组串的，所以它比 EMSA-PSS 更通用。当限制于运算体和散列、salt 均为八位组串的情况下，EMSA-PSS 等同于 EMSA4。

RSASSA-PSS 能够运算的消息的长度可以是不受限制的也可以受一个相当大的数的约束，这取决于 EMSA-PSS 编码方法基于的散列函数。

假设计算 e^{th} roots modulo n 是可行的，EMSA-PSS 中的散列函数和掩模生成函数具有适当的属性，RSASSA-PSS 提供安全签名。如果散列函数和掩模生成函数被看作是黑盒子或者是随机启示程序，在伪造签名的难度可以直接与使 RSA 函数反向的难度关联的意义上，这个保证是可证实的。安全验证的边界本质上是“紧的”，意思是对于最好的伪造者来说，成功的可能性和运行时间非常接近于最好的 RSA 反向算法的对应参数；进一步的讨论，请参见[4][13][31]。

与 RSASSA-PKCS1-v1_5 签名方案对比，EMSA-PSS 编码消息中不嵌入散列函数标识，所以在理论上，对手可能用一个不同的散列函数来替换由签名者选择的散列函数。因此，建议将 EMSA-PSS 掩模生成函数基于同一个散列函数。以这种方式，整个编码后的消息将依赖于散列函数，而且对手将难以用一个不同的散列函数来替换由签名者选择的散列函数。匹配散列函数只是为了防止散列函数被替换，而且如果采用其他方法（例如，验证者只接受指定的散列函数）防止散列函数被替换，则不需要匹配散列函数。关于这几点的进一步讨论，请参见[34]。RSASSA-PSS 的可证实的安全性不依赖于掩模生成函数中的散列函数（与应用于消息的散列函数一样）。

由于结合了随即生成 salt 的值，因此 RSASSA-PSS 与其它基于 RSA 的签名方案的不同之处在于它是概率性的而非确定性的。通过提供一个比确定性的可供选择方案（诸如全域散列法（FDH））更加“严密的”安全验证，Salt 值增强了这个方案的安全性；参见[4]中的讨论。然而，对安全性来说随机并非关键所在。由于最后的可证实安全性与 FDH[12]的相似，在随机生成不可能实现的情况下，一个固定值或者一串数字可以取代之。

8.1.1 签名生成运算

RSASSA-PSS-SIGN (*K*, *M*)

输入： *K* 签名者的 RSA 私钥

M 待签名的消息，是一个八位组串

输出： *S* 签名，是一个长度为 *k* 的八位组串，这里 *k* 是 RSA 合数模 *n* 的以八位组为计量单位的长度

出错提示： “消息太长”，“编码出错”

步骤：

1. *EMSA-PSS 编码：* 将 EMSA-PSS 编码运算（见 9.1.1 部分）应用于消息 *M*，从而产生一个长度为 $\lceil (modBits - 1)/8 \rceil$ 个八位组的编码消息 *EM*，以至于整数 OS2IP (*EM*)的比特长度至少是 *modBits* – 1，其中 *modBits* 是 RSA 合数模的比特长度。

EM = EMSA-PSS-ENCODE (M, $modBits$ – 1) 。

注意如果 $modBits$ – 1 可以被 8 整除，那么 EM 的八位组长度将比 k 小 1；否则就等于 k。如果编码运算输出“ 消息太长 ”，则“ 消息太长 ”然后中止运算。如果编码运算输出“ 编码出错 ”，输出“ 编码出错 ”然后中止运算。

2. *RSA 签名：*

   a. 将编码消息 EM 转换成一个整数消息代表 m （见 4.2 部分）：

   m = OS2IP (EM) 。

   b. 将 RSASP1 签名原语（见 5.2.1 部分）应用于 RSA 私钥 K 以及消息代表 m ，以产生一个整数签名代表 s：

   s = RSASP1 (K, m) 。

   c. 将签名代表 s 转换成长度为 k 个八位组的签名 S（见 4.1 部分）：

   S = I2OSP (s, k) 。

3. *输出签名代表 S。*

8.1.2 签名验证运算

RSASSA-PSS-VERIFY ((*n*, *e*), *M*, *S*)

输入： (*n*, *e*) 签名者的 RSA 公钥

M 签名待验证的消息，是一个八位组串

S 待验证的签名，是一个长度为 k 的八位组串，其中 k 是 RSA 合数模 n 的八位组长度

输出： “ 有效的签名 ”或者“ 无效的签名 ”

步骤：

1. *长度检查：* I 如果签名 S 的长度不是 k 个八位组，则输出“ 无效的签名 ”然后中止运算。.

2. *RSA 验证：*

   a. 将签名 S 转换为一个整数签名代表 s（见 4.2 部分）：

   s = OS2IP (S) 。

   b. 将 RSAVP1 验证原语作用于 RSA 公钥(n, e)和签名代表，从而产生一个整数消息代表 m ：

   m = RSAVP1 ((*n*, *e*), *s*) 。

   如果 RSAVP1 输出“ 签名代表超出范围 ”，则输出“ 有效的签名 ”然后中止运算。

c. 将消息代表 *m* 转换成长度为 *emLen* = ⌈(*modBits* – 1)/8⌉个八位组的编码消息 *EM*，其中 *modBits* 是 RSA 合数模 *n* 的比特长度（见 4.1 部分）：

$$EM = \text{I2OSP}\ (m, emLen)\ 。$$

注意如果 *modBits* – 1 能被 8 整除，那么 *emLen* 将比 *k* 小 1；否则 *emLen* 就等于。如果输出“整数太大”，则输出“无效的签名”然后中止运算。

3. *EMSA-PSS 验证：* 将 EMSA-PSS 验证运算（见 9.1.2）作用于消息 *M* 和编码消息 *EM*，以辨别它们是否一致：

$$Result = \text{EMSA-PSS-VERIFY}\ (M, EM, modBits - 1)\ .$$

4. 如果 *Result* = “consistent”，则输出“有效签名”。否则，输出“无效的签名”。

8.2 RSASSA-PKCS1-v1_5

RSASSA-PKCS1-v1_5 将 RSASP1 原语 、RSAVP1 原语和 EMSA-PKCS1-v1_5 编码方法结合起来了。它与在 IEEE Std 1363-2000 [26]中定义的 IFSSA 方案兼容，方案中的签名和验证原语是 IFSP-RSA1 和 IFVP-RSA1，而消息编码方法是 EMSA-PKCS1-v1_5（这在 IEEE Std 1363-2000 中未给出定义，但在 IEEE P1363a draft 草案[27]中给出定义）。

RSASSA-PKCS1-v1_5 可操作的消息的长度可以是不受限制的也可以是有一个非常大的数约束的，这取决于 EMSA-PKCS1-v1_5 方法的所依赖的散列函数。

假设计算 e^{th} roots modulo *n* 是不可行的以及在 EMSA-PKCS1-v1_5 中的散列函数有适当的属性，就可以推测 RSASSA-PKCS1-v1_5 提供了安全签名。进一步说就是，在不知道 RSA 私钥的情况下伪造签名被认为是计算上不可行的。同样地，在编码方法 EMSA-PKCS1-v1_5 中，散列函数标识被嵌入到编码过程中。因为这个特征，对手为了找到一个与先前已签名消息具有相同签名的消息，必须找到正在使用的特殊散列函数的冲突数据；对对手来说攻击与签名者选择的散列函数不同的散列函数是无用的。进一步的讨论参见[34]。

注释：正如 PKCS #1 v1.5 中的注释，EMSA-PKCS1-v1_5 编码方法具有保证编码消息在转换成为一个整数消息代表之后是大数而且至少是某种程度的“随机数”的特征。这一点防止了由 Desmedt 和 Odlyzko [16]提出的攻击，在该攻击中，通过将消息代表分解成一批具有较小值的因子（例如，一批小素数），在消息代表之间产生倍增的联系。Coron、Naccache,和 Stern [15]指出这类攻击的一种增强形式可能在攻击 ISO/IEC 9796-2 签名方案的一些实例方面相当有效。他们也分析了这类攻击用于 EMSA-PKCS1-v1_5 编码方法的复杂性，并且得出结论：当一个攻击需要的运算比在基础散列函数上进行一次冲突搜索还要多（也就是说，多于 2^{80} 次运算），那么这个攻击就是不现实的。Coppersmith、Halevi 和 Jutla [11]继续扩展 Coron *et al.*的攻

击，以攻破 ISO/IEC 9796-1 带恢复消息的签名方案。各种攻击说明了细心构造 RSA 签名原语的输入的重要性，特别是在带恢复消息的签名方案中。如前所说，EMSA-PKCS-v1_5 编码方法明确地包含一个散列运算，并且不是为带恢复消息的签名方案所编写的。此外，尽管没有已知的攻击是针对 EMSA-PKCS-v1_5 编码方法的，仍旧推荐逐步转换到 EMSA-PSS，作为对未来开发的预防措施。

8.2.1 签名生成运算

RSASSA-PKCS1-V1_5-SIGN (*K*, *M*)

输入： *K* 签名者的 RSA 私钥

M 待签名的消息，是一个八位组串

输出： *S* 签名，是一个长度为 *k* 的八位组串，其中 *k* 是 RSA 合数模 *n* 的八位组长度

出错提示： “消息太长”；“RSA 合数模太短”

步骤：

1. *EMSA-PKCS1-v1_5 编码：* 对消息进行 EMSA-PKCS1-v1_5 编码运算（见 9.2 部分）以产生一个长度为 *k* 个八位组的编码消息 *EM* ：

   EM = EMSA-PKCS1-V1_5-ENCODE (M, k) 。

   如果编码运算输出“消息太长”，则输出“消息太长”然后中止运算。如果编码运算输出“期望的编码消息长度太短”，则输出“RSA 合数模太短”然后中止运算。

2. *RSA 签名：*

   a. 将编码消息 *EM* 转换成一个整数消息代表 *m* （参见 4.2 部分）：

      m = OS2IP (EM) 。

   b. 将 RSASP1 签名原语（见 5.2.1）作用于 RSA 私钥 *K* 和消息代表 *m*，从而产生一个整数签名代表 *s*：

      s = RSASP1 (K, m) 。

   c. 将签名代表 *s* 转换成一个长度为 *k* 个八位组的签名 *S*（参见 4.1 部分）：

      S = I2OSP (s, k) 。

3. *输出签名 S*。

8.2.2 签名验证运算

RSASSA-PKCS1-V1_5-VERIFY ((*n*, *e*), *M*, *S*)

输入： (n, e) 签名者的 RSA 公钥

M 签名待验证的消息，是一个八位组串

S 待验证的签名，是一个长度为 k 的八位组串，其中 k 是 RSA 合数模 n 的八位组长度

输出： “有效签名”或者“无效签名”

出错提示： “message too long”; “RSA modulus too short” “消息太长”;“RSA 合数模太短”

步骤：

1. *长度检查：* 如果签名 S 的长度不是 k 个八位组，则输出“无效签名”然后中止运算。
2. *RSA 验证：*
   a. 将签名 S 转换成一个签名代表 s（参见 4.2 部分）：

      s = OS2IP (S) 。

   b. 将 RSAVP1 验证原语（参见 5.2.2 部分）作用于 RSA 公钥(n, e)和签名代表 s，以产生一个整数消息代表：

      m = RSAVP1 ((n, e), s) 。

      如果 RSAVP1 输出“签名代表超出范围”，则输出“无效签名”然后中止运算。

   c. 将签名代表 m 转换成一个长度为 k 个八位组的编码消息 EM'（参见 0 部分）：

      EM' = I2OSP (m, k) 。

      如果 I2OSP 输出“整数太长”，则输出“无效签名”然后中止运算。
3. *EMSA-PKCS1-v1_5 编码:* 对消息 M 进行 EMSA-PKCS1-v1_5 编码运算（见 0 部分），从而产生另一个长度为 k 个八位组编码消息 EM'：

   EM' = EMSA-PKCS1-v1_5-ENCODE (M, k) .

   如果编码运算输出“消息太长”，则输出“消息太长”并且中止运算。如果编码运算输出“期望的编码消息长度太短”，则输出“RSA 合数模太短”然后中止运算。
4. 比较编码消息 EM 和另一个编码消息 EM'。如果他们相同，则输出“有效签名”；否则，输出“无效签名”。

注释。实现签名验证运算的另一个方法是对编码消息进行一次“解码”运算（在本篇文档中没有定义），以恢复基础散列值，然后将它与一个新计算的散列值比较。这样做的优点在与它需要更少的中间存储（是两个散列值而非两个编码消息），而缺点是它需要另外的代码。

9 带附属的签名的编码方法

编码方法由在八位组串消息和八位组串编码消息之间进行变换的运算构成，而在方案中八位组串编码消息和证书消息代表互相转换。整数消息代表是通过原语进行转换的。因此编码方法在处理消息的方案和原语之间提供了连接。

就本篇文档而言，一个带附属的签名的编码方法由一个编码操作和一个可选的验证操作组成。一个编码操作将一个消息 *M* 变换为一个规定长度的编码消息 *EM*。一个验证运算决定一个消息 *M* 和一个编码消息 *EM* 是否一致，也就是说，编码消息 *EM* 是否是消息 *M* 的有效编码。

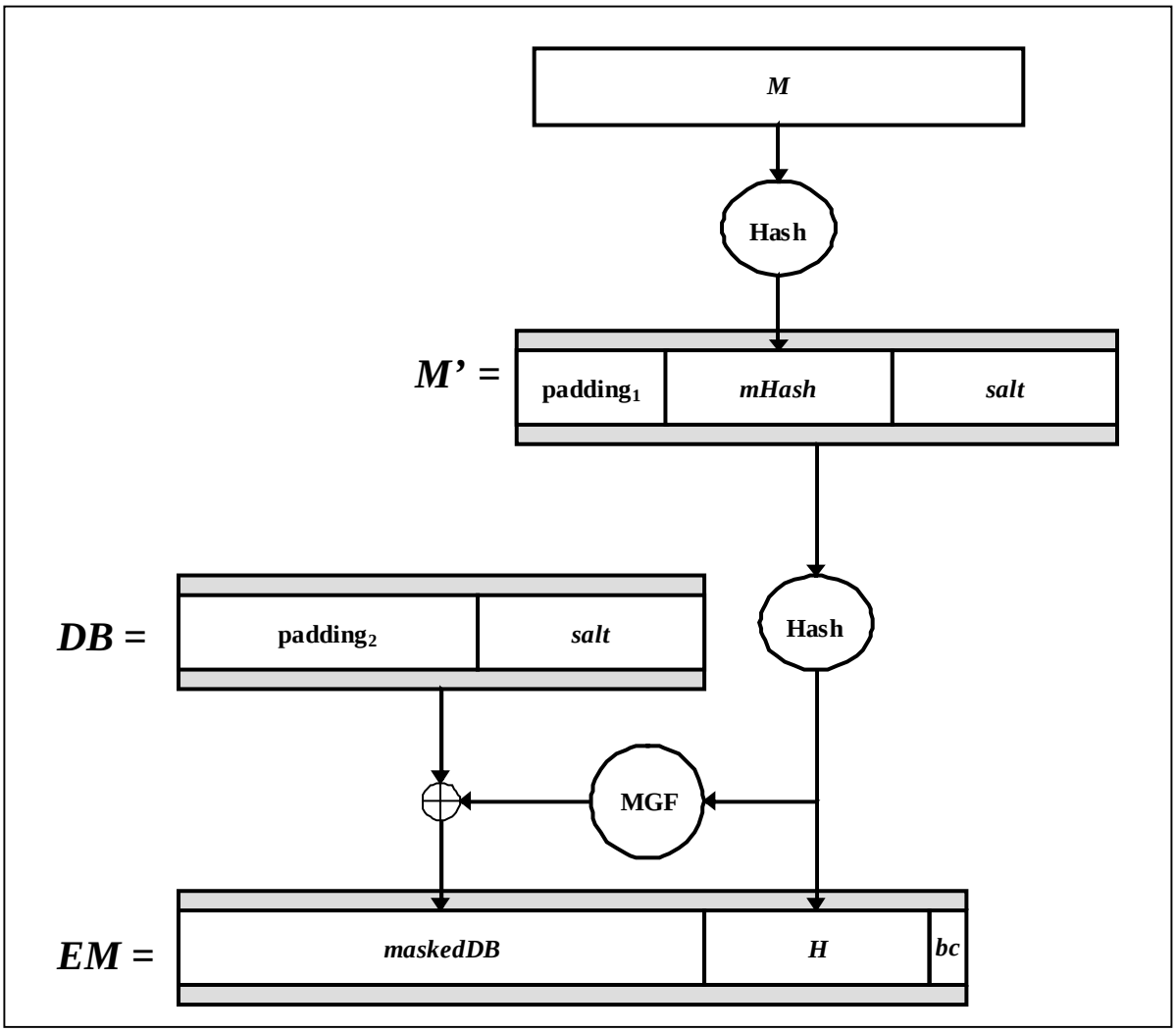
编码运算可能引入一些随机因素，以便对于同一个消息的进行编码运算的不同应用程序将产生不同的编码消息，这一点有利于可证实的安全性。对于这样一个编码方法，同时需要一个编码运算和一个验证运算，除非验证者能够重新产生随机因素（例如，通过从签名者那里获得 salt 的值）。对于确定的编码方法，只需要一个编码运算。

在签名方案中使用了两个带附属的签名的编码方法，这里定义为：EMSA-PSS 和 EMSA-PKCS1-v1_5。

9.1 EMSA-PSS

通过选择散列函数、掩模生成函数和 salt 的长度，可以是编码方法参数化。对于给定的 RSA 密钥，除了 salt 的长度可变之外，其它选项应该是固定的（讨论参见[31]）。在附录 B 中给出了建议的散列函数和掩模生成函数。这个编码方法是基于 Bellare 和 Rogaway 的概率签名方案（PSS：Probabilistic Signature Scheme）[4][5]。这个编码方案被随机化了，而且有一个编码运算和一个验证运算。

0 说明了这个编码运算。



0: EMSA-PSS 编码运算。验证运算遵循相反的步骤，从而恢复出 *salt* ，然后推进步骤以重新计算和比较 *H*。

注释：

1．这里定义的编码方法与 Bellare 和 Rogaway 给 IEEE P1363a [5]的建议中的编码方法在三个方面不同：

- 它对消息使用散列函数而不是掩模生成函数。尽管掩模生成函数是基于一个散列函数的，但直接 使用一个散列函数似乎更加自然。
- 与 salt 值一道被散列化的值是串(0x)00 00 00 00 00 00 00 00 || *mHash* 而不是消息 *M* 本身。这里，*mHash* 是 *M* 的散列。注意在两个步骤中散列函数是相同的。进一步的讨论见下面的注释 3。（同样，使用名称“ salt ”而不是“ seed ”是由于“ salt ”更能体现该值的角色。）

- EMSA-PSS 中的编码消息有九个固定的比特；第一个比特是 0，最后八个比特形成一个“尾部域”，即八位组 0xbc。在最初的方案中，只有第一个比特是固定的。尾部域的基本原理是为了与 IEEE Std 1363-2000 [26]中的 Rabin-Williams IFSP-RW 签名原语以及在草案 ISO/IEC 9796-2 [29]中的对应原语兼容。

2．假设掩模生成函数是基于一个散列函数的，建议这个散列函数与应用于消息的散列函数一样；进一步的讨论参见 8.1 部分。

3．在没有危及 RSASSA-PSS 的安全坚固性的情况下，可以在计算签名运算的其余部分的模块之外执行 EMSA-PSS-ENCODE 的第 1 和第 2 步以及 EMSA-PSS-VERIFY（散列函数对消息的应用），以便该模块的输入是 *mHash* 而不是消息 *M* 本身。换句话说，即使对手能够控制 *mHash* 的值，仍能保持 RSASSA-PSS 的安全坚固性。如果这个模块限制了 I/O 带宽（例如，一块智能卡），那么这是方便的。注意到 PSS[4][5]的先前版本没有这性质。当然，让其它安全理由使这个模块处理整个消息是件吸引人的事。举例来说，如果这个模块不信任负责计算散列值的组件，它可能需要“看到”它正在签名的是什么。

4．salt 的典型的八位组长度是 *hLen*（散列函数 Hash 的输出的长度）和 0。在两种情况下，RSASSA-PSS 的安全性与使 RSAVP1 反向运算的难度密切相关。Bellare 和 Rogaway[4]为最初的 RSA-PSS 方案指定了一个极度低级的界限，这粗略地与前一种情况对应；而 Coron [12]为相关的全域散列法方案制定了一个较低级的界限，这粗略地与后一种情况对应。在[13]中 Coron 提供了一个通用的处理各种 salt 长度（从 0 到 *hLen*）的方法；详细讨论参见[27]。同样参见[31]，它改编了[4][13]中的安全坚固性，以提出 RSA-PSS 最初的和目前的版本的不同之处（就是上面注释 1 中所列的）。

5．就像在 IEEE P1363a [27]中注释的，在签名方案中使用随机化——诸如 EMSA-PSS 中的 salt 值——可以为传送信息而不是被签名的消息提供一个“变换通道（covert channel）”。如需知道更多关于变换通道的信息，参见[50]。

9.1.1 编码运算

EMSA-PSS-ENCODE (*M*, *emBits*)

选项：
- Hash 散列函数（*hLen* 表示散列函数的输出的八位组长度）
- MGF 掩模生成函数
- *sLen* 期望的 salt 的八位组长度

输入：
- *M* 待编码的消息，是一个八位组串
- *emBits* 整数 OS2IP(*EM*)的最大比特长度（见 4.2 部分），至少为 $8hLen + 8sLen + 9$

输出：
- *EM* 编码后的消息，是一个长度为 $emLen = \lceil emBits/8 \rceil$ 的八位组串

出错提示： “编码出错”；“消息太长”

步骤：

1. 如果 *M* 的长度超出散列函数的输入限制（SHA-1 的限制是 $2^{61} - 1$），则输出“ 消息太长 ”并且中止运算。
2. 使 *mHash* = Hash (*M*)，这是一个长度为 *hLen* 的八位组串。
3. 如果 *emLen* < *hLen* + *sLen* + 2，输出“ 编码出错 ”然后中止运算。
4. 生成一个随机的长度为 *sLen* 的八位组串 salt；如果 *sLen* = 0，那么 *salt* 是一个空串。
5. 使

   $$M' = (0x)00\ 00\ 00\ 00\ 00\ 00\ 00\ 00 \| mHash \| salt;$$

   *M'*是一个长度为 length 8 + *hLen* + *sLen* 的八位组串，且开始的八个八位组均为 0。
6. 使 *H* = Hash (*M'*)，这是一个长度为 *hLen* 的八位组串。
7. 生成一个由 *emLen* – *sLen* – *hLen* – 2 个值为零的八位组构成的八位组串 *PS*。*PS* 的长度可以为 0。
8. 使 *DB* = *PS* || 0x01 || *salt*； *DB* 是一个长度为 *emLen* – *hLen* – 1 的八位组串。
9. 使 *dbMask* = MGF (*H*, *emLen* – *hLen* – 1)。
10. 使 *maskedDB* = *DB* ⊕ *dbMask*。
11. 把 *maskedDB* 的最左边的一个八位组中的最左边的 8*emLen* – *emBits* 位置 0。
12. 使 *EM* = *maskedDB* || *H* || 0xbc。
13. 输出 *EM*。

9.1.2 验证操作

EMSA-PSS-VERIFY (*M*, *EM*, *emBits*)

选项：

	Hash	散列函数（*hLen* 散列函数的输出的八位组长度）
	MGF	掩模生成函数
	sLen	期望的 salt 的八位组长度

输入：

	M	待验证的消息，是一个八位组串
	EM	编码消息，是一个长度为 *emLen* = ⌈*emBits*/8⌉的八位组串
	emBits	整数 OS2IP (*EM*)（参见 0 部分）的最大比特长度，至少是 8*hLen* + 8*sLen* + 9

输出： “ 一致 ”或者“ 不一致 ”

步骤：

1. 如果 *M* 的长度大于散列函数的输入限制（SHA-1 的输入限制是 $2^{61} - 1$），则输出“不一致”然后中止运算。
2. 使 *mHash* = Hash (*M*)，是一个长度为 *hLen* 的八位组串。
3. 如果 *emLen* < *hLen* + *sLen* + 2，则输出“不一致”然后中止运算。
4. 如果 *EM* 最右边的八位组的十六进制值不是 0xbc ，则输出“不一致”然后中止。
5. 使 *maskedDB* 成为 *EM* 最左边的 *emLen* – *hLen* – 1 个八位组，而且使 *H* 成为接下来的 *hLen* 个八位组。
6. 如果 *maskedDB* 最左边的八位组的最左边的 8*emLen* – *emBits* 比特不全为零，则输出“不一致”然后中止运算。
7. 使 *dbMask* = MGF (*H*, *emLen* – *hLen* – 1)。
8. 使 *DB* = *maskedDB* ⊕ *dbMask*。
9. 将 *DB* 最左边的八位组中的左边数起 8*emLen* – *emBits* 个比特置零。
10. 如果 *DB* 左边数起的 *emLen* – *hLen* – *sLen* – 2 个八位组不为零或者如果第 *emLen* – *hLen* – *sLen* – 1 个（最左边的八位组是第一个八位组）八位组的十六进制值不为 0x01，则输出“不一致”然后中止运算。
11. 使 *salt* 成为 *DB* 的后 *sLen* 个八位组。
12. 使

$$M' = (0x)00\ 00\ 00\ 00\ 00\ 00\ 00\ 00 \| mHash \| salt \ ;$$

*M'*是一个长度为 8 + *hLen* + *sLen* 的八位组串，且开头八个八位组的值为零。

13. 使 *H'* = Hash (*M'*)，这是一个长度为 *hLen* 的八位组串。
14. 如果 *H* = *H'*，则输出“一致”；否则，输出“不一致”。

9.2 EMSA-PKCS1-v1_5

这个编码方法是确定性的，只有一个编码运算。

EMSA-PKCS1-v1_5-ENCODE (*M*, *emLen*)

选项：

Hash 散列函数（*hLen* 表示散列函数输出的八位组长度）

输入：

M 待编码的消息

emLen 期望的编码后消息的八位组长度，至少为 *tLen* + 11，其中 *tLen* 是在编码运算过程中计算的某个值得 DER 编码 *T* 的八位组长度。

输出： *EM* 编码后的消息，是一个长度为 *emLen* 的八位组串。

出错提示： “消息太长”；“期望的编码消息长度太短”

步骤：

1. 将消息 *M* 代入散列函数，从而产生散列值 *H* ：

   $$H = \text{Hash}(M)$$ 。

   如果散列函数输出“消息太长”则输出“消息太长”然后中止运算。

2. 用可识别的编码规则（DER）将散列函数的算法标识和散列值编码成一个类型为 **DigestInfo** 的 ASN.1 值（见附录 A.2.4），其中类型 **DigestInfo** 具有语法

   ```
   DigestInfo ::= SEQUENCE {
       digestAlgorithm AlgorithmIdentifier,
       digest OCTET STRING
   }
   ```

   第一个域标识散列函数，第二个域包含散列函数。使 *T* 等于 **DigestInfo** 值的 DER 编码（参见下面的注释），且让 *tLen* 等于 *T* 的八位组长度。

3. 如果 *emLen* < *tLen* + 11，则输出“期望的编码消息长度太短”然后中止运算。

4. 生成一个由 *emLen* – *tLen* – 3 个十六进制值为 0xff 的八位组构成的串 *PS* 。*PS* 的长度将至少为八个八位组。

5. 连接 *PS*、 DER 编码 *T* 和其它填充以形成编码消息 *EM*

   $$EM = 0x00 \| 0x01 \| PS \| 0x00 \| T$$ 。

6. 输出 *EM*。

注释：.

1. 对于附录 0 中提到的六个散列函数，**DigestInfo** 值的 DER 编码 *T* 的值如下所示：

   MD2: **(0x)30 20 30 0c 06 08 2a 86 48 86 f7 0d 02 02 05 00 04 10** ‖ *H*。

   MD5: **(0x)30 20 30 0c 06 08 2a 86 48 86 f7 0d 02 05 05 00 04 10** ‖ *H*。

   SHA-1: **(0x)30 21 30 09 06 05 2b 0e 03 02 1a 05 00 04 14** ‖ *H*。

   SHA-256: **(0x)30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20** ‖ *H*。

   SHA-384: **(0x)30 41 30 0d 06 09 60 86 48 01 65 03 04 02 02 05 00 04 30** ‖ *H*。

   SHA-512: **(0x)30 51 30 0d 06 09 60 86 48 01 65 03 04 02 03 05 00 04 40** ‖ *H*。

2. 在本片文档的版本 1.5 中，*T* 被定义为是 **DigestInfo** 值的 BER 编码，而不是 **DigestInfo** 值的 DER 编码。特别的，至少在理论上，本篇文档中定义的验证运算不可能拒绝一个有效签名（考虑到在 PKCS #1 v1.5 中给定的规范）。如果将除了 DER 之外的其它规则应用于 **DigestInfo** （例如，基础 **SEQUENCE** 类型的无限定长度编码法），将出现拒绝有效签名的情况。尽管在实际当中不会发生此类事情，在应用基于 BER 解码运算（在中有定义）的验证运算时，可以选择谨慎的设备（cautious implementer）。以这种方式，可以获得对任何基于 PKCS #1 v1.5 的有效实现的兼容。这样一个验证操作应该指出基础 BER 编码法是否是 DER 编码法，由此指出考虑到本篇文档中给出的规范，签名是否是有效的。

A ASN.1 语法

A.1 RSA 密钥表示

这一节定义 RSA 公钥和 RSA 私钥的 ASN.Q 对象标识符，并且定义类型 **RSAPublicKey** 和 **RSAPrivateKey** 。这些定义的期望应用程序包括 X.509 认证、PKCS #8 [46] 和 PKCS #12 [47]。

对象标识符 **rsaEncryption** 将 RSA 公钥和私钥定义为附录 0 和 0 中所示。与类型为 **AlgorithmIdentifier** 的值中的 OID 相联系的 **parameters** 域应该具有类型为 **NULL** 的值。

```
rsaEncryption    OBJECT IDENTIFIER ::= { pkcs-1 1 }
```

在本节中的定义已被扩展，以支持多素数 RSA，但是具有与先前版本向后兼容的特点。

A.1.1 RSA 公钥语法

应该用 ASN.1 类型 **RSAPublicKey** 来表示一个 RSA 公钥：

```
RSAPublicKey ::= SEQUENCE {
   modulus           INTEGER,  -- n
   publicExponent    INTEGER   -- e
}
```

类型 RSAPublicKey 的域具有以下意义：

- **modulus** 是 RSA 的合数模 *n*。
- **publicExponent** 是 RSA 公开幂 *e*。

A.1.2 RSA 私钥语法

应该用 ASN.1 类型 **RSAPrivateKey** 来表示一个 RSA 私钥：

```
RSAPrivateKey ::= SEQUENCE {
    version          Version,
    modulus          INTEGER,  -- n
    publicExponent    INTEGER,  -- e
    privateExponent   INTEGER,  -- d
    prime1           INTEGER,  -- p
    prime2           INTEGER,  -- q
    exponent1         INTEGER,  -- d mod (p-1)
    exponent2         INTEGER,  -- d mod (q-1)
    coefficient       INTEGER,  -- (inverse of q) mod p
    otherPrimeInfos   OtherPrimeInfos OPTIONAL
}
```

类型 **RSAPrivateKey** 的各域具有以下意义：

- **version** 是版本号，为了与本文档的今后版本兼容。本篇文档的这个版本号应该是 0，如果使用了多素数，则版本号应该是 1。

  ```
  Version ::= INTEGER { two-prime(0), multi(1) }
    (CONSTRAINED BY {-- version must be multi if otherPrimeInfos present --})
  ```

- **modulus** 是 RSA 合数模 n。
- **publicExponent** 是 RSA 的公开幂 e。
- **privateExponent** 是 RSA 的私有幂 d。
- **prime1** 是 n 的素数因子 p。
- **prime2** i 是 n 的素数因子 q。
- **exponent1** 等于 $d \bmod (p-1)$。
- **exponent2** 等于 $d \bmod (q-1)$。
- **coefficient** 是 CRT 系数 $q^{-1} \mod p$。
- **otherPrimeInfos** 按顺序包含了其它素数 $r_3, \ldots, r_u$ 的信息。如果 **version** 是 0 ，它应该被忽略；而如果 **version** 是 1，它应该至少包含 **OtherPrimeInfo** 的一个实例。

  ```
  OtherPrimeInfos ::= SEQUENCE SIZE(1..MAX) OF OtherPrimeInfo


  OtherPrimeInfo ::= SEQUENCE {
     prime           INTEGER,  -- ri
     exponent         INTEGER,  -- di
     coefficient      INTEGER   -- ti
  }
  ```

OtherPrimeInfo 的各域具有以下意义：

- **prime** 是 n 的一个素数因子 r_i ，其中 $i \geq 3$。
- **exponent** 是 $d_i = d \bmod (r_i - 1)$。
- **coefficient** 是 CRT 系数 $t_i = (r_1 \cdot r_2 \cdot \ldots \cdot r_{i-1})^{-1} \bmod r_i$。

注释：重要的是要防止 RSA 私钥被泄漏或是修改。这类防御技术超出了本篇文档的范围。PKCS #12 和#15 中描述了存储和分发私钥的方法以及其它密码数据。

A.2 方案标识

本节定义加密和签名方案的对象标识。与 PKCS #1 v1.5 兼容的方案与 PKCS #1 v1.5 中的方案具有相同的标识。这些定义期望的应用程序包括 X.509 认证和 PKCS #7。

这里是对 PKCS #1 OID 的类型表示的定义：

```
PKCS1Algorithms    ALGORITHM-IDENTIFIER ::= {
   { OID rsaEncryption            PARAMETERS NULL }    |
   { OID md2WithRSAEncryption      PARAMETERS NULL }   |
   { OID md5WithRSAEncryption      PARAMETERS NULL }   |
   { OID sha1WithRSAEncryption     PARAMETERS NULL }   |
   { OID sha256WithRSAEncryption    PARAMETERS NULL }  |
   { OID sha384WithRSAEncryption    PARAMETERS NULL }  |
   { OID sha512WithRSAEncryption    PARAMETERS NULL }  |
   { OID id-RSAES-OAEP PARAMETERS RSAES-OAEP-params } |
   PKCS1PSourceAlgorithms                              |
   { OID id-RSASSA-PSS PARAMETERS RSASSA-PSS-params } ,
   ...  -- Allows for future expansion --
}
```

A.2.1 RSAES-OAEP

对象标识 **id-RSAES-OAEP** 标识了 RSAES-OAEP 加密方案。

```
id-RSAES-OAEP    OBJECT IDENTIFIER ::= { pkcs-1 7 }
```

在类型为 **AlgorithmIdentifier** 的值中，与这个 OID 相关的 **parameters** 域应该具有类型为 **RSAES-OAEP-params** 的值：

```
RSAES-OAEP-params ::= SEQUENCE {
   hashAlgorithm     [0] HashAlgorithm    DEFAULT sha1,
   maskGenAlgorithm   [1] MaskGenAlgorithm  DEFAULT mgf1SHA1,
```

```
   pSourceAlgorithm   [2] PSourceAlgorithm  DEFAULT pSpecifiedEmpty
}
```

类型 **RSAES-OAEP-params** 的域具有以下意义：

- **hashAlgorithm** 标识了散列函数。在 **OAEP-PSSDigestAlgorithms** 集中，它应该是一个具有 OID 的算法 ID。关于所支持的散列函数的讨论，请参见附录 B.1。

```
HashAlgorithm ::= AlgorithmIdentifier { {OAEP-PSSDigestAlgorithms} }

OAEP-PSSDigestAlgorithms   ALGORITHM-IDENTIFIER ::= {
   { OID id-sha1 PARAMETERS NULL   }|
   { OID id-sha256 PARAMETERS NULL }|
   { OID id-sha384 PARAMETERS NULL }|
   { OID id-sha512 PARAMETERS NULL },
   ...  -- Allows for future expansion --
}
```

默认的散列函数是 SHA-1：

```
sha1   HashAlgorithm ::= {
   algorithm   id-sha1,
   parameters  SHA1Parameters : NULL
}

SHA1Parameters ::= NULL
```

- **maskGenAlgorithm** 标识掩模生成函数。它应该是一个算法 ID（具有一个属于 **PKCS1MGFAlgorithms** 集（对于这个版本来说，它应该由 **id-mgf1** 构成）的 OID），以标识掩模生成函数 MGF1（参见附录 B.2.1）。与 **id-mgf1** 联系的 **parameters** 域应该是一个算法 ID（具有一个属于 **OAEP-PSSDigestAlgorithms** 集的 OID），标识 MGF1 基于的散列函数。

```
MaskGenAlgorithm ::= AlgorithmIdentifier { {PKCS1MGFAlgorithms} }

PKCS1MGFAlgorithms   ALGORITHM-IDENTIFIER ::= {
   { OID id-mgf1 PARAMETERS HashAlgorithm },
   ...  -- Allows for future expansion --
}
```

默认的掩模函数是基于 SHA-1 的 MGF1：

```
mgf1SHA1   MaskGenAlgorithm ::= {
   algorithm   id-mgf1,
   parameters  HashAlgorithm : sha1
}
```

- **pSourceAlgorithm** 标识标签 *L* 的源（也可能是值）。它应该是一个算法标识（具有一个属于 **PKCS1PSourceAlgorithms** 集（对这个版本来说，应该由 **id-pSpecified** 构成）的对象标识）， 指出这个标签被明确描述。与 **id-pSpecified** 联系的 **parameters** 域应该是一个类型为 **OCTET STRING** 的值，包含这个标签。这篇规范的先前版本中，使用的是术语“ 编码参数 ”而不是“ 标签 ”，从此就出现了下面这个类型的名称。

```
PSourceAlgorithm ::= AlgorithmIdentifier { {PKCS1PSourceAlgorithms} }

PKCS1PSourceAlgorithms   ALGORITHM-IDENTIFIER ::= {
   { OID id-pSpecified PARAMETERS EncodingParameters },
   ...  -- Allows for future expansion --
}

id-pSpecified   OBJECT IDENTIFIER ::= { pkcs-1 9 }

EncodingParameters ::= OCTET STRING(SIZE(0..MAX))
```

默认标签是一个空串（以至于 *lHash* 会包含空串的散列）：

```
pSpecifiedEmpty   PSourceAlgorithm ::= {
   algorithm   id-pSpecified,
   parameters  EncodingParameters : emptyString
}

emptyString   EncodingParameters ::= ''H
```

如果 **RSAES-OAEP-params** 中的域都使用了默认值，那么算法标识将具有以下的值：

```
rSAES-OAEP-Default-Identifier   RSAES-AlgorithmIdentifier ::= {
   algorithm   id-RSAES-OAEP,
   parameters  RSAES-OAEP-params : {
      hashAlgorithm      sha1,
      maskGenAlgorithm    mgf1SHA1,
      pSourceAlgorithm    pSpecifiedEmpty
   }
```

```
}

RSAES-AlgorithmIdentifier ::= AlgorithmIdentifier { {PKCS1Algorithms} }
```

A.2.2 RSAES-PKCS1-v1_5

对象标识 **rsaEncryption** （参见附录 A.1）标识 RSAES-PKCS1-v1_5 加密方案。在类型为 **AlgorithmIdentifier** 的值中与这个 OID 联系的 **parameters** 域应该具有一个类型为 **NULL** 的值。这与 PKCS #1 v1.5 中的一样。

```
rsaEncryption    OBJECT IDENTIFIER ::= { pkcs-1 1 }
```

A.2.3 RSASSA-PSS

对象标识 **id-RSASSA-PSS** 标识加密方案。

```
id-RSASSA-PSS    OBJECT IDENTIFIER ::= { pkcs-1 10 }
```

在类型为 **AlgorithmIdentifier** 的值中与这个 OID 相联系的 **parameters** 域应该具有一个类型为 **RSASSA-PSS-params** 的值：

```
RSASSA-PSS-params ::= SEQUENCE {
   hashAlgorithm     [0] HashAlgorithm     DEFAULT sha1,
   maskGenAlgorithm  [1] MaskGenAlgorithm  DEFAULT mgf1SHA1,
   saltLength        [2] INTEGER           DEFAULT 20,
   trailerField      [3] TrailerField      DEFAULT trailerFieldBC
}
```

类型 **RSASSA-PSS-params** 的各域具有以下意义：

- **hashAlgorithm** 标识散列函数。它应该是一个算法 ID（具有一个属于 **OAEP-PSSDigestAlgorithms** 集（参见附录 0）的 OID）。默认的散列函数是 SHA-1。
- **maskGenAlgorithm** 标识掩模生成函数。它应该是一个算法 ID（具有一个属于 **PKCS1MGFAlgorithms** 集（参见附录 0）的 OID）。默认的掩模生成函数是一个基于 SHA-1 的 MGF1。对于 MGF1（更一般地，对于基于一个散列函数地掩模生成函数）建议基础散列函数与 **hashAlgorithm** 标识的散列函数一样；进一步的说明请参见 9.1 节的注释 2。

- **saltLength** 是 salt 的八位组长度。它应该是个整数。对于一个给定的 **hashAlgorithm**，**saltLength** 的默认值是这个散列值的八位组长度。**saltLength** 不像类型 **RSASSA-PSS-params** 的其它域，它的值对指定的 RSA 密钥对来说不需要是固定值。
- **trailerField** 是尾部域数字，为了与草案 IEEE P1363a [27]兼容。对与本文档的这个版本它应该是 1，这表示尾部域具有十六进制值 0xbc 。本篇文档不支持其它尾部域（包括 IEEE P1363a 中的尾部域 *HashID* || 0xcc）。

  ```
  TrailerField ::= INTEGER { trailerFieldBC(1) }
  ```

如果 **hashAlgorithm**、**maskGenAlgorithm** 和 **RSASSA-PSS-params** 的 **trailerField** 域使用默认值，那么算法标识将具有以下值：

```
rSASSA-PSS-Default-Identifier    RSASSA-AlgorithmIdentifier ::= {
   algorithm   id-RSASSA-PSS,
   parameters  RSASSA-PSS-params : {
      hashAlgorithm      sha1,
      maskGenAlgorithm    mgf1SHA1,
      saltLength         20,
      trailerField       trailerFieldBC
   }
}

RSASSA-AlgorithmIdentifier ::= AlgorithmIdentifier { {PKCS1Algorithms} }
```

注释：在一些应用中，作为一个签名方案的散列函数脱离签名方案中的其它操作而独立标识。举例来说，在 PKCS #7 [45]中，一个散列函数标识放在消息前面，而“摘要加密”算法标识（指示其它操作）携带在签名中。为了 PKCS #7 让支持 RSASSA-PSS 签名方案，在 RSASSA-PSS 中需要用一个对象标识指示在散列函数（类似于 RSASSA-PKCS1-v1_5 方案的 **RSAEncryption** OID）之后的操作。S/MIME CMS [25]采用了不同的方式。尽管散列函数标识放在消息前面，但是全部签名方案的算法标识可能携带在 CMS 签名中（这是对于 DSA 签名）。在这个惯例之后，**id-RSASSA-PSS** OID 被用于标识 CMS 中的 RSASSA-PSS 签名。自从 CMS 被认为是 PKCS #7 的接班人之后，同时考虑到 CMS 而不是 PKCS #7，进行了许多新开发，诸如增加对 RSASSA-PSS 支持。

A.2.4 RSASSA-PKCS1-v1_5

下面之一应该是 RSASSA-PKCS1-v1_5 的对象标识。OID 的选择依赖于散列算法的选择：MD2、MD5、SHA-1、 SHA-256、SHA-384 或者 SHA-512。注意如果采用了 MD2 或者 MD5，那么这个 OID 就和 PKCS

#1 v1.5 中的一样。对于每个 OID，在类型为 **AlgorithmIdentifier** 的值中，与这个 OID 相联系的 **parameters** 域应该具有一个类型为 **NULL** 的值。应该依照下表选择这个 OID：

Hash algorithm	OID
MD2	md2WithRSAEncryption ::= {pkcs-1 2}
MD5	md5WithRSAEncryption ::= {pkcs-1 4}
SHA-1	sha1WithRSAEncryption ::= {pkcs-1 5}
SHA-256	sha256WithRSAEncryption ::= {pkcs-1 11}
SHA-384	sha384WithRSAEncryption ::= {pkcs-1 12}
SHA-512	sha512WithRSAEncryption ::= {pkcs-1 13}

EMSA-PKCS1-v1_5 编码方法包括一个类型为 **DigestInfo** 的 ASN.1 值，其中类型 **DigestInfo** 具有语法

```
DigestInfo ::= SEQUENCE {
   digestAlgorithm DigestAlgorithm,
   digest OCTET STRING
}
```

digestAlgorithm 标识散列函数且应是一个算法 ID（具有一个属于 **PKCS1-v1-5DigestAlgorithms** 集的 OID）。关于支持的散列函数的讨论，参见附录 0。

```
DigestAlgorithm ::= AlgorithmIdentifier { {PKCS1-v1-5DigestAlgorithms} }

PKCS1-v1-5DigestAlgorithms   ALGORITHM-IDENTIFIER ::= {
   { OID id-md2 PARAMETERS NULL   }|
   { OID id-md5 PARAMETERS NULL   }|
   { OID id-sha1 PARAMETERS NULL  }|
   { OID id-sha256 PARAMETERS NULL }|
   { OID id-sha384 PARAMETERS NULL }|
   { OID id-sha512 PARAMETERS NULL }
}
```

B 支撑技术

本部分给出了几个关于支持第 7 部分中的加密方案和第 9 部分中的编码方案的基础函数的例子。为了移植到新技术上也为了与已存在的应用程序兼容，这里给出了一个技术范围。尽管这些支撑技术适合用应用程序实现，但是它们之中没有一个需要被实现。希望开发描述特定技术的 PKCS #1 v2.1 文档。

这个部分也给支撑技术指定了对象标识。

B.1 散列函数

第 0 部分和第 0 部分的运算使用了散列函数。散列函数是确定性的，意思是输出完全有输入决定。散列函数接纳可变长度的八位组串，然后生成固定长度的八位组串。第 0 部分和第 0 部分的运算使用的散列函数一般应该是限制冲突的。这意味着要找到有相同输出的散列函数的两个不同的输入是不可行的。一个限制冲突散列函数也具有吸引人的单方性能；这意味着给定一个输出，就不可能找到一个输入，使它的散列就事给定的输出。除了这些技术规格，用伪随机输出散列函数会产生一个掩模生成函数（见附录 0）。

本篇文档中给出六个编码函数的散列函数作为例子：MD2 [33]、 MD5 [41]、 SHA-1 [38]以及申请算法 SHA-256、SHA-384 和 SHA-512[39]。对于 RSAES-OAEP 加密方案和 EMSA-PSS 编码方案，只推荐使用 SHA-1 和 SHA-256/384/512，新应用程序推荐使用 SHA-1 和 SHA-256/384/512。MD2 和 MD5 只推荐给与已存在的基于 PKCS #1 v1.5 的应用程序兼容。

对象标识 **id-md2**、 **id-md5**、 **id-sha1**、 **id-sha256**、 **id-sha384**,和 **id-sha512**，分别标识散列函数：

```
id-md2     OBJECT IDENTIFIER ::= {
   iso (1) member-body (2) us (840) rsadsi (113549) digestAlgorithm (2) 2
}

id-md5     OBJECT IDENTIFIER ::= {
   iso (1) member-body (2) us (840) rsadsi (113549) digestAlgorithm (2) 5
}

id-sha1    OBJECT IDENTIFIER ::= {
   iso(1) identified-organization(3) oiw(14) secsig(3) algorithms(2) 26
}

id-sha256   OBJECT IDENTIFIER ::= {
   joint-iso-itu-t (2) country (16) us (840) organization (1) gov (101)
   csor (3) nistalgorithm (4) hashalgs (2) 1
}

id-sha384   OBJECT IDENTIFIER ::= {
   joint-iso-itu-t (2) country (16) us (840) organization (1) gov (101)
   csor (3) nistalgorithm (4) hashalgs (2) 2
}

id-sha512   OBJECT IDENTIFIER ::= {
   joint-iso-itu-t (2) country (16) us (840) organization (1) gov (101)
   csor (3) nistalgorithm (4) hashalgs (2) 3
}
```

在类型为 **AlgorithmIdentifier** 的值中，与这些对象标识联系的 **parameters** 域应该具有类型为 **NULL** 的值。

注释：PKCS #1 的版本 1.5 也允许在签名方案中使用 MD4。MD4 的密码分析在这几年里取得了重大的进展。举例来说，Dobbertin [18]证实了如何找到 MD4 的冲突数据，而且 MD4 的前两 round 不是单方的[20]。由于这些结果以及其它结果（例如[8]），不再推荐使用 MD4。MD2 和 MD5 的密码分析中也有进展，尽管还不足以正名从以存在的应用程序中删除是正当的。Rogier 和 Chauvaud [43]证实了如何在 MD2 的修改版本中找到冲突数据。没有人证实如何找到所有 MD5 算法的冲突数据，尽管部分结果已经被发现（例如[9][19]）。

为了 address these concerns，不推荐新应用程序使用 SHA-1、SHA-256 、SHA-384 或者 SHA-512。就今天来说，针对这些散列函数的最著名的攻击是带复杂度 $2^{L/2}$ 的类属攻击，其中 L 是散列函数输出的比特长度。对于本篇文档中的签名方案，一个冲突数据攻击很容易就转化成一个签名伪造。因此，值 $L / 2$ 应该至少等于签名方案期望的安全等级的比特长度（B 比特安全等级意思是攻击具有的复杂度为 2^B）。同样的拇指规则能够应用于 RSAES-OAEP；建议 seed 的比特长度（等于散列函数输出的比特长度）应该为期望安全等级的比特长度的两倍。

B.2 掩模生成函数

一个掩模生成函数将一个可变长度的八位组串和一个期望的输出长度作为输入，并且输出一个具有期望长度的八位组串。对输入长度方面和输出八位组可能有限制，但是这种界限一般非常大。一个掩模生成函数的数出应该是伪随机的：给定输出而 RSASSA-PSS 不是输入的一部分，预算另一部分输出是不可能的。RSAES-OAEP 和 RSASSA-PSS 的可证实安全性赖于掩模生成函数输出的随机本质，而输出的随机性反过来依赖于基础散列的随机本质。

这里给定一个掩模生成函数：MGF1，它依赖于散列函数。MGF1 与 IEEE Std 1363-2000 [26]以及草案 ANSI X9.44 [1]中定义的掩模生成函数一致。本篇文档的今后版本可以定义其它掩模生成函数。

B.2.1 MGF1

MGF1 是一个基于散列函数的掩模生成函数。

MGF1 (*mgfSeed*, *maskLen*)

选项：	Hash	散列函数（*hLen* 表示散列函数输出的八位组长度）
输入：	*mgfSeed*	掩模生成所用的 seed，使一个八位组串
	maskLen	期望的掩模的八位组长度，至多是 2^{32} *hLen*
输出：	*mask*	掩模，使一个长度为 *maskLen* 的八位组

出错提示： “掩模太长”

步骤：

1. 如果 *maskLen* > 2^{32} *hLen*，则输出“掩模太长”然后中止运算。
2. 使 *T* 为空的八位组串。
3. 使 *counter* 从 0 步进到⌈ *maskLen / hLen* ⌉– 1，循环以下步骤：
   a. 将 *counter* 转换成一个长度为 4 个八位组的串 *C* （见 0 节）：

      C = I2OSP (*counter*, 4) 。

   b. 连接 seed *mgfSeed* 的散列和 *C*，产生一个八位组串 *T*：

      $T = T \| \text{Hash}\ (mgfSeed \| C)$ 。
4. 将 *T* 的前 *maskLen* 个八位组作为八位组串掩模输出。

对象标识 **id-mgf1** 标识了掩模生成函数 MGF1：

```
id-mgf1    OBJECT IDENTIFIER ::= { pkcs-1 8 }
```

在类型为 **AlgorithmIdentifier** 的值中与这个 OID 相联系的 **parameters** 域应该是一个类型为 **hashAlgorithm** 的值，标识 MGF1 所依赖的散列函数。

C ASN.1 模块

```
PKCS-1 {
   iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) modules(0)
   pkcs-1(1)
}

-- $ Revision: 2.1 $

-- This module has been checked for conformance with the ASN.1 standard by
-- the OSS ASN.1 Tools

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL
-- All types and values defined in this module are exported for use in other
```

```
-- ASN.1 modules.

IMPORTS

id-sha256, id-sha384, id-sha512
    FROM NIST-SHA2 {
        joint-iso-itu-t(2) country(16) us(840) organization(1) gov(101)
        csor(3) nistalgorithm(4) modules(0) sha2(1)
    };

-- ============================
--  Basic object identifiers
-- ============================

-- The DER encoding of this in hexadecimal is:
-- (0x)06 08
--       2A 86 48 86 F7 0D 01 01
--
pkcs-1    OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) 1
}

--
-- When rsaEncryption is used in an AlgorithmIdentifier the parameters
-- MUST be present and MUST be NULL.
--
rsaEncryption    OBJECT IDENTIFIER ::= { pkcs-1 1 }

--
-- When id-RSAES-OAEP is used in an AlgorithmIdentifier the parameters MUST
-- be present and MUST be RSAES-OAEP-params.
--
id-RSAES-OAEP    OBJECT IDENTIFIER ::= { pkcs-1 7 }

--
-- When id-pSpecified is used in an AlgorithmIdentifier the parameters MUST
-- be an OCTET STRING.
--
id-pSpecified    OBJECT IDENTIFIER ::= { pkcs-1 9 }

--
-- When id-RSASSA-PSS is used in an AlgorithmIdentifier the parameters MUST
-- be present and MUST be RSASSA-PSS-params.
```

```
--
id-RSASSA-PSS    OBJECT IDENTIFIER ::= { pkcs-1 10 }

--
-- When the following OIDs are used in an AlgorithmIdentifier the parameters
-- MUST be present and MUST be NULL.
--
md2WithRSAEncryption      OBJECT IDENTIFIER ::= { pkcs-1 2 }
md5WithRSAEncryption      OBJECT IDENTIFIER ::= { pkcs-1 4 }
sha1WithRSAEncryption     OBJECT IDENTIFIER ::= { pkcs-1 5 }
sha256WithRSAEncryption   OBJECT IDENTIFIER ::= { pkcs-1 11 }
sha384WithRSAEncryption   OBJECT IDENTIFIER ::= { pkcs-1 12 }
sha512WithRSAEncryption   OBJECT IDENTIFIER ::= { pkcs-1 13 }

--
-- This OID really belongs in a module with the secsig OIDs.
--
id-sha1    OBJECT IDENTIFIER ::= {
   iso(1) identified-organization(3) oiw(14) secsig(3) algorithms(2) 26
}

--
-- OIDs for MD2 and MD5, allowed only in EMSA-PKCS1-v1_5.
--
id-md2 OBJECT IDENTIFIER ::= {
   iso(1) member-body(2) us(840) rsadsi(113549) digestAlgorithm(2) 2
}

id-md5 OBJECT IDENTIFIER ::= {
   iso(1) member-body(2) us(840) rsadsi(113549) digestAlgorithm(2) 5
}

--
-- When id-mgf1 is used in an AlgorithmIdentifier the parameters MUST be
-- present and MUST be a HashAlgorithm, for example sha1.
--
id-mgf1    OBJECT IDENTIFIER ::= { pkcs-1 8 }

-- ================
--   Useful types
-- ================

ALGORITHM-IDENTIFIER ::= CLASS {
```

```
    &id   OBJECT IDENTIFIER  UNIQUE,
    &Type  OPTIONAL
}
    WITH SYNTAX { OID &id [PARAMETERS &Type] }

-- Note: the parameter InfoObjectSet in the following definitions allows a
-- distinct information object set to be specified for sets of algorithms
-- such as:
-- DigestAlgorithms   ALGORITHM-IDENTIFIER ::= {
--    { OID id-md2  PARAMETERS NULL }|
--    { OID id-md5  PARAMETERS NULL }|
--    { OID id-sha1 PARAMETERS NULL }
-- }
--

AlgorithmIdentifier { ALGORITHM-IDENTIFIER:InfoObjectSet } ::= SEQUENCE {
    algorithm
        ALGORITHM-IDENTIFIER.&id({InfoObjectSet}),
    parameters
        ALGORITHM-IDENTIFIER.&Type({InfoObjectSet}{@.algorithm})  OPTIONAL
}



-- ==============
--  Algorithms
-- ==============

--
-- Allowed EME-OAEP and EMSA-PSS digest algorithms.
--
OAEP-PSSDigestAlgorithms   ALGORITHM-IDENTIFIER ::= {
    { OID id-sha1 PARAMETERS NULL   }|
    { OID id-sha256 PARAMETERS NULL }|
    { OID id-sha384 PARAMETERS NULL }|
    { OID id-sha512 PARAMETERS NULL },
    ...  -- Allows for future expansion --
}

--
-- Allowed EMSA-PKCS1-v1_5 digest algorithms.
--
PKCS1-v1-5DigestAlgorithms   ALGORITHM-IDENTIFIER ::= {
```

```
    { OID id-md2 PARAMETERS NULL   }|
    { OID id-md5 PARAMETERS NULL   }|
    { OID id-sha1 PARAMETERS NULL  }|
    { OID id-sha256 PARAMETERS NULL }|
    { OID id-sha384 PARAMETERS NULL }|
    { OID id-sha512 PARAMETERS NULL }
}

sha1   HashAlgorithm ::= {
    algorithm   id-sha1,
    parameters  SHA1Parameters : NULL
}

HashAlgorithm ::= AlgorithmIdentifier { {OAEP-PSSDigestAlgorithms} }

SHA1Parameters ::= NULL

--
-- Allowed mask generation function algorithms.
-- If the identifier is id-mgf1, the parameters are a HashAlgorithm.
--
PKCS1MGFAlgorithms   ALGORITHM-IDENTIFIER ::= {
    { OID id-mgf1 PARAMETERS HashAlgorithm },
    ...  -- Allows for future expansion --
}

--
-- Default AlgorithmIdentifier for id-RSAES-OAEP.maskGenAlgorithm and
-- id-RSASSA-PSS.maskGenAlgorithm.
--
mgf1SHA1   MaskGenAlgorithm ::= {
    algorithm   id-mgf1,
    parameters  HashAlgorithm : sha1
}

MaskGenAlgorithm ::= AlgorithmIdentifier { {PKCS1MGFAlgorithms} }

--
-- Allowed algorithms for pSourceAlgorithm.
--
PKCS1PSourceAlgorithms   ALGORITHM-IDENTIFIER ::= {
    { OID id-pSpecified PARAMETERS EncodingParameters },
    ...  -- Allows for future expansion --
```

```
}

EncodingParameters ::= OCTET STRING(SIZE(0..MAX))

--
-- This identifier means that the label L is an empty string, so the digest
-- of the empty string appears in the RSA block before masking.
--

pSpecifiedEmpty    PSourceAlgorithm ::= {
   algorithm   id-pSpecified,
   parameters  EncodingParameters : emptyString
}

PSourceAlgorithm ::= AlgorithmIdentifier { {PKCS1PSourceAlgorithms} }

emptyString    EncodingParameters ::= ''H

--
-- Type identifier definitions for the PKCS #1 OIDs.
--
PKCS1Algorithms    ALGORITHM-IDENTIFIER ::= {
   { OID rsaEncryption            PARAMETERS NULL } |
   { OID md2WithRSAEncryption      PARAMETERS NULL } |
   { OID md5WithRSAEncryption      PARAMETERS NULL } |
   { OID sha1WithRSAEncryption     PARAMETERS NULL } |
   { OID sha256WithRSAEncryption    PARAMETERS NULL } |
   { OID sha384WithRSAEncryption    PARAMETERS NULL } |
   { OID sha512WithRSAEncryption    PARAMETERS NULL } |
   { OID id-RSAES-OAEP PARAMETERS RSAES-OAEP-params } |
   PKCS1PSourceAlgorithms                          |
   { OID id-RSASSA-PSS PARAMETERS RSASSA-PSS-params } ,
   ...  -- Allows for future expansion --
}

-- ====================
--   Main structures
-- ====================

RSAPublicKey ::= SEQUENCE {
   modulus          INTEGER,  -- n
   publicExponent    INTEGER   -- e
}
```

```
--
-- Representation of RSA private key with information for the CRT algorithm.
--
RSAPrivateKey ::= SEQUENCE {
    version         Version,
    modulus         INTEGER,  -- n
    publicExponent   INTEGER,  -- e
    privateExponent  INTEGER,  -- d
    prime1          INTEGER,  -- p
    prime2          INTEGER,  -- q
    exponent1       INTEGER,  -- d mod (p-1)
    exponent2       INTEGER,  -- d mod (q-1)
    coefficient      INTEGER,  -- (inverse of q) mod p
    otherPrimeInfos   OtherPrimeInfos OPTIONAL
}

Version ::= INTEGER { two-prime(0), multi(1) }
    (CONSTRAINED BY {-- version must be multi if otherPrimeInfos present --})

OtherPrimeInfos ::= SEQUENCE SIZE(1..MAX) OF OtherPrimeInfo


OtherPrimeInfo ::= SEQUENCE {
    prime           INTEGER,  -- ri
    exponent        INTEGER,  -- di
    coefficient      INTEGER   -- ti
}

--
-- AlgorithmIdentifier.parameters for id-RSAES-OAEP.
-- Note that the tags in this Sequence are explicit.
--
RSAES-OAEP-params ::= SEQUENCE {
    hashAlgorithm     [0] HashAlgorithm    DEFAULT sha1,
    maskGenAlgorithm   [1] MaskGenAlgorithm  DEFAULT mgf1SHA1,
    pSourceAlgorithm   [2] PSourceAlgorithm  DEFAULT pSpecifiedEmpty
}

--
-- Identifier for default RSAES-OAEP algorithm identifier.
-- The DER Encoding of this is in hexadecimal:
-- (0x)30 0D
```

```
--       06 09
--          2A 86 48 86 F7 0D 01 01 07
--       30 00
-- Notice that the DER encoding of default values is "empty".
--

rSAES-OAEP-Default-Identifier   RSAES-AlgorithmIdentifier ::= {
   algorithm   id-RSAES-OAEP,
   parameters  RSAES-OAEP-params : {
      hashAlgorithm      sha1,
      maskGenAlgorithm    mgf1SHA1,
      pSourceAlgorithm    pSpecifiedEmpty
   }
}

RSAES-AlgorithmIdentifier ::= AlgorithmIdentifier { {PKCS1Algorithms} }

--
-- AlgorithmIdentifier.parameters for id-RSASSA-PSS.
-- Note that the tags in this Sequence are explicit.
--
RSASSA-PSS-params ::= SEQUENCE {
   hashAlgorithm     [0] HashAlgorithm     DEFAULT sha1,
   maskGenAlgorithm   [1] MaskGenAlgorithm   DEFAULT mgf1SHA1,
   saltLength        [2] INTEGER          DEFAULT 20,
   trailerField      [3] TrailerField      DEFAULT trailerFieldBC
}

TrailerField ::= INTEGER { trailerFieldBC(1) }

--
-- Identifier for default RSASSA-PSS algorithm identifier
-- The DER Encoding of this is in hexadecimal:
-- (0x)30 0D
--       06 09
--          2A 86 48 86 F7 0D 01 01 0A
--       30 00
-- Notice that the DER encoding of default values is "empty".
--
rSASSA-PSS-Default-Identifier   RSASSA-AlgorithmIdentifier ::= {
   algorithm   id-RSASSA-PSS,
   parameters  RSASSA-PSS-params : {
      hashAlgorithm      sha1,
```

```
        maskGenAlgorithm    mgf1SHA1,
        saltLength         20,
        trailerField       trailerFieldBC
    }
}

RSASSA-AlgorithmIdentifier ::= AlgorithmIdentifier { {PKCS1Algorithms} }

--
-- Syntax for the EMSA-PKCS1-v1_5 hash identifier.
--
DigestInfo ::= SEQUENCE {
    digestAlgorithm DigestAlgorithm,
    digest OCTET STRING
}

DigestAlgorithm ::= AlgorithmIdentifier { {PKCS1-v1-5DigestAlgorithms} }

END  -- PKCS1Definitions
```

D 知识产权因素

在美国专利 4,405,829 中描述了 RSA 公钥密码系统，这个专利在 2000 年 9 月 20 日到期。尽管可能会涉及特定的基础技术，但 RSA 安全公司没有对这篇文档中所描述句法结构作出其它专利声明。

在美国专利 5,848,159 中描述了多原语 RSA。

加利福尼亚大学已经指出它拥有 pending on PSS 签名方案[5]的专利。它也向 IEEE P1363 工作组提交了蚁封信，说如果 PSS 签名方案包括在 IEEE 标准中，那么“在那个标准被采用后，加利福尼亚大学就免费将任何 PSS 的一致实现注册为一个获得带附属的数字签名的一项技术” [23]。这个 PSS 签名方案定义在 IEEE P1363 a 草案[27]中，当本篇文档发布时，这个草案还在投票决议中。

如果在所有提及或参考了本篇文档的材料中，将之标识为“RSA 安全公司公开密钥密码标准（PKCS）”，则拷贝本篇文档是得到许可的。

RSA 安全公司考虑到其它方的知识产权声明，则没有提出其它陈述。这个决定是用户的责任。

E 修订历史

版本 1.0 – 1.3

版本 1.0 – 1.3 的发布是为了参与 RSA 数据安全，即 1991 年二月和三月公司的公开密钥密码标准会议。

版本 1.4

版本 1.4 是 1991 年 6 月 3 日首次公开发布的 PKCS 的一部分。版本 1.4 是作为 NIST/OSI 实现的专题讨论会文档 SEC-SIG-91-18 而发布的。

版本 1.5

版本 1.5 编入了几个版本的变化，包括参考文档的更新和增加修订历史。

下面是主要的变化：

— 第 10 部分：增加了“带 RSA 的 MD4”签名和验证运算

— 第 11 部分：增加了对象标识 **md4WithRSAEncryption**

版本 1.5 是作为 IETF RFC 2313 发布的。

版本 2.0

版本 2.0 编入了在文档结构方面的主要编辑变化，而且引入了 RSAES-OAEP 加密方案。尽管由于这几年来的密码发展，不再允许使用散列算法 MD4，但这个版本继续支持版本 1.5 中的加密和签名处理。版本 2.0 是作为 IETF RFC 2437 [35]发布的。

版本 2.1

版本 2.1 伴随着几个编辑上的改进，引入了多素数 RSA 和 RSASSA-PSS 带附属的签名方案。这个版本继续支持版本 2.0 中的方案。

F 参考文档

[1] ANSI X9F1 WORKING GROUP. *ANSI X9.44 DRAFT D2: KEY ESTABLISHMENT USING INTEGER FACTORIZATION CRYPTOGRAPHY*. WORKING DRAFT, MARCH 2002.

[2] M. BELLARE, A. DESAI, D. POINTCHEVAL AND P. ROGAWAY. RELATIONS AMONG NOTIONS OF SECURITY FOR PUBLIC-KEY ENCRYPTION SCHEMES. IN H. KRAWCZYK, EDITOR, *ADVANCES IN CRYPTOLOGY – CRYPTO '98*, VOLUME 1462 OF *LECTURE NOTES IN COMPUTER SCIENCE*, PP. 26 – 45. SPRINGER VERLAG, 1998.

[3] M. BELLARE AND P. ROGAWAY. OPTIMAL ASYMMETRIC ENCRYPTION – HOW TO ENCRYPT WITH RSA. IN A. DE SANTIS, EDITOR, *ADVANCES IN CRYPTOLOGY – EUROCRYPT '94*, VOLUME 950 OF *LECTURE NOTES IN COMPUTER SCIENCE*, PP. 92 – 111. SPRINGER VERLAG, 1995.

[4] M. BELLARE AND P. ROGAWAY. THE EXACT SECURITY OF DIGITAL SIGNATURES – HOW TO SIGN WITH RSA AND RABIN. IN U. MAURER, EDITOR, *ADVANCES IN CRYPTOLOGY – EUROCRYPT '96*, VOLUME 1070 OF *LECTURE NOTES IN COMPUTER SCIENCE*, PP. 399 – 416. SPRINGER VERLAG, 1996.

[5] M. BELLARE AND P. ROGAWAY. *PSS: PROVABLY SECURE ENCODING METHOD FOR DIGITAL SIGNATURES.* SUBMISSION TO IEEE P1363 WORKING GROUP, AUGUST 1998. AVAILABLE FROM HTTP://GROUPER.IEEE.ORG/GROUPS/1363/.

[6] D. Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In H. Krawczyk, editor, *Advances in Cryptology – Crypto '98*, volume 1462 of *Lecture Notes in Computer Science*, pp. 1 – 12. Springer Verlag, 1998.

[7] D. Bleichenbacher, B. Kaliski and J. Staddon. *Recent Results on PKCS #1: RSA Encryption Standard.* RSA Laboratories' Bulletin No. 7, June 1998.

[8] B. den Boer and A. Bosselaers. An Attack on the Last Two Rounds of MD4. In J. Feigenbaum, editor, *Advances in Cryptology – Crypto '91*, volume 576 of *Lecture Notes in Computer Science*, pp. 194 – 203. Springer Verlag, 1992.

[9] B. DEN BOER AND A. BOSSELAERS. COLLISIONS FOR THE COMPRESSION FUNCTION OF MD5. IN T. HELLESETH, EDITOR, *ADVANCES IN CRYPTOLOGY – EUROCRYPT '93*, VOLUME 765 OF *LECTURE NOTES IN COMPUTER SCIENCE*, PP. 293 – 304. SPRINGER VERLAG, 1994.

[10] D. COPPERSMITH, M. FRANKLIN, J. PATARIN AND M. REITER. LOW-EXPONENT RSA WITH RELATED MESSAGES. IN U. MAURER, EDITOR, *ADVANCES IN CRYPTOLOGY – EUROCRYPT '96*, VOLUME 1070 OF *LECTURE NOTES IN COMPUTER SCIENCE*, PP. 1 – 9. SPRINGER VERLAG, 1996.

[11] D. COPPERSMITH, S. HALEVI AND C. JUTLA. *ISO 9796-1 AND THE NEW FORGERY STRATEGY.* PRESENTED AT THE RUMP SESSION OF CRYPTO '99, AUGUST 1999.

[12] J.-S. CORON. ON THE EXACT SECURITY OF FULL DOMAIN HASHING. IN M. BELLARE, EDITOR, *ADVANCES IN CRYPTOLOGY – CRYPTO 2000*, VOLUME 1880 OF *LECTURE NOTES IN COMPUTER SCIENCE*, PP. 229 – 235. SPRINGER VERLAG, 2000.

[13] J.-S. CORON. OPTIMAL SECURITY PROOFS FOR PSS AND OTHER SIGNATURE SCHEMES. IN L. KNUDSEN, EDITOR, *ADVANCES IN CRYPTOLOGY – EUROCRYPT 2002*, VOLUME 2332 OF *LECTURE NOTES IN COMPUTER SCIENCE*, PP. 272 – 287. SPRINGER VERLAG, 2002.

[14] J.-S. CORON, M. JOYE, D. NACCACHE AND P. PAILLIER. NEW ATTACKS ON PKCS #1 V1.5 ENCRYPTION. IN B. PRENEEL, EDITOR, *ADVANCES IN CRYPTOLOGY – EUROCRYPT 2000*, VOLUME 1807 OF *LECTURE NOTES IN COMPUTER SCIENCE*, PP. 369 – 379. SPRINGER VERLAG, 2000.

[15] J.-S. Coron, D. Naccache and J. P. Stern. On the Security of RSA Padding. In M. Wiener, editor, *Advances in Cryptology – Crypto '99*, volume 1666 of *Lecture Notes in Computer Science*, pp. 1 – 18. Springer Verlag, 1999.

[16] Y. Desmedt and A.M. Odlyzko. A Chosen Text Attack on the RSA Cryptosystem and Some Discrete Logarithm Schemes. In H.C. Williams, editor, *Advances in Cryptology – Crypto '85*, volume 218 of *Lecture Notes in Computer Science*, pp. 516 – 522. Springer Verlag, 1986.

[17] T. Dierks and C. Allen. *IETF RFC 2246: The TLS Protocol Version 1.0*. January 1999.

[18] H. Dobbertin. Cryptanalysis of MD4. In D. Gollmann, editor, *Fast Software Encryption '96*, volume 1039 of *Lecture Notes in Computer Science*, pp. 55 – 72. Springer Verlag, 1996.

[19] H. Dobbertin. *Cryptanalysis of MD5 Compress*. Presented at the rump session of Eurocrypt '96, May 1996.

[20]H. Dobbertin. The First Two Rounds of MD4 are Not One-Way. In S. Vaudenay, editor, *Fast Software Encryption '98*, volume 1372 in *Lecture Notes in Computer Science*, pp. 284 – 292. Springer Verlag, 1998.

[21]E. Fujisaki, T. Okamoto, D. Pointcheval and J. Stern. RSA-OAEP is Secure under the RSA Assumption. In J. Kilian, editor, *Advances in Cryptology – Crypto 2001*, volume 2139 of *Lecture Notes in Computer Science*, pp. 260 – 274. Springer Verlag, 2001.

[22]H. Garner. The Residue Number System. *IRE Transactions on Electronic Computers*, EC-8 (6), pp. 140 – 147, June 1959.

[23] M.L. GRELL. *RE: ENCODING METHODS PSS/PSS-R.* LETTER TO IEEE P1363 WORKING GROUP, UNIVERSITY OF CALIFORNIA, JUNE 15, 1999. AVAILABLE FROM HTTP://GROUPER.IEEE.ORG/GROUPS/1363/P1363/PATENTS.HTML.

[24] J. HÅSTAD. SOLVING SIMULTANEOUS MODULAR EQUATIONS OF LOW DEGREE. *SIAM JOURNAL OF COMPUTING*, VOLUME 17, PP. 336 – 341, 1988.

[25] R. Housley. *IETF RFC 2630: Cryptographic Message Syntax.* June 1999.

[26] *IEEE Std 1363-2000: Standard Specifications for Public Key Cryptography*. IEEE, August 2000.

[27] IEEE P1363 working group. *IEEE P1363a D10: Standard Specifications for Public Key Cryptography: Additional Techniques.* November 1, 2001. Available from http://grouper.ieee.org/groups/1363/.

[28] *ISO/IEC 9594-8:1997: Information technology – Open Systems Interconnection – The Directory: Authentication Framework*. 1997.

[29] *ISO/IEC FDIS 9796-2: Information Technology – Security Techniques – Digital Signature Schemes Giving Message Recovery – Part 2: Integer Factorization Based Mechanisms.* Final Draft International Standard, December 2001.

[30]*ISO/IEC 18033-2: Information Technology – Security Techniques – Encryption Algorithms – Part 2: Asymmetric Ciphers*. V. Shoup, editor, Text for 2nd Working Draft, January 2002.

[31] J. Jonsson. Security Proof for the RSA-PSS Signature Scheme (extended abstract). *Second Open NESSIE Workshop.* September 2001. Full version available from http://eprint.iacr.org/2001/053/.

[32] J. Jonsson and B. Kaliski. On the Security of RSA Encryption in TLS. In *Advances in Cryptology – Crypto 2002*, to appear.

[33] B. Kaliski. *IETF RFC 1319: The MD2 Message-Digest Algorithm*. April 1992.

[34] B. Kaliski. On Hash Function Identification in Signature Schemes. In B. Preneel, editor, *RSA Conference 2002, Cryptographers' Track*, volume 2271 of *Lecture Notes in Computer Science*, pp. 1 – 16. Springer Verlag, 2002.

[35] B. Kaliski and J. Staddon. *IETF RFC 2437: PKCS #1: RSA Cryptography Specifications Version 2.0.* October 1998.

[36] J. Manger. A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0. In J. Kilian, editor, *Advances in Cryptology – Crypto 2001*, volume 2139 of *Lecture Notes in Computer Science*, pp. 260 – 274. Springer Verlag, 2001.

[37] A. Menezes, P. van Oorschot and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[38] National Institute of Standards and Technology (NIST). *FIPS Publication 180-1: Secure Hash Standard*. April 1994.

[39] National Institute of Standards and Technology (NIST). *Draft FIPS 180-2: Secure Hash Standard.* Draft, May 2001. Available from http://www.nist.gov/sha/.

[40] J.-J. Quisquater and C. Couvreur. Fast Decipherment Algorithm for RSA Public-Key Cryptosystem. *Electronics Letters*, 18 (21), pp. 905 – 907, October 1982

[41] R. Rivest. *IETF RFC 1321: The MD5 Message-Digest Algorithm*. April 1992.

[42] R. Rivest, A. Shamir and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21 (2), pp. 120-126, February 1978.

[43] N. Rogier and P. Chauvaud. The Compression Function of MD2 is not Collision Free. Presented at *Selected Areas of Cryptography '95*. Carleton University, Ottawa, Canada. May 1995.

[44] RSA Laboratories. *PKCS #1 v2.0: RSA Encryption Standard*. October 1998.

[45] RSA Laboratories. *PKCS #7 v1.5: Cryptographic Message Syntax Standard*. November 1993.

[46] RSA Laboratories. *PKCS #8 v1.2: Private-Key Information Syntax Standard*. November 1993.

[47] RSA Laboratories. *PKCS #12 v1.0: Personal Information Exchange Syntax Standard*. June 1999.

[48]V. Shoup. OAEP Reconsidered. In J. Kilian, editor, *Advances in Cryptology – Crypto 2001*, volume 2139 of *Lecture Notes in Computer Science*, pp. 239 – 259. Springer Verlag, 2001.

[49]R. D. Silverman. *A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths.* RSA Laboratories Bulletin No. 13, April 2000. Available from http://www.rsasecurity.com.rsalabs/bulletins/.

[50]G. J. Simmons. Subliminal communication is easy using the DSA. In T. Helleseth, editor, *Advances in Cryptology – Eurocrypt '93*, volume 765 of *Lecture Notes in Computer Science*, pp. 218-232. Springer-Verlag, 1993.