DMSS-动态Merkle可信树签名方案

蔡永泉, 刘 芳
(北京工业大学计算机学院, 北京 100124)

摘 要: 本文通过构造子树及密钥的动态循环更新实现了二叉树的动态更新, 改进了Merkle可信树签名方案中, 签名数量的增加使得二叉树庞大导致签名效率低下的缺陷, 使签名的数量不在受到二叉树大小的影响. 此外, 本文对改进后的方案进行了安全分析, 分析结果表明, 该签名方案具有原始Merkle可信树签名方案的安全性, 并且, 由于采用了分时间段的密钥管理方式, 该方案还具有前向安全性.

关键词: Merkle可信树; 数字签名; 动态; 前向安全
中图分类号: TP393.08 文献标识码: A 文章编号: 0372-2112 (2009) 4A-097-05

DMSS-Dynamic Merkle Authentication-tree Signature

CAI Yong-quan, LIU Fang
(*College of Computer Science and Technology, Beijing University of Technology, Beijing* 100124, *China*)

Abstract: We achieve a dynamic update of the authentication-tree through creating the sub tree and dynamic update of the key. In the original signature, the number of the signature influence the scale of the authentication-tree, and the increase of the number of signature makes lower efficient. In our scheme, the number of signature is not affected by the scale of the authentication-tree. In addition, the paper do a safety analysis to the improved authentication Merkle tree signature, the results show that the signature of the improver Merkle authentication is as safe as the original Merkle authentication signature tree, and as used key management according the time, the The signature also has forward-secure.

Key words: merkle authentication-tree ; digital signature; dynamic; forward-secure

1 引言

1989年, 著名的密码学家Merkle[1]提出了Merkle可信树的思想, 在Merkle可信树认证中, 签名者可以通过一次签名来实现对大量数据的认证. 之后Merkle提出了基于可信树思想的Merkle可信树签名方案. 在该签名方案中, 它的安全性仅仅依赖于哈希函数的安全性, 而原始的签名方案, 如RSA[2], DSA[3], ECDSA[4], 其安全性都依赖于一些数学难题, 如大因素分解难题和离散对数难题. 在最近30年内, 学者们对离散对数问题和大因素分解难题的破解已经有了重大的突破, 所以基于大因素分解和离散对数问题的签名方案的安全性受到了威胁. 并且在Merkle可信树签名方案中, 没有太多的理论假设, 这使得Merkle可信树签名更具有实用价值, 这些都使得Merkle可信树签名成为当前的研究热点.

原始的Merkle可信树签名是静态的, 需要构造的二叉树大小与需要签名的数量相关, 签名的数量越大, 所需要构造的二叉树越大, 消耗的时间和空间代价也越大. 如果签名的数量达到一定的数量级, 由于其巨大的时间和空间代价, 原始的Merkle可信树签名方案是不可行的. 所以在原始的Merkle可信树签名中, 签名的数量是受限制的.

目前已经有一些学者在提高原始Merkle可信树签名数量方面做了一些研究, 其中有Johannes Buchmann提出的CMSS[5]方案及之后提出的GMSS[6]方案, CMSS方案和GMSS方案较原始的Merkle签名方案虽然大大地增加了签名的数量, 但是签名的数量仍然不是完全不受限制的.

本文针对原始Merkle签名方案的不可动态更新的缺陷, 第一次提出了动态的Merkle签名的思想, 该思想使Merkle签名方案的签名数量不再受到限制. 此外, 本文提出的动态Merkle签名方案对原始Merkle签名方案的安全性也进行了改进, 它具有前向安全性.

2 原始的Merkle可信树签名方案

2.1 Merkle可信树认证

收稿日期: 2008-09-11; 修回日期: 2008-12-20
基金项目: 国家973重点基础研究计划(No. 2007CB11106)

在一次签名方案中, A 需要将签名公钥 Y 值告诉 B, 那么 B 如何知道 A 发送过来的 Y 是真实的呢? 我们可以利用 Merkle 可信树来对 Y 值的真实性进行认证. 原始的 Merkle 可信树是一棵完全二叉树, 二叉树的每个叶子节点都与一个 $Y_i(0< i< s)$ 相对应. 二叉树中的每个节点都有一个哈希函数值与之对应. 对于叶子节点的哈希函数值是通过对认证数据 Y_i 进行哈希运算得到的, 而中间节点的哈希函数值是将其孩子节点的哈希函数值进行哈希运算得到的. 以此类推, 可以得到根节点的哈希函数值, 将根节点的哈希函数值作为公钥. 每个叶子节点都有一个与之对应的认证路径, 每个叶子节点都能通过其自身的哈希函数值和其对应的认证路径上的节点的哈希函数值计算得到根节点的值, 将计算得到的根节点值与公钥比较, 若相等则接受签名. 叶子节点的认证路径是指, 从叶子节点到根的路径上的所有节点的兄弟节点.

2.2 Merkle 签名方法(MSS)

$H:\{0,1\}^* \rightarrow \{0,1\}^s$ 是一个加解密哈希函数, 假设一次签名方案已经给出.

MSS 密钥对产生: 首先, 产生 2^h 个一次签名密钥对 (X_i, Y_i), $i=1, \cdots, 2^h$, X_i 是签名密钥, Y_i 是认证密钥. MSS 私钥是一次签名密钥序列. 为了决定 MSS 的公钥, 构建一棵二叉树如下: 考虑每一个认证密钥 Y_i, 认证树的叶子节点是认证密钥的哈希函数值 $H(Y_i)$, MSS 的公开密钥是认证树的根.

MSS 签名产生: 以使用第 i 个密钥对 (X_i, Y_i) 对文件 d 进行签名为例, 签名包含序列号 i, 第 i 个认证密钥 Y_i, 由第 i 个签名密钥 X_i 计算的 σ, 以及对应于认证密钥 Y_i 的认证路径 A.

MSS 签名认证: 为认证一个 MSS 签名 (i, Y, σ, A), 认证者首先认证一次签名 σ 的正确性, 然后用 A 验证 Y 的正确性, 若两者的验证值都正确则接受签名. 否则不接受签名.

3 DMSS-动态 Merkle 可信树签名方案

动态 Merkle 可信树是在原始树型结构上的改进, 原始树型结构是一棵完全二叉树. 在 DMSS 方案中, 我们使用两层结构, 即使用两棵数, 一棵为主树, 一棵为子树, DMSS 的动态性体现在对主树和子树的更新上, 这里我们设主树和子树的高度均为 h, 即主树和子树均含有 2^h 个叶子节点. 数据是用子树的叶子节点对应的密钥进行签名的, 子树的根并不是签名公钥, 子树的根是由主树相应的叶子节点进行认证的. 主树的根才是签名公钥. 在动态 Merkle 可信树中, 我们先生成整棵主树、主树第一个叶子节点的认证路径节点值、第一棵子树、第一棵子树第一个叶子节点的认证路径节点值, 现在可以使用子树的第一个叶子节点对应的密钥进行签名了, 在使用第一棵子树叶子节点对应的密钥对进行签名的过程中, 第二棵子树的第一个叶子节点的密钥对产生, 第一棵子树的第一个叶子节点的签名完成之后继续使用第一棵子树的第二个叶子节点的密钥进行签名, 以此类推, 直到第一棵子树的所有叶子节点的密钥都用于签名之后, 即产生了 2^h 个签名之后, 第二棵子树构造完毕. 同时, 主树的第一个叶子节点的密钥开始更新. 之后, 产生第二棵子树的所有签名, 这时主树的第二个叶子节点的密钥开始更新, 并更新主树中与第一个叶子节点和第二个叶子节点相对应的父母节点. 即此时主树中的一棵包含三个节点的子树已经更新完毕. 一个叶子节点的认证路径包含所有从该叶子节点到根的路径上的节点的兄弟节点.

整个签名过程由三个部分组成: 密钥产生阶段、数字签名阶段、认证阶段.

3.1 DMSS 密钥产生

在密钥产生阶段, 我们使用了伪随机数产生器 PRNG. 对于 PRNG 来说, 只需要输入一个种子, 就能产生密钥输出. 相同的种子输入会产生相同的密钥输出. 使用了伪随机数产生器后, 我们不需要将所有的密钥对作为私钥, 而只需要将种子作为私钥保存即可. 在密钥产生阶段, 我们的任务是产生签名过程所需要的公钥和私钥. 由上面的介绍, 我们已经知道, DMSS 的私钥是产生密钥对的种子, 即认证路径序列对, 公钥是主树的根节点的哈希函数值. 其中使用了 Winternitz 一次签名[1]密钥对产生算法来产生密钥对.

算法 1: Winternitz 密钥产生算法

输入: 种子 $seed_{i-1}$

步骤 1: 计算 $(seed_i, s_0)=f(seed_{i-1})$.

步骤 2: 对于 $k=1, \cdots, t$,

计算 $(s_k, x_k)=f(s_{k-1})$.

步骤 3: 设置 $X_i=(x_1, \cdots, x_t)$.

步骤 4: 对于 $k=1, \cdots, t$,

计算 $y_k=H^{2^w-1}(x_k)$.

步骤 5: 计算 $Y_i=H(y_1, \cdots, y_t)$.

输出: Y_i 和 $seed_i$.

算法 2: DMSS 的密钥产生算法

系统参数:

$PRNG f:\{0,1\}^s \rightarrow \{0,1\}^s \times \{0,1\}^s$.

哈希函数 $H:\{0,1\}^* \rightarrow \{0,1\}^s$.

$t=\lceil s/w \rceil+\lceil (\lfloor \log_2 \lceil s/w \rceil+1+w \rfloor)/w \rceil, w \in N$.

输入: 树的高度 $h \in N$, 随机选择 $\zeta_0 \in_R \{0,1\}^s$ 和 $x_0 \in_R \{0,1\}^s$.

步骤 1: 令 $N=2^h$, 初始时, 设 i、j 为 0.

步骤 2: 初始化主树的栈 $stack_{main}$, 子树的栈 $stack_{sub}$.

步骤 3: 初始化主树认证路径节点队列 A_{main}, 子树认证路径节点队列 A_{sub}.

步骤 4: 初始化主树下一个认证路径队列 B_{main}, 子树下一个认证路径队列 B_{sub}.

步骤 5: 将 ζ_{i-1}输入算法 1 中, 求得 Y_i 和 ζ_i 的值.

步骤 6: 计算 $H(Y_i)$, 设 $a = H(Y_i)$.

步骤 7: 如果 a 在树中的高度与栈顶元素在树中的高度不相同, 执行步骤 12.

步骤 8: 如果 a 比 A_{main}中的最后一个节点在树中的高度高, 则将 a 节点加入 A_{main}序列中.

步骤 9: 将栈顶元素 $stack_{top}$出栈.

步骤 10: 计算 $a = H(stack_{top} \| a)$.

步骤 11: 将 a 节点压入栈 $stack_{main}$.

步骤 12: $i++$, 当 $i \leqslant N$, 返回步骤 5.

步骤 13: 取出 $stack_{main}$中的节点, 该节点值即为根节点的哈希函数值, 即为公钥 R.

步骤 14: 令 $\zeta_{next} = \zeta_N$.

步骤 15: 将 x_{j-1}输入算法 1 中, 得到 Y_j 和 x_j 的值.

步骤 16: 计算 $H(Y_j)$, 设 $a = H(Y_j)$.

步骤 17: 如果 a 在树中的高度与栈顶元素在树中的高度不相同, 执行步骤 22.

步骤 18: 如果 a 比 A_{sub}中的最后一个节点在树中的高度高, 则将 a 节点加入 A_{sub}序列中.

步骤 19: 将栈顶元素 $stack_{top}$出栈.

步骤 20: 计算 $a = H(stack_{top} \| a)$.

步骤 21: 将 a 节点压入栈 $stack_{sub}$.

步骤 22: $j++$, 当 $j \leqslant N$, 返回步骤 15.

步骤 23: $stack_{sub}$中的仅剩的一个节点值为子树的根节点值, 即 R_1.

步骤 24: 令 $x_{next} = x_N$.

在该算法中, ζ_0 是主树产生密钥对的种子, x_0 是子树产生密钥对的种子. $stack_{main}$用来存放计算主树根节点的哈希函数值时的中间节点值, A_{main}用来存放主树的第一个叶子节点的认证路径节点值. $stack_{sub}$用来存放计算子树根节点的哈希函数值时的中间节点值, A_{sub}用来存放子树的第一个叶子节点的认证路径节点值. B_{sub}用来存放将要使用的子树的认证路径节点值. 从步骤 5 到步骤 14, 计算出了更新主树密钥的种子 ζ_{next}. 从步骤 15 到步骤 24, 计算出了更新子树密钥的种子 x_{next}. 整个 DMSS 的私钥为 ζ_0、x_0、ζ_{next}、x_{next}、A_{main}、A_{sub}、B_{sub}、$stack_{main}$、$stack_{sub}$、$stack_{main\_next}$、$stack_{sub\_next}$、R_1、τ、i、j. 其中 i 表示主树目前正在进行签名的节点序列号, j 表示子树目前正在进行签名的节点序列号. τ 存储子树根节点的签名.

3.2 DMSS 签名产生

我们使用 DMSS 的私钥来进行签名, 在签名过程中, 我们不仅要完成签名的过程, 我们还要进行密钥的更新, 子树的更新和主树的更新操作. 这些操作体现在私钥的变化上.

算法 3: 签名产生算法

系统参数:

哈希函数 $H: \{0,1\}^* \rightarrow \{0,1\}^s$,

$PRNG f: \{0,1\}^s \rightarrow \{0,1\}^s \times \{0,1\}^s$.

输入: 签名数据 d, 私钥 ζ_i、x_j、ζ_{next_i}、x_{next_j}、A_i、B_j、B_{sub}、$stack_{main}$、$stack_{sub}$、$stack_{main\_next}$、$sack_{sub\_next}$、R_i、i、j、τ. 初始时 $x_{next\_j} = x_{next}$, $\zeta_{next\_j} = \zeta_{next}$, $i = 1$, $j = 1$.

输出: 签名及更新后的私钥.

步骤 1: 将 x_j 输入算法 1 中, 获得了一次签名密钥对 (X_j, Y_j)和 x_{j+1}.

步骤 2: 使用 X_j 和一次签名算法对数据 d 进行签名, 得到签名 σ_j.

步骤 3: 用 x_{j+1}代替 x_j.

步骤 4: 当 $j = 1$ 时, 执行步骤 5 到 6. 否则执行步骤 7.

步骤 5: 将 ζ_i 输入算法 1, 得到一次签名密钥对 (X_i, Y_i)和 ζ_{i+1}.

步骤 6: 用 ζ_{i+1}替换 ζ_i.

步骤 7: 使用 X_i 和一次签名算法对 R_i 进行一次签名, 得到签名 τ_i, 将 τ_i 的值放在 τ 中.

步骤 8: 设置签名为

$$sig = (i, j, \sigma_j, \tau, B_j, A_i)$$

步骤 9: 使用 Szydlo[7] 的认证路径算法, 输入 B_j、x_j、$stack_{sub}$得到 B_{j+1}.

步骤 10: 用 B_{j+1}替换 B_j, 即更新了子树的认证路径.

步骤 11: 开始构造下一棵子树的部分节点.

步骤 12: 将 x_{next_j} 输入算法 1 中, 得到 (X_{next_j}, Y_{next_j})和 x_{next_j+1}.

步骤 13: 用 x_{next_j+1}替换 x_{next_j}.

步骤 14: 计算 $H(Y_{next\_j})$ 的值, 并设置 $a = H(Y_{next\_j})$.

步骤 15: 如果 a 在树中的高度与栈顶元素在树中的高度不相同, 转到执行步骤 20.

步骤 16: 如果 a 比 B_{sub}中的最后一个节点在树中的高度高, 将 a 节点加入 A_{sub}序列中.

步骤 18: 将栈顶元素 $stack_{top}$出栈.

步骤 19: 计算 $a = H(stack_{top} \| a)$.

步骤 20: 将 a 节点压入栈 $stack_{sub\_next}$.

步骤 21: 如果 $j < 2^h$, 则 $j = j + 1$, 返回步骤 1.

步骤 22: 令 R_{i+1} 是 $stack_{sub\_next}$ 中的唯一节点, R_{i+1} 是第 $i+1$ 棵子树的根节点.

步骤 23: 使用 Szydlo 的认证路径算法, 输入 A_i、ζ_i、$stack_{main}$ 得到 A_{i+1}.

步骤 24: 用 A_{i+1} 代替 A_i.

步骤 25: 用 R_{i+1} 替换 R_i, 用 B_{sub} 代替 B_j.

步骤 26: 令 $j = 1$, $i = i + 1$.

步骤 27: 将 ζ_{next_i} 输入算法 1 中, 得到密钥对 (X_{next_i}, Y_{next_i}) 和 ζ_{next_i+1}.

步骤 28: 用 ζ_{next_i+1} 替换 ζ_{next_i}.

步骤 29: 计算 $H(Y_{next\_i})$, 令 $a = H(Y_{next\_j})$.

步骤 30: 执行步骤 15 到步骤 20, 将 $stack_{sub\_next}$ 换成 $stack_{main\_next}$.

步骤 31: 当 $i = 2^h + 1$ 时, $i = 1$ 转到步骤 1.

步骤 1 使用一次签名密钥产生算法产生一次签名密钥对. 在步骤 2 获得了数据的签名, 步骤 7 获得了子树根的一次签名. 步骤 9 和步骤 10 更新了子树的认证路径. 步骤 12 到步骤 20, 更新了下一棵子树的部分节点, 其中 $next\_j = j$. 步骤 21 到步骤 30 更新主树节点, 其中 $next\_i = i - 1$.

3.3 DMSS 认证过程

在认证阶段, 我们知道签名信息 $sig = (i, j, \sigma_j, \tau_i, B_j, A_i)$, 以及公开密钥 R.

步骤 1: 验证 σ_j 的正确性.

步骤 2: 验证 τ_j 的正确性.

步骤 3: 根据 σ_j 和 B_j 的值计算根节点 R_i 的值, 判断与由 τ_i 得到的 R_i 值是否相等. 相等则成立.

步骤 4: 根据 τ_i 和 A_i 的值计算 R 的值, 判断与公开密钥 R 是否相等. 相等则成立.

以上四个步骤都成立时, 则接受这个签名.

4 安全性分析

4.1 前向安全性

具有前向安全的数字签名是将密钥对进行分时间段的管理. 具有前向安全的数字签名确保了当前签名之前的所有签名的安全性. 在动态 Merkle 可信树数字签名的签名阶段, 签名的私钥中包含 ζ_i 和 x_j, ζ_i 为主树的产生密钥对的种子, x_j 为子树的产生密钥对的种子. 在签名阶段的步骤 9 中, 我们用 x_{j+1} 代替了 x_j, 即产生当前子树的密钥对的种子已经被新的种子取代了, 即使该种子被攻击者截获, 攻击者只能产生当前时段以后的签名, 而不可能产生当前时段之前的签名, 所以子树的签名是具有前向安全性的. 在签名阶段的步骤 13 中, 我们用 ζ_{i+1} 替换 ζ_i, 同上所述可知, 主树上也是具有前向安全性的, 故我们可以得出结论, 动态 Merkle 可信树数字签名方案是具有前向安全性的数字签名方案.

4.2 防选择性伪造攻击

原始的 Merkle 可信树签名已经被 Luis Carlos Coronado Garc′ıa[9] 证明具有防选择性伪造攻击, 原始的 Merkle 可信树签名具有防选择性伪造攻击主要基于所选择的哈希函数的抗冲突性, 动态的 Merkle 可信树签名方法中仍然需要选用具有抗冲突性的哈希函数, 所以它也具有防选择性伪造攻击.

4.3 防抵赖性

Merkle 可信树签名与其它签名方案最大的区别在于, 它的安全性是依赖于哈希函数的安全性, 哈希函数 $y = F(x)$ 具有如下特点, 对于给定的 x 值计算 y 是很容易的, 但是知道 y 值想要计算 x 值是不可能的. 并且不存在两个不同的 x 值 x_1 和 x_2, 使 $F(x_1) = F(x_2)$ 成立. 即只有知道正确的 x 值的人才能得到正确的 y 值. 现假设签名方为 A, 验证签名方为 B, A 发送一个 y 值给 B, 然后 A 要对某信息进行签名时, 发送对应的 x 值给 B, 这时 B 就能知道该签名确实是 A 发送过来的. 因为 y 值是 A 发送给 B 的, 只有 A 才知道正确的 x 值, 所以 A 不可能对签名进行抵赖. 即该签名方法具有防抵赖的功能.

4.4 数据完整性

在一般的签名方案中, 使用一个 *MAC* 校验码来保证数据的完整性, 在动态 Merkle 可信树签名方法中, 利用认证路径来保证数据的完整性, 因为如果发送的认证密钥 y 发生了改变, 则在验证阶段计算出的根节点 R 值将与公开密钥 R 值不符, 这将导致验证失败, 并且签名被拒绝. 同时认证密钥 y 的完整性和哈希函数的单向性也保证了发送数据信息的完整性. 由于 y 值改变会导致签名被拒绝, 所以要改变发送信息必须知道 y 所对应的 x 值, 但是根据哈希函数的单向性可知, 这是不可能的. 故本方法能保证数据的完整性.

5 效率分析

在原始 Merkle 可信树数字签名中, 每次产生新的签名数据时, 需要重新构造整棵二叉树节点值, 不具有动态更新性, 从而使算法的效率低下. 而在我们的动态 Merkle 可信树签名方案中, 每次有新的签名数据到达时, 只需要更新其中的一部分子树, 大大减少了更新节点的数量. 从而提高了签名的效率.

6 结论

本文在原始静态 Merkle 可信树签名的基础上, 首

次提出了动态 Merkle 可信树签名思想,其动态性主要体现在分层结构中主树和子树的更新上. 动态方案因其动态特性,二叉树大小不受签名数量的限制,该方案改进了原始方案中因需构造庞大的二叉树而导致的效率低下. 在安全性方面,动态的 Merkle 可信树签名在原始方案的基础上通过密钥的更新及删除操作提供了前向安全性. 所以动态 Merkle 可信树签名方案无论在效率还是安全性上都对原始 Merkle 可信树方案进行了改进.

参考文献:

[1] Ralph C Merkle. A certified digital signature[A]. Advances in Cryptology-CRYPTO' 89 [C]. Berlin: Springer-Verlag, 1989. 218- 238.

[2] R L Rivest, A Shamir, L Adleman. A method for obtaining digital signatures and public-key cryptosystems[J]. New York: Communications of the ACM, 1978, 21(2): 120- 126.

[3] T Elgamal. A aublic key cryptosystem and a signature scheme based on discrete logarithms[A]. Advances in Cryptology-CRYPTO' 89[C]. Berlin: Springer-Verlag, 1985. 10- 18.

[4] D Johnson. A Menezes. The Elliptic Curve Digital Signature Algorithm (ECDSA)[R]. Waterloo: Technical report CORR, 1999. 99- 34.

[5] Johannes Buchmann, Luis Carlos Coronado Garcia, Erik Dahmen, Martin D¨oring, and Elena Klintsevich. CMSS-an improved Merkle signature scheme[A]. Proc. Progress in Cryptology-INDOCRYPT[C]. Berlin: Springer-Verlag, 2006. 431- 238.

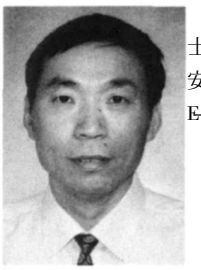
[6] Johannes Buchmann, Erik Dahmen, Elena Klintsevich, Katsuyuki Okeya, and Camille Vuillaume. Merkle signatures with virtually unlimited signature capacity[A]. In Proc Progress in Cryptology-INDOCRYPT[C]. Berlin: Springer-Verlag, 2007. 31- 45.

[7] M Szydlo. Merkle tree traversal in log space and time[A]. Advances in Cryptology-EUROCRPT 2004[C]. Berlin: Springer-Verlag, 2004. 541- 554.

[8] M Bellare, S Miner. A forward-secure digital signature scheme [A]. Advances in Cryptology-CRYPTO' 99[C]. Berlin: Springer-Verlag, 1999. 431- 448.

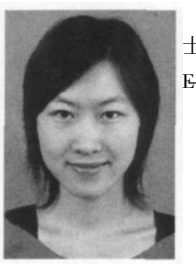
[9] Luis Carlos Coronado Garc´ıa. On the security and the efficiency of the Merkle signature scheme[DB/OL]. Cryptology ePrint Archive, 2005, http://eprint.iacr.org/2005/192/.

作者简介:



蔡永泉　男, 1956 生于安徽, 博士, 教授, 博士生导师, 主要从事计算机网络协议开发、网络安全及其算法的研究.

E-mail: cyq@bjut.edu.cn



刘　芳　女, 1984 年生于湖南省临湘市, 硕士, 主要从事信息安全方面的研究.

E-mail: liuf1216@yahoo.cn