基于单向陷门置换的长消息签密方案*

胡振宇** 林东岱 吴文玲 冯登国

(中国科学院软件研究所信息安全国家重点实验室, 北京 100080; 中国科学院研究生院, 北京 100039)

摘要 在随机 Oracle 模型的基础上, 提出一种基于单向陷门置换(trapdoor permutations, TDPs)的、可并行的、长消息签密方案——PLSC (parallel long-message signcryption). 该方法采用“整体搅乱, 局部加密(scramble all, and encrypt small)”的思想, 用一个伪随机数对要传送的消息和用户的身份(ID)进行“搅乱(scrambling operation)”, 然后对两个固定长度的小片段(并行地)进行单向陷门置换(TDP)操作. 这种设计使得整个方案可直接高效地处理任意长度的消息, 既可避免循环调用单向陷门置换(如 CBC 模式)所造成的计算资源的极度消耗, 也可避免由“对称加密方案”与“签密方案”进行“黑盒混合(black-box hybrid)”所造成的填充(padding)冗余. 不仅可以显著地节约消息带宽, 而且可以显著地提高整体效率. 具体地说, 该方法对任何长度的消息进行签密, 仅需进行一次接收方的 TDP 运算(相当于加密), 以及一次发送方的 TDP 运算(相当于签名), 从而最大限度地降低了 TDP 运算的次数, 提高了整体的运算效率. 因为, 对于公钥加密算法来说, 运算量主要集中在 TDP 运算上, TDP 运算是整个算法的瓶颈所在. 另一方面, 由于避免了填充上的冗余, 新方案的效率也高于标准的“黑盒混合”方案.

重要的是, 新方案能够达到选择密文攻击下的紧致的语义安全性(IND-CCA2)、密文完整性(INT-CTXT)以及不可否认性(non-repudiation). 而且所有这些安全要求都可以在多用户(multi-user)、内部安全(insider-security)的环境下得以实现.

另外, 尽管新方案主要针对长消息的签密, 但它也可应用于某些不能进行大块数据处理的环境(智能卡或其他只有少量内存的环境). 也就是说, 对于这些小内存设备来说, 仍然可以用该方案来实现长消息的签密处理.

关键词 认证加密 签密 单向陷门置换 并行

收稿日期: 2005-12-28; 接受日期: 2006-05-16

* 国家“973”计划(2004CB318004)和自然科学基金(批准号: 60373047, 90604036)资助项目

** E-mail: zhenyu@iscas.cn

1 引言

1.1 背景知识

一直以来, 保密性和完整性都被看成是两个完全独立的安全目标, 前者用来保证受到保护的信息不能让除了目的接收方外的第三方知道, 后者则用来保证受到保护的信息在传递的途中不被篡改. 然而, 在相当多的应用中, 如安全电子邮件, 要求保密和认证二者要同时得到满足. 为了实现这一安全目标, 一种可能的方法就是将一个加密方案和一个签名方案结合起来, 形成一个认证加密方案. 然而, 简单地将一个加密方案和一个签名方案结合起来却未必能达到理想的保密性或完整性要求[1]. 因此, 探讨如何设计一个简洁高效的方案来同时实现保密性和完整性要求就显得很有必要.

签密(signcryption)的概念最早由Zheng在 1997 年提出[2]. 其明确的目标是设计一种签密方案, 其运行代价比将一个签名方案和一个加密方案按顺序的方式结合的总代价要小. 在此之后, 已有数篇文章[3~5]对签密方案的安全性进行了研究, 但也只是到近来, Baek 等人[6]和An等人[1]才首次对这一主题进行了正式的研究. 在二者的文献中, 签密方案都被定义为一个多用户的操作原语, 它可同时获得CCA安全的保密性及INT-PTXT安全的完整性. Baek等人在文献[6]中证明了基于“离散对数”的、最初由Zheng提出的签密方案[2], 在Gap Diffie-Hellman假设下的随机Oracle模型中是安全的. An等人[1]则说明了将一个加密方案与一个认证方案进行合成以获得签密方案的方法. 这种合成的方法使得用户可以方便地选择自己偏爱的方案与其他用户进行通信. 虽然Zheng提出的签密方案[2]在离散对数的环境下, 是一个相当优秀的方案, 并可能获得很好的运行效率, 但所有的用户都必须采用统一的公开参数(如公共的离散对数群). 这就要求对于公开参数或签密方案的任何修改都必须获得全体用户的一致同意才行. 但另一方面, 尽管文献[1]中的通用方法可以将一个签名方案和一个加密方案结合起来获得签密方案, 但这样得到的方案在运行效率和安全性上都不能尽如人意.

实际上, 许多切有实效的签名方案和加密方案(如OAEP[7], OAEP+[8], PSS-R[9])都是在单向陷门置换(TDP)的基础上构造而成. 它们的安全性分析也都是在随机Oracle(RO)模型中进行. 然而, 如果将签名方案和加密方案结合在一起时, 即使是这些高效的实现方案也未必能获得高效的或安全的签密方案[1]. 当使用认证和加密来合成时, 用户必须分别维护一对签名密钥和一对加密密钥; 由于可能重复的消息填充(padding)及为防止身份伪造而进行的额外操作, 可使最终的签密方案的消息带宽达不到最优的效果; 而签密方案的安全性也可能不是那么紧致(tight). 况且, 多数情况下, 这些基于单向陷门置换的公钥方案能够处理的消息长度不能超过其密钥的长度. 如果要加密大块数据, 如一个文件, 就必须不

断地重复调用这些方案(如CBC或CTR模式), 这将会导致由于重复进行TDP运算而造成极度的时间消耗. 另一个常用的加密大块数据的方法是采用KEM-DEM的混合技术, 将一个密钥封装机制(key encapsulation mechanism, KEM)与一个数据封装机制(data encapsulation mechanism, DEM)结合起来[10,11]. 不管是上面的重复调用技术, 还是KEM-DEM的混合技术, 都是将基本的加密方案(公钥方案或对称方案)看成是一个独立的“黑盒(black-box)”, 从而不可避免地造成对数据的重复填充. 考虑到单向陷门置换及消息填充技术在各种实际方案中的应用, 本文采用“整体搅乱, 局部加密”的思想, 提出一种利用单向陷门置换构建签密方案的方法, 使之既解决由“黑盒”合成方法造成的效率问题, 又具有简单、高效、近乎最佳的安全性等优点. 此外, 还能够直接处理任意长度的消息.

1.2 相关工作介绍

我们将本文的结果与其他一些相关[1,12,13]的工作作一比较.

我们先考查文献[1]提出的“commit-then-encrypt-and-sign ($\mathcal{CtE\&S}$)”. 在该方案中, 如果要对消息m进行签密, 首先将原始消息与用户的身份(identitie)合在一起, 形成消息m'. 然后对消息m'进行承诺操作, 将其变成两部分(d, c), 对于其中的一部分d进行加密, 对另一部分c进行签名. 在进行加密和签名时, 因为是进行两个完全独立操作, 通常要再分别进行两次独立的填充操作(如RSA-OAEP[7,8], PSS-R[9]), 将(d, c)变成(w, s), 之后才分别对w和s分别进行两个独立的单向陷门置换操作, 以获得最终的密文. 结果, 对一个消息进行处理需要进行 4 次填充操作(1 次关于密钥, 1 次用于承诺, 2 次用于单向陷门置换), 7 次Hash运算(1 次关于密钥, 2 次用于承诺, 2 次用于加密, 2 次用于签名). 其实, 就目前现状来看, 基于单向陷门置换的加密方法要么难以达到精确的安全要求(exact security)[8], 要么必须将消息填充得超过一次单向陷门置换可处理的长度[14]($\mathcal{CtE\&S}$既不能达到INT-CTXT (integrity of ciphertext)安全, 也不能达到IND-CCA (indistinguishability under adaptive chosen-ciphertext attacks)安全, 但可以达到比此弱一点的安全要求, 具体细节可参见文献[1]). 即便如此, 承诺后的两个片段依然被要求不能超过相应的(加密或签名)密钥长度. 在本文的方案中, 我们仅需对消息进行一次填充(如果我们将其看成是填充的话), 两次Hash运算. 该方案在内部安全的环境中可以达到INT-CTXT 安全和 IND-CCA2 安全, 并且可以处理任意长度的消息.

Dodis等人在文献[12]中提出了一个用于并行加密和签名操作的填充方案 ——PSEP (probabilistic signature and encryption padding). 该方案要求对原始消息先进行承诺操作和Feistel变换[15], 然后再对得到的两个片段分别实施单向陷门置换操作. 主要的处理思路是: 如果要对消息m进行签密, 先对消息m进行下面的承诺变换, 将原始消息拆分成两子消息(d, c):

$$d = m_2 \| r,$$

$$c = (m_1 \oplus \mathcal{H}(r)) \| \mathcal{H}'(m_2 \| r).$$

然后对(d, c)实施一个 Feistel 变换得到填充后的两个片段(w, s), 再分别对 w 和 s 分别实施单向陷门置换操作(分别代表签名和加密), 得到最终密文.

尽管作者声称, 对于长度为k的密钥, PSEP 可以处理的消息长度大致为 $2k$, 但实际上是不可能的. 因为, 如果要使得值 $2^{-(k-|m_i|)}$ (i=1,2)成为一个对于猜测攻击来说是可忽略的量, 就必须有$k - |m_i| \geqslant 160$, 这就意味着$|m| = |m_1| + |m_2| \leqslant 2(k - 160)$. 在本文的方案中, 消息的长度不受限制, 而消息带宽及处理效率随着消息长度的增加而提高. 如果我们采用的密钥长度设为 1024, 则当消息的长度为 1024 时, 其带宽为 30%, 但当消息的长度增加到密钥长度的 2 倍时, 带宽为 50%, 如果将消息的长度增加到密钥长度的 8 倍, 则带宽可以提高到 80%. 处理消息的长度越长, 则带宽就越高, 但总的计算代价却差不多保持不变. 其主要原因是Hash运算可以非常快速地实现, 主要的时间消耗都集中在两个单向陷门置换的计算上了.

为了处理长消息, 文献**[12]**的作者也提出用“黑盒混合(black-box hybrid)”的方法将一个短消息签密方案扩展成可以处理任意长度消息的签密方案. 对于OTP(one time padding)方案及作者提出的 PSEP 填充方法, 当对一个消息进行签密时, 需要进行 4 次 Hash 运算(1 次用于 OTP 操作, 2 次用于承诺操作, 1 次用于 Feistel 变换). 与本文提出的长消息签密方案相比, 该方案多花费 2 个 Hash 运算, 但在带宽上却没有任何提高.

我们再将本文的方案与RSA-TBOS[13]进行比较. RSA-TBOS是使用PSS-R填充技术, 将RSA签名方案与RSA加密方案按顺序的方式结合而成的签密方案. 其对消息m的签密可用函数$\mathrm{RSA}_R(\mathrm{RSA}^{-1}{}_S\ (w\|s))$来表示, 其中$w\|s$是PSS-R填充方案对消息$m$的处理结果, RSA_U和$\mathrm{RSA}^{-1}{}_U$分别是在用户U的密钥下的RSA函数及逆RSA函数. 然而, PSS-R并不像人们期望的那样是一个优秀的填充方案, 即使是采用 2048 bit的RSA模数, 也不能保证据此获得的方案能达到实用的安全要求. 此外, RSA-PSS-R加密的带宽和消息恢复效率都相当低. 在典型的密钥配置($k = |N_U| = 2048$, $k_0 = 160$, N_U为用户的RSA模数)情况下, 消息的最大长度为 826 bit, 其带宽为 42%. 而本文提出的方案能够提供紧致的安全性, 可以并行处理任意长度的消息, 从而获得比RSA-TBOS更好的性能.

1.3　本文的主要贡献

本文提出的签密方案也是建立在单向陷门置换(TDP)之上的. 每个用户独立地选择一个单向陷门置换f_U(及其陷门信息 f_U^{-1}), 并公开f_U作为其公钥, 保密 f_U^{-1} 作为私钥. 我们的思路是采用“整体搅乱, 局部加密(scramble all, and encrypt

small)”的思想, 用一个伪随机数对要传送的消息和用户的身份(ID)进行“搅乱(scrambling operation)”, 然后分别对其中的两个固定长度的小片段(并行地)进行单向陷门置换操作. 在整个操作过程中, 我们仅进行一次“搅乱”操作, 而不是分别对每个单向陷门置换各进行一次“搅乱”操作. 这种设计使得整个方案可直接高效地处理任意长度的消息, 既可避免循环调用单向陷门置换(如CBC模式)所造成的计算资源的极度消耗, 也可避免由“对称加密方案”与“签密方案”进行“黑盒混合”所造成的填充冗余. 不仅可以显著地节约消息带宽, 而且可以显著地提高整体效率. 我们并没有对单向陷门置换的计算进行改进, 而是设法减少对单向陷门置换的调用次数. 因为在整个方案的处理过程中, 单向陷门置换及其逆置换所花的时间占着绝对统治的地位. 事实上, 之所以KEM-DEM的混合技术会被广泛采用, 就是因为可以借助对称方案来减少单向陷门置换及其逆置换被调用的次数. 具体地说, 当对任何一个消息进行签密时, 本文提出的签密方案仅需进行一次接收方的TDP运算(相当于加密), 以及一次发送方的TDP运算(相当于签名), 从而最大限度地降低了TDP运算的次数, 提高了整体的运算效率. 另一方面, 由于避免了重复填充带来的冗余, 我们新方案的效率也高于广泛应用的所谓“黑盒混合”的KEM-DEM方案.

本文提出的签密方案来源于Dodis等人[12,16]提出的基于填充技术的并行签密方案. 与直接将一个签名方案和一个加密方案进行合成相比, 虽然基于填充技术的并行签密方案的主要优势在于其计算上的可并行性, 但其在安全性、灵活性以及相容性方面的优点, 也更增加了其在应用上的优势(关于两种方式的详细比较可参阅文献[16]). 值得一提的是, 我们的新方案能够达到选择密文攻击下的紧致的语义安全性(IND-CCA2)、密文完整性(INT-CTXT)以及不可否认性(non-repudiation). 而且所有这些安全要求都可以在多用户(multi-user)、内部安全(insider-security)的环境下得以实现.

为了评价新方案的性能, 我们对不同长度的消息作一实验比较. 我们利用Crypto++4.2 来实现我们的算法, 通过少量的修改, 使其包含两个 RSA 操作和一个“搅乱”操作. 为了便于进行对比, 我们采用与 Crypto++4.2 中的 RSA-OAEP 方案相同的 RSA 操作和 Hash 操作, 并采用相同的密钥长度. 结果表明, 在进行大块数据签密时, 本文的新方案表现出良好的性能.

2 预备知识

2.1 记号

本文采用符号“$|x|$”表示串x的比特长度, “$x||y$”表示x与y的连接, 符号“$\oplus$”表示“异或”操作. 如果 n 是一个正整数, 则符号$\{0,1\}^n$ 表示n 比特长的二进制串集合(也用符号$\{0,1\}^*$ 来表示长度不确定的二进制串集合). 如果函数f是一个随

机(确定性)函数, 则符号“$x \xleftarrow{R} f(y)$”(“$x \leftarrow f(y)$”)表示运行f得到y并将y赋值给x的操作. 如果$\boldsymbol{M}$是一个集合, 则“$x \xleftarrow{R} \boldsymbol{M}$”表示从$\boldsymbol{M}$中随机选取一个$x$.

2.2　单向陷门置换

一个单向陷门置换(TDP)函数簇是一个具有陷门信息的单向置换簇. 这样的函数簇中的每个函数f, 都具有一个只被拥有者所知的秘密陷门信息f^{-1}. 该陷门信息可以使函数的拥有者在值域中每一点对函数f求逆. 对于一个单向陷门置换函数簇来说, 应该比较容易地随机产生一个单向陷门置换f及其陷门信息f^{-1}. 对其定义域中的每一点x, $f(x)$可以很容易计算出来, 并且, 如果知道其对应的陷门信息, 其值域中的每一点y也可以很容易地计算其逆$f^{-1}(y)$. 但是, 如果不知道该陷门信息, 要计算值域中的任何一点的逆都是困难的. 也就是说, 对于任何敌手$\mathcal{A}$和一个函数值$y = f(x)$, 找到正确的x的概率对于安全参数k来说是可以忽略的. 我们说单向陷门置换函数簇F是$\varepsilon_{\mathrm{TDP}}$安全的, 如果对于其中的任何一个函数$f_i$和任意的 $f_i(x)$, 不存在任何PPT (probabilistic polynomial time)敌手$\mathcal{A}$, 在不知道陷门信息的情况找到$f_i(x)$的原像x的概率会超过$\varepsilon_{\mathrm{TDP}}$.

2.3　签密方案及其安全性

我们来描述多用户条件下的签密方案及其安全性. 我们将文献[1]和[17]的概念拓展到包括双方用户的身份(公钥), 以便使签密方案有更广泛的应用.

2.3.1　签密方案的语法结构

一个基于公钥的认证加密(签密)方案$\mathcal{SC}=(\mathcal{K}, \mathcal{SE}, \mathcal{VD})$由三个算法组成:

1. 随机化密钥生成算法$\mathcal{K}$. 当输入安全参数k时为每个参与方P生成一对密钥(pk_P, sk_P)作为配对的公私密钥, 记作$(pk_P, sk_P) \xleftarrow{R} \mathcal{K}(k)$.

2. 签密(signcryption——sing/encrypt)算法$\mathcal{SE}$ 是一个随机算法, 当输入发送方S的公私钥pk_S, sk_S及接收方R的公钥pk_R和明文$m \in \boldsymbol{M}$时, 返回一个签名密文C, 记作$C \xleftarrow{R} \mathcal{SE}_{pk_S, sk_S, pk_R}(m)$.

3. 解签密(de-signcryption——verify/decrypt)算法$\mathcal{VD}$是一个确定性算法, 当输入发送方S的公钥pk_S, 接收方R的公私钥pk_R, sk_R和密文C时返回明文m, 或符号“$\perp$”以示非法. 记作$m \leftarrow \mathcal{VD}_{pk_S, pk_R, sk_R}(C)$, 其中$m \in \boldsymbol{M} \cup \{\perp\}$. 我们要求对所有的$m \in \boldsymbol{M}$, $\mathcal{VD}_{pk_S, pk_R, sk_R}(\mathcal{SE}_{pk_S, sk_S, pk_R}(m)) = m$均成立.

以上定义与An[1,17]给出的两个定义的主要区别在于, 这里我们在签密和解签密运算时将通信双方的公钥同时参与运算, 从而可将传送的签密消息与双方的身份进行绑定. 这样做的原因首先是因为, 通信双方的公钥是公开的, 任何人(包括敌手)都可以获取. 其次也更有利于形成安全的签密方案. 例如, 在先加密后签

名的方案中, 一个中间人攻击者可以截获一个签密密文, 然后用自己的私钥重新签名, 并声称该消息是自己产生的, 然后再将新的签密密文传送到目的接收方. 这样, 接收方就会相信消息是真的, 是攻击者产生的. 如果我们在加密时将发送方的公钥一起参与运算, 则中间人攻击者用以上方法篡改后, 目的接收方将不会得到合法的明文消息. 同样道理, 将接收方的公钥参与解签密运算也是为了将接收方的身份与消息进行绑定, 以防止接收方篡改自己的公私钥对来诬陷发送方. 例如, 在先加密后签名的方案中, 接收方收到签密密文后, 可能重新选择自己的公私钥对, 使得加密密文在新私钥下解密得到不同的明文, 以此来诬陷发送方. 如果我们在签密时将接收方的公钥参与运算, 则当接收方重新选择自己的公私钥对后, 将不能通过合法性检验, 从而阻止接收方的这种诬陷.

2.3.2 签密方案的安全性

本文我们仅考虑公钥认证加密方案的最强的多用户条件下的内部安全性[1]. 内部安全性是针对通讯的参与方来说, 假设攻击者来自通讯合法的参与方. 涉及的安全性也是两个方面——不可伪造性和保密性. 不可伪造性主要指发送方发送数据后不可抵赖, 并且接收方不能篡改数据以诬陷发送方. 保密安全性则主要针对发送方而言的, 如果不是发送方自己签密的消息, 他无法知道其中的有用的信息. 假若在某种情况下, 一个敌手窃取了发送方的密钥, 这时候, 他不应该了解到以前曾经签密的消息内容. 内部安全性实际上是forward security. 此时, 由于敌手是合法的内部人员, 掌握了接收方或发送方的私钥(但不能二者同时掌握), 签密方案相对于敌手来说只不过是个单纯的签名方案或单纯的公钥加密方案. 因此, 我们可用特殊的加密方案和签名方案来分别界定签密方案的内部安全性. 我们将这样的签名方案和(公钥)加密方案, 称为由SC导出的签名方案和加密方案.

定义 1(由签密方案导出的签名方案) 设$SC = (\mathcal{K}, \mathcal{SE}, \mathcal{VD})$是签密方案, (sk_R, pk_R)和(sk_S, pk_S)分别是通信双方的公私钥对. 设$pub = (pk_S, pk_R)$. 当SC用作内部签名方案时, 签名密钥为$SK=(sk_S, pub)$, 验证密钥为$VK = (sk_R, pub)$. 当要对消息m进行签名时, 计算$C=\mathcal{SE}_{SK}(m)$, 当要进行验证时, 计算$m=\mathcal{VD}_{VK}(C)$, 当且仅当$m \neq \perp$时, 敌手成功. 具体描述如下:

签名算法 $S_{sk_S, pk_S, pk_R}(m)$	验证算法 $\mathcal{V}_{sk_R, pk_S, pk_R}(s)$
$pub=(pk_S, pk_R)$, $SK=(sk_S, pub)$	$pub=(pk_S, pk_R)$, $VK=(sk_R, pub)$
$s \leftarrow \mathcal{SE}_{SK}(m)$	$v \leftarrow \mathcal{VD}_{VK}(s)$
返回 s	if $v \neq \perp$ 则返回 1 else 返回 0

当潜在的敌手是消息的接收方时, 签密方案的安全性可用该签名方案的安全性来度量. 根据敌手的能力, 其安全性可分为 INT-PTXT (integrity of plaintext)或 INT-CTXT (integrity of ciphertext) 安全.

定义 2 (由签密方案导出的加密方案) 当SC用作内部加密方案时, 加密密钥为$EK = (sk_S, pub)$, 解密密钥为$DK = (sk_R, pub)$. 当要对消息m进行加密时, 计算$C = \mathcal{SE}_{EK}(m)$, 当要进行解密时, 计算$m = \mathcal{VD}_{DK}(C)$. 具体描述如下:

加密算法 $\mathcal{E}_{sk_S, pk_S, pk_R}(m)$	解密算法 $\mathcal{D}_{sk_R, pk_S, pk_R}(c)$
$pub = (pk_S, pk_R), EK = (sk_S, pub)$	$pub = (pk_S, pk_R), DK = (sk_R, pub)$
$c \leftarrow \mathcal{SE}_{EK}(m)$	$m \leftarrow \mathcal{VD}_{DK}(c)$
返回 c	返回 m

当潜在的敌手是消息的发送方时, 签密方案的安全性可用该加密方案的安全性来度量. 根据敌手的能力, 其安全性可分为 IND-CPA (indistinguishability under chosen-plaintext attacks)或 IND-CCA2 (indistinguishability under adaptive chosen-ciphertext attacks)安全.

外部敌手和内部敌手的区别在于, 内部敌手是消息的合法接收者, 他可以创建自己的公私钥对, 并在以后需要时进行更换. 而外部敌手则不具备这种能力. 对于一个合法的签密密文, 如果改变了接收者的公私钥对, 则以接收方新的公私钥对进行解密验证时极有可能出现新的明文结果, 从而导致一个成功的伪造明文消息.

满足以上的内部安全性的签密方案具有很强的安全性, 可以广泛地应用于各种环境中. 正如文献[1]中分析的那样, 将一个加密方案与一个签名方案直接复合起来, 未必能获得这样强的内部安全性. 特别是, 任何由PKCS#1[18]中的加密方案与签名方案直接复合而成的签密方案都不能达到这样的安全要求(即使按文献[1]的做法, 先进行承诺, 然后再分别加密和签名). 本文中提出的签密方案先将消息与用户ID进行“搅乱”, 然后再对其中的两个小片段进行并行TDP操作, 可以达到以上的内部安全要求.

3　基于单向陷门置换的长消息签密方案及其安全性

我们先介绍新方案的一般结构. 为描述方便, 我们将新方案命名为“并行长消息签密方案(parallel long-message signcryption, PLSC)”. PLSC方案的密钥生成算法可以为每个用户U生成一对 $(t, \varepsilon_{\mathrm{TDP}})$ 安全的TDP (f_U, f_U^{-1}) (可利用RSA, Rabin函数等著名的TDP). 从内部安全的角度来看, 消息的发送方S和消息的接收方R都可被看作是潜在的敌手. PLSC方案使用两个密码Hash函数, 将要处理的明文消息、一个与TDP分组等长的随机串, 还有为防止身份伪造而加入的双方的公

钥作为输入, 其工作过程可形象地用图 1 来表示.

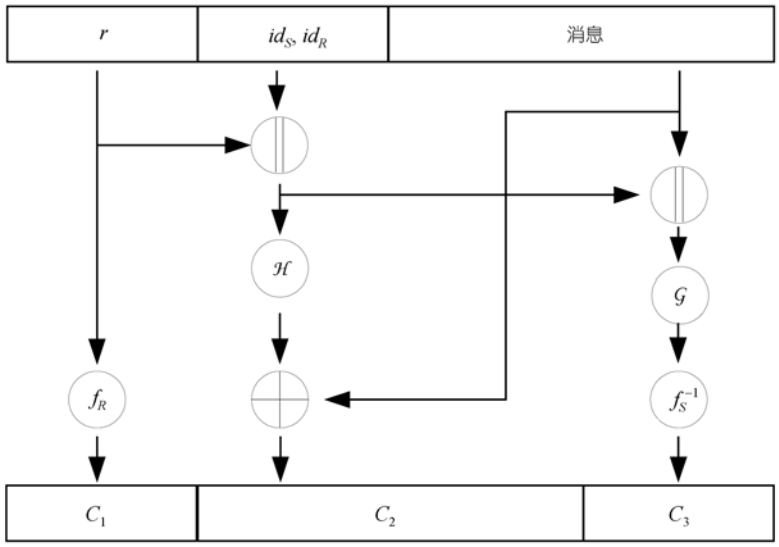

图 1 PLSC 的构造

定义 3 (PLSC的结构) 设n和l分别是两个正整数, $\mathcal{H}$: $\{0,1\}^* \to \{0,1\}^n$和$\mathcal{G}$: $\{0,1\}^* \to \{0,1\}^l$是两个密码Hash函数, F: $\{0,1\}^l \to \{0,1\}^l$是一个单向陷门置换函数簇. $\mathcal{ID}$是一个公开的确定性算法, 以将一个用户的公钥转换成相应的串(可简单使用Hash函数或恒等变换). 使用密码Hash函数$\mathcal{H}$, $\mathcal{G}$及单向陷门置换函数簇F的签密方案$\mathcal{SC}$=($\mathcal{K}$, $\mathcal{SE}$,$\mathcal{VD}$)的结构如下:

算法 $\mathcal{K}(k)$	算法 $\mathcal{SE}_{f_S, f_S^{-1}, f_R}(m)$:	算法 $\mathcal{VD}_{f_S, f_R^{-1}, f_R}(c)$:
$f_R \xleftarrow{R} F$ $f_S \xleftarrow{R} F$ 返回$\langle f_R, f_S^{-1}\rangle, \langle f_R, f_R^{-1}\rangle$	$id_S \leftarrow \mathcal{ID}(f_S)$ $id_R \leftarrow \mathcal{ID}(f_R)$ $r \xleftarrow{R} \{0,1\}^l$ $p \leftarrow \mathcal{H}(r, id_S\|\|id_R)$ $g \leftarrow \mathcal{G}(m\|\|r, id_S\|\|id_R)$ $\sigma \leftarrow f_S^{-1}(g)$ $c_1 \leftarrow f_R(r)$ $c_2 \leftarrow m \oplus p$ $c \leftarrow c_1\|\| c_2 \|\|\sigma$ 返回 c	$id_S \leftarrow \mathcal{ID}(f_S)$ $id_R \leftarrow \mathcal{ID}(f_R)$ Parse c as $c_1\|\| c_2 \|\|\sigma$ $r \leftarrow f_R^{-1}(c_1)$ $p \leftarrow \mathcal{H}(r, id_S\|\|id_R)$ $m \leftarrow c_2 \oplus p$ $g \leftarrow \mathcal{G}(m\|\|r, id_S\|\|id_R)$ $g' \leftarrow f_S(\sigma)$ If $g = g'$ then 返回 m Else 返回 $\perp$

$\mathcal{G}(\cdot)$可以是任何具有固定输出长度的“抗碰撞”Hash函数(collision resistant Hash function, CRHF), 只要它可以作为PKCS#1 中定义的随机Oracle模型, 输出长度与TDP的逆的输入长度相匹配即可(如SHA256). 函数$\mathcal{H}(\cdot)$可以采用“掩码生成函数(mask generating function, MGF)”, 将一个固定长度的CRHF输出转换成变长的CRHF输出. 使用CRHF来构造MGF的方法可参阅PKCS#1 v2.1 提供的文本[18].

定理 1 (PLSC的安全性)　设n和l分别是两个正整数, $\mathcal{H}$: $\{0,1\}^*\to\{0,1\}^n$和$\mathcal{G}$: $\{0,1\}^*\to\{0,1\}^l$是两个密码Hash 函数, F: $\{0,1\}^l\to\{0,1\}^l$是一个安全性为$\varepsilon_{\mathrm{TDP}}$的单向陷门置换函数簇. 设$\mathcal{SC}=(\mathcal{K}, \mathcal{SE}, \mathcal{VD})$ 是按照定义 3 的方法构造的签密方案. 对于任何攻击$\mathcal{SC}$的IND-CCA2 安全性的PPT 敌手$\mathcal{A}$, 或一个攻击$\mathcal{SC}$的INT-CTXT安全性的PPT敌手$\mathcal{B}$, 访问随机Oracle $\mathcal{H}$和$\mathcal{G}$、签密Oracle $\mathcal{SE}$及解签密Oracle $\mathcal{VD}$的次数最多分别为q_H, q_G, q_S及q_D, 花费的时间为t, 有:

$$Adv^{ind-cca2}_{\mathcal{SC},\mathcal{A}}(t,q_H,q_G,q_S,q_D)\leqslant q_D\cdot 2^{-n+1}+(q_D+q_S\cdot((2^{-n}+1)q_S+q_H+2^{-n}\cdot q_G))\cdot 2^{-l+1}+4\cdot\varepsilon_{\mathrm{TDP}}, \tag{1}$$

$$Adv^{int-ctxt}_{\mathcal{SC},\mathcal{B}}(t,q_H,q_G,q_S,q_D)\leqslant(q_D+1)\cdot 2^{-n}+(q_D+q_S\cdot((2^{-n}+1)q_S+q_H+2^{-n}\cdot q_G)+1)\cdot 2^{-l}+\varepsilon_{\mathrm{TDP}}, \tag{2}$$

$$t=O((q_G+2\cdot q_S+q_D)T_f). \tag{3}$$

其中, T_f 表示进行一次TDP操作所花的时间.

定理 1 的证明放在附录 A 里.

4　PLSC 的性能分析

我们将PLSC与其他方案进行性能上的对比. 为了统一比较的基础, 我们设PLSC方案中的单向陷门置换为RSA函数. 根据RSA假设, RSA是一个强单向陷门函数簇[19]. 因此基于RSA的PLSC方案仍然是内部IND-CCA2 安全和INT-CTXT安全的.

先分析PLSC中各组件所需的代价. 因为Hash函数可以快速地实现, 整个方案的时间代价主要集中在RSA的模指数运算上. 设T_{exp}为RSA的模指数运算所花的时间, T_h为Hash运算所花的时间, T_r为从$\{0,1\}^l$中选择随机串所花的时间. 再设$T_{\text{se-PLSC}}$和$T_{\text{vd-PLSC}}$分别表示签密和解签密所花的总代价, 则

$$T_{\text{se-PLSC}}=T_r+2T_h+2T_{\mathrm{exp}},\ T_{\text{vd-PLSC}}=2T_h+2T_{\mathrm{exp}}.$$

我们再来分析文献[13]中的基于PSS-R填充技术的RSA-TBOS方案的性能. 在填充技术中, 为了填充后的消息(看作是一个整数)不大于N_A, 要采用一个“试错(trial-and-error)”方法进行检验. 设$T_{\text{se-rsa-tbos}}$和$T_{\text{vd-rsa-tbos}}$分别表示RSA-TBOS方案进行签密和解签密所花的总代价. 如果采用PSS-R填充中使用的“0 先导

(leading-0)”技术, 则$T_{\text{se-rsa-tbos}} = T_r + 2T_h + 2T_{\exp}$与PLSC 的代价相同.

但是在签密时, 如果遇到$\sigma = Padding(m, r)^{d_A} \pmod{N_A} > N_B$, 为了能够产生正确的密文, 必须将其最高位砍掉(be chopped off), 然后再由解签密时进行恢复. 这种情况出现的概率大致为 1/2. 因此, 为了产生正确的明文, 在解签密时将有 1/2 的概率会重复计算Hash和RSA指数幂, 其平均次数应该为 1.5. 于是

$$T_{\text{vd-rsa-tbos}} = T_{\exp} + 1.5(2 \cdot T_h + T_{\exp}) = 3T_h + 2.5T_{\exp}.$$

考虑到$T_{\exp}$在整个时间代价中占据着绝对统治的地位, 以上的代价远比PLSC中相应的代价要高.

再来看看RAS-TBOS的通讯代价(带宽). RAS-TBOS中采用的RSA-PSS-R填充技术要求明文消息的长度不能超过RSA的模的长度的一半, 其消息带宽相当低. 在典型的密钥配置($k = |N_U| = 2048$, $k_0 = 160$, N_U是RSA 模数)情况下, 消息的最大长度为 826 bit, 其带宽为 42%. 而对于PLSC方案, 如果采用与RAS- TBOS相同的RSA模数N_U, 当$n = 2 \cdot l$, $|r| = l$, $|p| = |m| = n$, $|c_2| = |p| = n$, $|c_1| = |\sigma| = l$时, 有$|m| = 0.5|c|$. 此时的消息带宽为 50%. 如果继续提高n与l的比值, 其消息恢复带宽就会更高. 如果将其比值提高到$n = 8 \cdot l$, 则PLSC的消息带宽可提高到 80%.

我们再来分析文献[12]提出的基于“黑盒混合”技术的方案Hybrid-PSEP的性能. 对于OTP(one time padding)方案及作者提出的PSEP 填充方法, 当对一个消息进行签密时, 需要进行 4 次Hash运算(1 次用于OTP 操作, 2 次用于承诺操作, 1 次用于Feistel变换). 设$T_{\text{se-hybrid-psep}}$和$T_{\text{vd-hybrid-psep}}$分别为Hybrid-PSEP方案在签密和解签密时所花的总代价. 则

$$T_{\text{se-hybrid-psep}} = T_r + 4T_h + 2T_{\exp},\ T_{\text{vd-hybrid-psep}} = 4T_h + 2T_{\exp}.$$

该方案与 PLSC 方案相比具有相同的带宽, 但多花费 2 个 Hash 运算.

PLSC 方案与 Hybrid-PSEP 和 TBOS 的性能比较可小结为表 1.

表 1 PLSC 方案与 Hybrid-PSEP 和 TBOS 的性能比较

Cost	Hybrid-PSEP	TBOS	PLSC
Computations in signcryption	$T_r + 4T_h + 2T_{\exp}$	$T_r + 2T_h + 2T_{\exp}$	$T_r + 2T_h + 2T_{\exp}$
Computations in de-signcryption	$4T_h + 2T_{\exp}$	$3T_h + 2.5T_{\exp}$	$2T_h + 2T_{\exp}$
Ciphertext length	$\|m\| + 2\|N_U\|$	$\|N_U\|$	$\|m\| + 2\|N_U\|$

表 2 列举了对不同长度的消息, PLSC 中各组件所花费的 CPU 周期. 实验用 Crypto++ 4.2 来实现, 运行环境是 Windows 2000 SP 4, CPU 2.60GH, RAM 512MB, RSA 模长度为 1024 bit. 对于每个消息, 取 5 次实验的平均值. “MSGLEN” 表示消息长度; “EN_G”, “EN_H”, “S_RSA”和“R_RSA” 分别表示在进行签密过程中$\mathcal{G}(\cdot)$、$\mathcal{H}(\cdot)$在发送方私钥(签名密钥)作用下的 RSA 模指数运算及在接收方公钥(加密密钥) 作用下的 RSA 模指数运算; “R_DERSA”, “S_DERSA”, “DE_H”和“DE_G”分别表示进行解签密过程中在接收方私钥(解密密钥) 作用下的 RSA 模

指数运算、在发送方公钥(验证密钥) 作用下的 RSA 模指数运算, $\mathcal{H}(\bullet)$和 $\mathcal{G}(\bullet)$. 图 2 显示了每个组件的 CPU 消耗随消息长度的变化情况.

表 2　PLSC 中各组件对不同长度的消息所花费的 CPU 周期

MSGLEN (byte)	EN_G (10^7)	EN_H (10^7)	R_RSA (10^7)	S_RSA (10^7)	R_DERSA (10^7)	S_DERSA (10^7)	DE_H (10^7)	DE_G (10^7)	Total (10^7)
128	0.46	0.28	0.29	0.29	11.94	11.97	0.28	0.46	25.97
256	0.64	0.51	0.29	0.29	11.96	11.93	0.52	0.64	26.78
384	0.81	0.80	0.29	0.29	11.97	11.98	0.81	0.81	27.76
512	0.98	1.04	0.29	0.29	11.99	11.93	1.04	0.99	28.55
640	1.16	1.28	0.29	0.29	11.97	11.93	1.30	1.16	29.38
768	1.34	1.56	0.29	0.29	11.96	11.96	1.56	1.34	30.30
896	1.51	1.80	0.29	0.29	11.95	11.97	1.80	1.53	31.14
1024	1.69	2.08	0.29	0.29	11.95	11.94	2.09	1.70	32.03

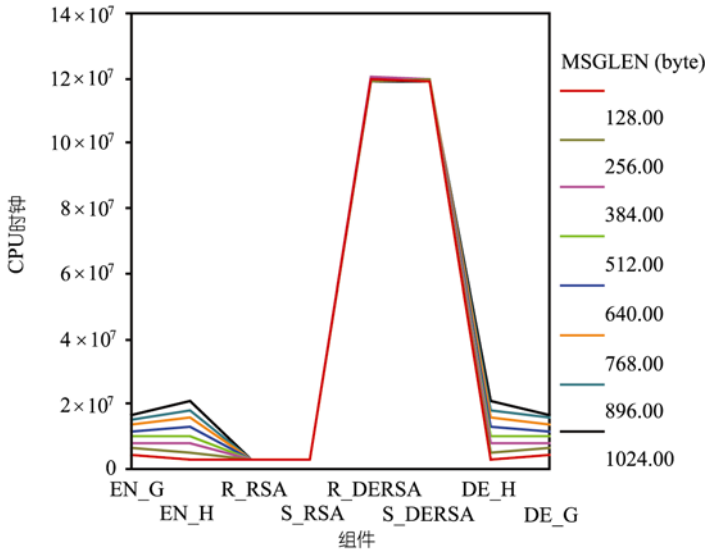

图 2　PLSC 中各组件所花费的 CPU 周期随消息长度的变化情况

实验结果显示, 对于不同长度的消息, Hash 函数所花费的 CPU 周期随着消息长度的增加而直线上升, 而单向陷门函数运算所花的时间却基本保持不变. 由于进行单向陷门置换所花的时间要远远大于 Hash 运算所花的时间, 因此, 总的时间消耗基本上仍保持在单向陷门置换所花的时间上.

5　关于 PLSC 的进一步讨论

在本文的 PLSC 方案中, 我们明确地将双方的公钥与要处理的消息 m 和随机串 r 邦定在一起. 这样做的目的是为了防止所谓的“身份伪造(identity fraud)攻击”. 在用户可以拥有多个密钥的情况下, 可以增强方案的安全性(内部安全).

为了简明起见, 在定义 3 中我们要求作为“签名”和“加密”用的两个 TDP 的输入具有相同的长度. 在实际上, 两个 TDP 的输入可以有不同的长度, 以适应

长度不等的用户密钥. 这一点, 对于两个具有不等长密钥的用户之间进行通讯来说是相当重要的. 例如, 在某地, 法律规定用户要使用 1024 bit RSA 密钥进行签名, 而加密所用的密钥则为 512 bit. 此时, 如果一对用户要进行通信, 发送方不必特地生成一个 512 bit 的临时签名密钥, 而直接使用发送方的 1024 bit 密钥与接收方的 512 bit 密钥进行通信.

如果某些应用要求签密方案具有“不可否认性”, PLSC也可达到这个目标. 如果接收方想要第三方验证一个签名密文, 它只需将c_1的解密结果与自己的公钥及消息的签名c_3一起作为验证的证据, 提交给第三方进行验证. 在这个过程中, 接收方无须向第三方泄露自己的私钥.

PLSC也可以应用在一些不能处理大块数据的环境中(如内存很小的智能卡). 在这种环境中, 我们建议PLSC中的Hash函数采用像PKCS#1[18]中的MGF那样的迭代方法来构造. 在签密时采用下面的“流”模式: 首先选择一个随机串并计算输出c_1; 然后对明文消息一组一组地产生密文c_2及Hash值g(迭代地产生); 最后对g实施OTP变换得到输出c_3.

6 结论

从以上的讨论可以看出, PLSC是一个非常实用的签密方案, 可以用来在各种各样的环境中设计具有高度安全保障的协议. 它具有高效率、易实现、内部安全及不可否认等优点. 在实际的应用中, PLSC还可能具有其他一些优点. 如用户可能会想到将PLSC中的TDP操作直接用现成的签名方案或加密方案代替. 这样的唯一不足是会带来性能上的稍稍降低(重复填充). 还有, PLSC操作也可以分布式地进行. 用户可以将对随机串r和$G(\cdot)$的“加密”和“签名”工作交给其他机器来完成. 当其他机器完成这些“加密”和“签名”时, 不能获得明文的任何有用信息. 此外, 还可以在要处理的明文消息到来之前对$f_R(\cdot)$和$H(\cdot)$进行预处理, 从而获得更好的性能.

如上所述, PLSC 是一个非常实用的长消息签密方案, 然而, 如果要处理的消息非常短($|m| < l$), 则它的带宽将变得非常糟糕(这一点与混合技术相同). 但与它在其他方面的优异表现相比, 我们认为这个问题是微不足道的.

参 考 文 献

1 An J H, Dodis Y, Rabin T. On the security of joint signature and encryption. In Advances in Cryptology—EUROCRYPT '02, LNCS Vol. 2332 (Knudsen L, ed.), Berlin: Springer-Verlag, 2002, 83－107. Available from http://eprint.iacr.org/2002/046/

2 Zheng Y. Digital signcryption or how to achieve cost (signature & encryption) cost (signature) + cost (encryption), In Advances in Cryptology—CRYPTO'97, LNCS Vol. 1294 (ed. Kaliski B S). Berlin: Springer-Verlag, 1997, 165－179

3 Zheng Y, Imai H. Efficient signcryption schemes on elliptic curves. Inform Process Letters, 1998, 68(6):

227－233[DOI]
4 Petersen H, Michels M. Cryptanalysis and improvement of signcryption schemes. IEEE Computers and Digital Communications, 1998, 145(2): 140－151
5 He W, Wu T. Cryptanalysis and improvement of petersen-michels signcryption schemes. IEEE Computers and Digital Communications, 1999, 146(2): 123－124[DOI]
6 Baek J, Steinfeld R, Zheng Y. Formal proofs for the security of signcryption. In 5th International Workshop on Practice and Theory in Public Key Cryptosystems PKC 2002, LNCS Vol. 2274 (eds. Naccache D, Pailler P). Berlin: Springer-Verlag, 2002, 80－98
7 Bellare M, Rogaway P. Optimal asymmetric encryption. In Advances in Cryptology——EUROCRYPT 94, LNCS Vol. 950 (ed. Santis A D). Berlin: Springer-Verlag, 1995, 92－111. Revised version available from http://www-cse.ucsd.edu/users/mihir/
8 Shoup V. OAEP reconsidered, In Advances in Cryptology——CRYPTO 2001, LNCS Vol. 2139 (ed. Kilian J). Berlin: Springer-Verlag, 2001, 240－259
9 Bellare M, Rogaway P. The exact security of digital signatures: How to sign with RSA and Rabin. In Advances in Cryptology——EUROCRYPT 96, LNCS Vol. 1070 (ed. Maurer U.), Berlin: Springer-Verlag, 1996, 399－416. Revised version appears in http://www-cse.ucsd.edu/users/mihir/papers/crypto-papers.html
10 Fujisaki E, Okamoto T. Secure integration of asymmetric and symmetric encryption schemes. In Advances in Cryptology——Proceedings of CRYPTO'99, LNCS Vol. 1666 (ed. Wiener M). Berlin: Springer-Verlag, 1999, 537－554
11 Shoup V. A proposal for an ISO standard of public key encryption (version 2.1). Cryptology ePrint Archive, Report 2001/112, http://eprint.iacr.org/2001/112
12 Dodis Y, Freedman M J, Jarecki S, Walfish S. Optimal Signcryption from Any Trapdoor Permutation. Cryptology ePrint Archive, Report 2004/20, http://eprint.iacr.org/2004/20
13 Mao W, Malone-Lee J. Two birds one stone: Signcryption using RSA. In Progress in Cryptology——CT-RSA 2003, LNCS Vol. 2612 (ed. Joye M). Berlin: Springer-Verlag, 2003, 210－224
14 Kobara K, Imai H. Oaep++ : A very simple way to apply oaep to deterministic ow-cpa primitives. Cryptology ePrint Archive, Report 2002/130, http://eprint.iacr.org/2002/130
15 Luby M, Rackoff C. How to construct pseudorandom permutations from pseudorandom functions. SIAM Journal on Computing, 1988, 17(2): 373－386[DOI]
16 Dodis Y, Freedman M J, Walfish S. Parallel Signcryption with OAEP, PSS-R, and other Feistel Paddings, Cryptology ePrint Archive, Report 2003/043. http://eprint.iacr.org/2003/043
17 An J H. Authenticated Encryption in the Public-Key Setting: Security Notions and Analyses. Cryptology ePrint Archive, Report 2001/079, http://eprint.iacr.org/2001/079
18 RSA Laboratories. PKCS #1 v2.1: RSA Encryption Standard. June 2002. Available from http://www.rsa.com/rsalabs/pubs/PKCS/
19 Rivest R L, Shamir A, Adlema L M. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 1978, 21(2): 120－126

附录 A

证明定理 1

A1　证明公式(1)

我们先来界定 $Adv_{SC,\mathcal{A}}^{ind-cca2}$ 的大小以证明公式(1). 设 f_S ， f_S^{-1} 分别是发送方的公钥与

私钥, f_R , f_R^{-1} 分别是接收方的公钥与私钥.

假设$\mathcal{A}$是一个攻击PLSC的IND-CCA2 安全性的敌手. 我们通过一系列攻击游戏G_0, …, G_4来逐渐对敌手$\mathcal{A}$成功的概率进行界定, 其中G_0就是敌手$\mathcal{A}$进行IND-CCA2 攻击的原始游戏. 设S_i表示敌手$\mathcal{A}$在游戏G_i中获得成功这一事件. 对于每一个游戏G_i, 我们将分析概率$Pr[S_i]$ 与概率$Pr[S_{i-1}]$之间差别. 敌手$\mathcal{A}$ 在G_0中成功的优势则通过其在游戏G_4中的成功概率$Pr[\boldsymbol{S}_4]$, 以及$Pr[\boldsymbol{S}_0]-Pr[\boldsymbol{S}_4]$的值来界定. 而$Pr[\boldsymbol{S}_0]-Pr[\boldsymbol{S}_4]$的值则通过各游戏之间的关系来界定. 在证明中, 我们用$c^* = c_1^*||c_2^*||\ \sigma^*$表示当敌手在进行IND-CCA2 攻击时, 提供给敌手的相应于挑战明文m_b (b是一个挑战比特)的挑战密文.

Game G_1:

我们构造的G_1与G_0除了用以下的模拟器来替换G_0中的随机Oracle、签密Oracle $\mathcal{SE}$ 以及解签密Oracle $\mathcal{VD}$ 之外完全一样.

IND-CCA2 攻击中的随机Oracle模拟器 $\tilde{\mathcal{G}}$ 和 $\tilde{\mathcal{H}}$. 每当$\mathcal{A}$向$\mathcal{G}$提出询问$(m^i||r^i,\ id_S||id_R)$时, $\tilde{\mathcal{G}}$ 先检查是否已经存在一个$\mathcal{G}$ $(m^i||r^i,\ id_S||id_R)$的定义. 如果存在, 则 $\tilde{\mathcal{G}}$ 向$\mathcal{A}$输出这个已经存在的值. 否则, 就随机选取一个串$\sigma^i\in\{0,\ 1\}^l$, 定义$\mathcal{G}$ $(m^i||r^i,\ id_S||id_R)$的值为 f_S (σ^i). 然后, $\tilde{\mathcal{G}}$ 将元组$(\sigma^i,\ m^i||r^i,\ id_S||id_R)$记录在查询表中, 以备以后查询(进行解签密Oracle $\mathcal{VD}$ 模拟时会用到), 同时向$\mathcal{A}$输出 f_S (σ^i).

以上的模拟仍使每次询问的返回值保持均匀分布, 因此其模拟是完善的(perfectly). 每次返回以前要计算一个 f_S (σ^i), 所花的时间为T_f. 所需的空间代价为$|\sigma| + |m| + |r| + 2|id| = 2l + n + 2|id|$, 其中$|id|$是$id_U$的比特长度.

随机Oracle $\mathcal{H}$的模拟器 $\tilde{\mathcal{H}}$ 与之类似, 其差别仅在于无需经过f_R (r)计算, 所需的空间代价为$|p| + |r| + 2|id| = n + l + 2|id|$.

IND-CCA2 攻击中的解签密Oracle模拟器 $\widetilde{\mathcal{VD}}$. 当敌手提交一个密文c进行询问时, $\widetilde{\mathcal{VD}}$ 先将该密文解析成$c_1||\ c_2\ ||\sigma$的形式. 然后 $\widetilde{\mathcal{VD}}$ 把自己作为密文的接收者, 在$\mathcal{G}$的查询表内查找有无包含$(\sigma,\ id_S||id_R)$的元组$(\sigma,\ m||r,\ id_S||id_R)$, 同时在$\mathcal{H}$的查询表中查找有无包含$(r,\ id_S||id_R)$的元组$(p,\ r,\ id_S||id_R)$. 如果二者之一查找失败, $\widetilde{\mathcal{VD}}$ 则返回非法符号“⊥”. 否则, 因为$\mathcal{H}(r,\ id_S||id_R)$已经被询问过, 于是就计算$m\leftarrow c_2\oplus\mathcal{H}(r,\ id_S||id_R)$, $c_1'\leftarrow f_R\ (r)$. 如果$c_1 = c_1'$成立, $\widetilde{\mathcal{VD}}$ 就返回m, 否则返回⊥. 如果查找成功, $\widetilde{\mathcal{VD}}$ 就能生产正确的输出. 其时间代价为T_f.

IND-CCA2 攻击中的签密 Oracle 模拟器 $\widetilde{\mathcal{SE}}$. 当敌手向签密 Oracle $\mathcal{SE}$提交一个消息 m 时, $\widetilde{\mathcal{SE}}$ 把自己作为消息的发送者, 按照下列的方式产生输出:

- 随机选择$r\in\{0,\ 1\}^l$, $c_2\in\{0,\ 1\}^n$; 计算 $c_1 = f_R(r)$, $p\ = m\oplus c_2$. 定义$\mathcal{H}(r,\ id_S||id_R)= p$. 如果$\mathcal{H}(r,\ id_S||id_R)$ 已经存在, 则失败.
- 随机选择$\sigma\in\{0,\ 1\}^l$, 计算$y = f_S(\sigma)$.
- 定义$\mathcal{G}(m||r,\ id_S||id_R) = y$. 如果$\mathcal{G}(m||r,\ id_S||id_R)$ 已经存在, 则失败.
- 输出$c_1||c_2||\sigma$.

如果不失败, 则 $\widetilde{\mathcal{SE}}$ 返回一个正确的签密Oracle询问. 因为$\sigma= f_S^{-1}(y)$, 其时间代价为 $2\cdot T_f$.

G_1使用以上定义的三个模拟器分别替代G_0中的三个Oracle. 在以上的模拟都成功时, G_0和G_1完成一样. 因此, $Pr[S_0]$–$Pr[S_1]$的值被界定在以上模拟失败的概率之内.

用符号$Fial_{\mathcal{VD}}$来表示模拟器$\widetilde{\mathcal{VD}}$失败这一事件. 则模拟器$\widetilde{\mathcal{VD}}$只有当敌手询问一个合法的密文而它却返回非法符号“⊥”时才算失败. 也就是说, 敌手$\mathcal{A}$向提交一个密文, 其中包含一个串r, 而这个串r从来没有向$\tilde{\mathcal{H}}$询问过(即没有相应的$\mathcal{H}(r, id_S||id_R)$记录), 或者包含$(m, r)$, 而$m||r$从来没有向$\tilde{\mathcal{G}}$询问过(即没有相应的$\mathcal{G}(m||r, id_S||id_R)$记录). 前一种情况意味着$p$的值是一个随机值, 其正好使$p=c_2 \oplus m$成立的概率为 2^{-n}. 后一种情况意味着g的值是一个随机值, 其正好使$g=f_S(\sigma)$成立的概率为 2^{-l}. 于是, 在经过q_D次询问后$\widetilde{\mathcal{VD}}$失败的概率为

$$Pr[Fial_{\mathcal{VD}}] \leqslant q_D(2^{-n}+2^{-l}). \tag{A1}$$

用符号$Fial_{\mathcal{SE}}$来表示模拟器$\widetilde{\mathcal{SE}}$失败这一事件. 则模拟器$\widetilde{\mathcal{SE}}$只有当一个新询问$m||r$使$(m||r, id_S||id_R)$与某个以前曾经的$(m'||r', id_S||id_R)$询问产生“碰撞”, 即值$\mathcal{G}(m||r, id_S||id_R)$或$\mathcal{H}(r, id_S||id_R)$已经存在时才会失败. 而对于数$q_S$, q_H 和q_G来说, 最多有q_S+q_H 个这样的$\mathcal{H}(r', id_S||id_R)$值存在, 最多有$q_S+q_G$个这样的$\mathcal{G}(m'||r', id_S||id_R)$值存在. 如果保持用户不变是随机选取的, 则一个新r值与以前的某个值r'发生碰撞的概率最多是$q_S \cdot (q_S+q_H) \cdot 2^{-l}$. 同理, 新$m||r$会发生碰撞的概率最多是$q_S \cdot (q_S+q_G) \cdot 2^{-(n+l)}$. 于是在经过$q_S$次询问后$\widetilde{\mathcal{SE}}$失败的概率为

$$\begin{aligned} Pr[Fial_{\mathcal{SE}}] &\leqslant q_S \cdot (q_S+q_H) \cdot 2^{-l} + q_S \cdot (q_S+q_G) \cdot 2^{-(n+l)} \\ &= q_S \cdot ((2^{-n}+1)q_S + q_H + 2^{-n} \cdot q_G) \cdot 2^{-l}. \end{aligned} \tag{A2}$$

用符号 $SimFial$ 表示以上模拟器有一个失败事件. 则由(A1)和(A2)式可得

$$\begin{aligned} Pr[SimFial] &\leqslant Pr[Fial_{\mathcal{VD}}] + Pr[Fial_{\mathcal{SE}}] \\ &\leqslant q_D(2^{-n}+2^{-l}) + q_S \cdot ((2^{-n}+1)q_S + q_H + 2^{-n} \cdot q_G) \cdot 2^{-l} \\ &\leqslant q_D \cdot 2^{-n} + (q_D + q_S \cdot ((2^{-n}+1)q_S + q_H + 2^{-n} \cdot q_G)) \cdot 2^{-l}. \end{aligned} \tag{A3}$$

因此

$$Pr[S_0]–Pr[S_1] \leqslant Pr[SimFial]. \tag{A4}$$

Game G_2:

我们对G_1稍加修改, 使其当敌手$\mathcal{A}$发出一个询问$\mathcal{H}(r^*, id_S||id_R)$(其中$r^*$表示挑战密文$c^*$对应的随机串)时就终止. 修改后的游戏记为$G_2$. 用符号$Halt_i$表示游戏$G_i$以这种方式终止. 因为除非$Halt_2$发生, G_2与G_1完全相同, 因此有

$$Pr[S_1]–Pr[S_2] \leqslant Pr[Halt_2]. \tag{A5}$$

Game G_3:

G_3与G_2相同, 只是返回给敌手$\mathcal{A}$的挑战密文中的第二部分(即c_2^*)替换为一个随机串. 注意到当敌手$\mathcal{A}$询问$\mathcal{H}(r^*, id_S||id_R)$时, 游戏就终止, 因此只要游戏还在进行, 敌手$\mathcal{A}$就不能获得$\mathcal{H}(r^*, id_S||id_R)$的任何信息. 因此, 如果将$c_2^*$替换为一个随机串$c_2'$, 我们有理由认为将$\mathcal{H}(r^*, id_S||id_R)$的值定义为$m_b \oplus c_2'$是正确的. 从这方面来看, G_3具有与G_2相同的概率分布.

$$Pr[S_3] = Pr[S_2]. \tag{A6}$$

Game G_4:

我们将G_3中的整个挑战密文c^*完全用随机串来代替, 从而得游戏G_4. 也就是说, 将

挑战密文中的c_1*和σ*也分别用两个随机串来替换. 在G_3中, c_2*已经被替换成了一个随机串, 所以敌手$\mathcal{A}$无法获得$m_b \oplus p$的任何信息, 当然也无法获得挑战明文m_b的信息. 因此, 对于任何σ', 我们可以认为$f_S(\sigma')= \mathcal{G}(m_b||r^*, id_S||id_R)$是正确的. 如果将$c_1$*和$\sigma$*也分别用两个随机串来替换, 则$c_1$*也不能为敌手提供任何明文信息了. 因此$G_4$具有与$G_3$相同的概率分布.

$$Pr[S_4] = Pr[S_3]. \tag{A7}$$

因为将整个挑战密文完全用一个随机串代替了, 它不会给敌手提供任何明文信息了. 此时, 如果游戏没有终止, 则敌手 $\mathcal{A}$ 只能以 1/2 概率来猜测其中的挑战比特 b. 于是

$$Pr[S_4] \leqslant 1/2 + Pr[Halt_4]. \tag{A8}$$

最后, 我们来界定G_2或G_4终止的概率. 以上的攻击游戏中, 各个模拟器都是在不知道秘密的陷门信息 f_R^{-1} 的条件下进行的. 这些模拟器可以作为对TDP的安全归约来使用. 先来看G_2的运行. r*是一个随机选取的串, 其单向陷门置换值$c^*_1 = f_R(r^*)$输出给敌手$\mathcal{A}$作为挑战密文的一部分. 当敌手提交一个形如$\mathcal{H}(r^*, \cdot)$ 的询问恰好使得 $f_R(r^*) = c^*_1$成立时, G_2终止. 因此, 如果G_2终止, 则敌手找到了c^*_1的原象r*, 也就成功地求出了TDP的逆. 所以有

$$Pr[Halt_2] \leqslant \varepsilon_{\mathrm{TDP}}, \tag{A9}$$

同理

$$Pr[Halt_4] \leqslant \varepsilon_{\mathrm{TDP}}. \tag{A10}$$

将(A3)~(A10)式结合起来, 可得

$$\begin{aligned} &Adv^{ind-cca2}_{SC,\mathcal{A}}(t, q_H, q_G, q_S, q_D) = 2 \cdot Pr[\mathrm{S}_0] - 1 \\ &\leqslant 2 \cdot (Pr[SimFial] + \varepsilon_{\mathrm{TDP}} + 1/2 + \varepsilon_{\mathrm{TDP}}) - 1 \\ &= 2 \cdot (Pr[SimFial] + 2 \cdot \varepsilon_{\mathrm{TDP}}) \\ &\leqslant q_D \cdot 2^{-n+1} + (q_D + q_S \cdot ((2^{-n}+1)q_S + q_H + 2^{-n} \cdot q_G)) \cdot 2^{-l+1} + 4 \cdot \varepsilon_{\mathrm{TDP}}. \end{aligned} \tag{A11}$$

公式(1)成立.

A2 证明公式(2)

我们再来界定 $Adv^{int-ctxt}_{SC,\mathcal{B}}$, 从而证明公式(2).

我们将敌手对PLSC的INT-CTXT安全性的攻击归约为对PLSC中的TDP的攻击. 在归约过程中要运行一个经过修改的敌手$\mathcal{B}$的攻击游戏, 像在界定 $Adv^{ind-cca2}_{SC,\mathcal{A}}$ 时的G_1一样, 也要构造一个随机Oracle模拟器、一个解签密Oracle模拟器和一个签密Oracle模拟器. 最后将敌手$\mathcal{B}$的攻击归约为对于一个挑战值y, 求其相应的原象x, 使得$y = f_U(x)$.

随机 Oracle 模拟器 $\tilde{\mathcal{G}}$ 和 $\tilde{\mathcal{H}}$ 以及签密 Oracle 模拟器 $\widetilde{SE}$. 这些模拟器与证明 IND-CCA2 安全性时的模拟器完全相同.

解签密Oracle模拟器 $\widetilde{\mathcal{VD}}$. 该模拟器也与证明IND-CCA2 安全性时的模拟器相同. 但是, 在进行INT-CTXT攻击时, 没有挑战密文, 所以只要$\mathcal{G}(m||r, id_S||id_R)$ 或 $\mathcal{H}(r, id_S||id_R)$没有被询问过, 或者未能通过正确地解签密(解签密时得到非法符号$\perp$), 该模拟器就输出$\perp$.

用符号 *SimFail* 来表示以上的模拟器中有一个不正确的响应. 随机 Oracle 的模拟器 $\tilde{\mathcal{G}}$ 和 $\tilde{\mathcal{H}}$ 总能正常地响应, 解签密 Oracle $\mathcal{VD}$ 的模拟器 $\widetilde{\mathcal{VD}}$ 和签密 Oracle $\mathcal{SE}$ 的模拟器 $\widetilde{\mathcal{SE}}$ 失败的情况与证明 IND-CCA2 安全性时的情况完全相同. 因此 *SimFail* 发生的概率为

$$\begin{aligned} Pr[SimFail] &\leqslant Pr[Fial_{\mathcal{VD}}] + \Pr[Fial_{\mathcal{SE}}] \\ &\leqslant q_D \cdot 2^{-n} + (q_D + q_S \cdot ((2^{-n}+1)q_S + q_H + 2^{-n} \cdot q_G)) \cdot 2^{-l}. \end{aligned} \tag{A12}$$

用符号***Bad***表示敌手$\mathcal{B}$产生了一个合法的伪造密文, 但其中的(m,r)使得相应的$\mathcal{G}(m||r, id_S||id_R)$ 或 $\mathcal{H}(r, id_S||id_R)$从来没有被询问过. 此时, 相应的$g$ 或 p是两个随机值, 其恰好使$g = f_S(\sigma)$成立的概率是 2^{-l}, 恰好使$p=c_2 \oplus m$成立的概率是 2^{-n}. 于是

$$Pr[\boldsymbol{Bad}] \leqslant 2^{-l}+2^{-n}. \tag{A13}$$

当$\neg\boldsymbol{Bad}$发生时, $\mathcal{G}(m||r, id_S||id_R)$ 和 $\mathcal{H}(r, id_S||id_R)$都曾被询问过. 此时, 如果敌手$\mathcal{B}$成功地伪造了一个密文$c'_1||c'_2||\sigma'$, 则其中的$\sigma'$一定是新的, 而$\mathcal{G}(m||r, id_S||id_R)$即是$\sigma'$的原象, 因为有$f_S(\sigma') = \mathcal{G}(m||r, id_S||id_R)$. 故有

$$Pr[\mathcal{B}\text{ success} \wedge \neg SimFail \wedge \neg\boldsymbol{Bad}] \leqslant \varepsilon_{\mathrm{TDP}}. \tag{A14}$$

结合(A12)~(A14), 可得

$$\begin{aligned} &Adv^{int-ctxt}_{SC,\mathcal{B}}(t, q_H, q_G, q_S, q_D) = Pr[\mathcal{B}\text{ success}] \\ &= Pr[\mathcal{B}\text{ success} \wedge SimFail] + Pr[\mathcal{B}\text{ success} \wedge \neg SimFail] \\ &\leqslant Pr[SimFail] + Pr[\mathcal{B}\text{ success} \wedge \neg SimFail] \\ &= Pr[SimFail] + Pr[\mathcal{B}\text{ success} \wedge \neg SimFail \wedge \boldsymbol{Bad}] + Pr[\mathcal{B}\text{ success} \wedge \neg SimFail \wedge \neg\boldsymbol{Bad}] \\ &\leqslant Pr[SimFail] + Pr[\boldsymbol{Bad}] + Pr[\mathcal{B}\text{ success} \wedge \neg SimFail \wedge \neg\boldsymbol{Bad}] \\ &\leqslant q_D \cdot 2^{-n} + (q_D + q_S \cdot ((2^{-n}+1)q_S + q_H + 2^{-n} \cdot q_G)) \cdot 2^{-l} + 2^{-l} + 2^{-n} + \varepsilon_{\mathrm{TDP}} \\ &= (q_D+1)\,2^{-n} + (q_D + q_S \cdot ((2^{-n}+1)q_S + q_H + 2^{-n} \cdot q_G) + 1) \cdot 2^{-l} + \varepsilon_{\mathrm{TDP}}. \end{aligned}$$

公式(2)成立.

A.3　证明公式(3)

最后我们来估计以上证明的复杂度. 在归约过程中各模拟器运行的时间大约为$O((q_G + 2 \cdot q_S + q_D)T_f)$, 其中$T_f$表示单个TDP操作所用的时间. 空间需求大约为$O((n + l + 2|id|)(q_H + q_S) + (2l+n+2|id|)(q_G + q_S))$, 其中$l$表示TDP的输入长度, 是$n$消息的长度, $|id|$是用户ID的长度.