基于默克尔前缀树的 TPM 外部密钥管理方案

肖勇才[1],徐　健[1],邱日轩[1],李　腾[2],卢　笛[3]
(1. 国网江西省电力有限公司电力科学研究院,江西 南昌 330096;
2. 西安电子科技大学 网络与信息安全学院,陕西 西安 710071;
3. 西安电子科技大学 计算机科学与技术学院,陕西 西安 710071)

摘　要:密钥管理是可信平台模块(Trusted Platform Module,TPM)最为关键的功能之一。但由于 TPM 内部存储空间有限,大量子密钥需要存储在外部空间,有效保护这些外部密钥可以保证系统的安全性。然而传统 TPM 规范存在限制,其无法单独撤销某个无效密钥,只能一次性撤销所有密钥,导致外部密钥管理复杂。为解决该问题,引入了默克尔前缀树(Merkle Patricia Tree,MPT)。该方案将 MPT 树根节点安全存储在 TPM 中,其余节点均存储在 TPM 外部。新增、撤销密钥时,可以动态地向 MPT 树增加或删除密钥分支。校验密钥时,可以利用其字典树特性生成由根节点至密钥对应叶节点的存在性证明路径,从而建立高效的 TPM 外部密钥管理系统。利用 TPM 芯片 Z32H330TC 在树莓派 4B 开发板中实现了原型系统。实验证明,基于 MPT 树的外部密钥管理方案相比现有研究方案,新增、撤销、验证密钥的效率均高于已有方案,并极大降低了外部存储占用空间。

关键词:密钥管理;可信平台模块;默克尔前缀树;存在性证明;密钥散列值
中图分类号:TP309　　**文献标识码**:A　　**文章编号**:1673-629X(2024)12-0073-08
doi:10.20165/j.cnki.ISSN1673-629X.2024.0249

TPM External Key Management Solution Based on Merkle Patricia Tree

XIAO Yong-cai[1],XU Jian[1],QIU Ri-xuan[1],LI Teng[2],LU Di[3]
(1. State Grid Jiangxi Electric Power Research Institute,Nanchang 330096,China;
2. School of Cyber Engineering,Xidian University,Xi'an 710071,China;
3. School of Computer Science and Technology,Xidian University,Xi'an 710071,China)

Abstract:Key management is one of the most critical functions of Trusted Platform Module (TPM). However,due to the limited internal storage of TPM,a large number of sub-keys need to be stored in external space. Effective protection of these external keys can ensure the security of the system. Unfortunately,there are limitations in the traditional TPM specification. It cannot revoke an invalid key individually and can only revoke all keys at once,making external key management complicated. In order to solve this problem,we introduce Merkle Patricia Tree (MPT). In this scheme,the MPT root node is securely stored within TPM,while the remaining nodes are stored externally. When adding or revoking keys,the MPT dynamically adds or deletes branches. When verifying keys,its trie properties are used to generate a proof path from the root node to the leaf node corresponding to the key,thereby establishing an efficient external key management system for TPM. We implement a prototype system using the TPM chip Z32H330TC on a Raspberry Pi 4B board. Experiments demonstrate that the external key management scheme based on MPT is more efficient than existing schemes in adding,revoking,and verifying keys,and significantly reduces external storage space usage.

Key words:key management;trusted platform module;Merkle Patricia Tree;existence proof;key hash value

0　引　言

可信平台模块(Trusted Platform Module,TPM)是一种硬件安全模块,旨在提供计算机系统的安全基础。它通常嵌入在计算机的主板上,为系统提供了多种安全功能,包括密钥管理、数据加密认证、随机数生成和安全引导等[1]。TPM 的主要目标是保护计算机免受物理和逻辑攻击,并且恶意软件无法篡改 TPM 的安全功能,以确保系统的安全性和完整性。比如在最新的

收稿日期:2024-04-11　　**修回日期**:2024-08-13
基金项目:国家自然科学基金资助项目(62272370);国家重点研发计划(2023YFB2904000);中国科协青年人才托举工程(2022QNRC001)
作者简介:肖勇才(1987-),男,高级工程师,硕士,研究方向为电力物联网通信安全、物联终端漏洞挖掘技术;李　腾(1991-),男,副教授,博士,研究方向为软件漏洞挖掘;通信作者:卢　笛(1983-),男,副教授,博士,研究方向为可信计算、云计算安全。

操作系统 Window 11 中,TPM 2.0 作为最低硬件要求,必须默认实现并被开启[2]。智能电网中,TPM 也被用于实现终端的可信网络连接和用户数据的隐私保护[3-4]。现阶段,通过将虚拟化的 TPM 分配给虚拟机,虚拟机的使用者也像使用物理 TPM 一样调用加解密和完整性度量等功能,从而保护云计算的安全性[5-7]。

密钥管理是 TPM 最重要的功能之一,通常 TPM 基于安全存储在内部的主密钥,派生出子密钥以供用户使用。然而因为 TPM 内部的存储空间有限,因此大量派生的子密钥只能存储在外部空间中。保护这些外部密钥的安全性成为了计算机安全领域的一个重要挑战。然而,传统的 TPM 规范存在一些限制,其中最显著的问题之一是无法有效地单独撤销某个无效密钥,而只能一次性撤销所有密钥[8],这导致了外部密钥管理的复杂性。现有的一些研究中,基于变色龙认证树和默克尔哈希树(Merkle Hash Tree,MHT)构建了外部密钥管理结构[9-10],解决了撤销单一无效密钥的问题。但是这两种结构存在数据检索、更新效率低、树结构存储信息量少的问题。在树结构被破坏或删除的情况下,难以对数据进行恢复。

该文的主要贡献如下:

(1)为 TPM 的外部密钥管理引入了默克尔前缀树(Merkle Patricia Tree,MPT)[11-12]。通过将外部密钥使用散列函数计算得出的散列值存储在 MPT 树中,这些散列值通过 MPT 树的节点结构相互关联,建立了高效的密钥管理系统。

(2)依据 MPT 树所提供的安全性证明机制,该文提供了外部密钥串在 MPT 树中的存在性证明。同时,在 MPT 树被攻击者破坏时,密钥持有者仍可以利用其字典树的特性,快速恢复出完整的树结构。这有效保证了存储在 MPT 树中外部密钥的完整性,从而增强了 TPM 密钥管理的安全性。

(3)利用 TPM 芯片 Z32H330TC[13]在树莓派 4B 开发板中实现了基于 MPT 树的外部单一密钥撤销方案。实验结果表明,基于 MPT 树的外部密钥管理方案与现有研究方案相比,新增、撤销、验证密钥的效率均高于已有方案,并极大降低了外部存储占用空间。

1 背景及相关工作

1.1 TPM 外部密钥管理

TPM 内部存储空间有限,生成的密钥绝大部分并不会存储于较为安全的 TPM 内部,而是经过父密钥加密后再存储于外部存储空间,不完全受 TPM 控制。TPM1.2 和 TPM2.0 规范中未提供相关命令撤销单一无效的密钥的相关命令,只提供了撤销所有密钥的命令。若不撤销该无效的密钥,攻击者可能会将其加载到 TPM 中使用,会带来安全隐患。

为了弥补 TPM 外部密钥管理的缺陷,Katzenbeisser 等人[8]提出了黑名单和白名单两种外部密钥撤销方案。其中,黑名单是一条包含了所有已撤销密钥的哈希链。当 TPM 加载外部密钥时,需要同时依次将哈希链中的密钥加载进 TPM 中。如果该密钥存在于黑名单中,则停止该密钥的使用。而白名单中存储了每个有效密钥与当前白名单版本号联合计算的哈希值,其中白名单版本号被安全地保存在 TPM 内。当且仅当密钥哈希值包含的版本号与 TPM 中一致时,密钥有效。通过黑白名单结合的方式,可以较为灵活地创建和撤销 TPM 外部密钥。

Yu 等人[9]提出了基于变色龙认证树(Chameleon Authentication Tree,CAT)[14-15]的动态密钥管理方案。该方案中认证树每一层最左侧的节点为其两个子节点的变色龙散列函数计算结果,而其余的左节点用于存储子节点的普通散列值,右节点存储子节点的变色龙散列值,所有的密钥信息都在叶子节点中,并自底向上增长形成完整的二叉树。其中,最左侧的节点存储在 TPM 中,当使用密钥时,由叶子节点沿着路径直至计算对应最左侧节点的值,并与 TPM 内的相应节点值比较,如不相同则说明密钥被撤销。

Yu 等人[10]提出了基于动态 Merkle 树和静态 Merkle 树两种结构的密钥撤销方案。未撤销的密钥散列值和撤销密钥与撤销标志拼接后的散列值都存放在叶节点中。树的根节点保存在 TPM 中用以验证密钥的有效性。这些基于二叉树的方案能在一定程度上提升密钥的验证效率,但是当密钥增多时树的规模亦会增大,密钥的管理效率也会下降。

1.2 MPT 树

MPT 树具有多种节点类型,包括扩展节点、分支节点、叶子节点、根节点和空节点,如图 1 所示。

扩展节点(Extension Node):由于所有数据的散列值都是十六进制串,所以存储在 MPT 树中的数据均可用 0-9,a-f 这 16 个字符表示。扩展节点的作用是合并这些十六进制串的公共前缀,因此扩展节点的 next_node 必然连接一个分支节点,如图 1 中“2a3bd”和“2a56f”等数据串共享了 value 值为“2a”的扩展节点。

分支节点(Branch Node):在除去所有公共前缀后,十六进制串首个不同的字母用于构造分支节点。因此分支节点的 16 个子节点分别连接由对应字符开头的子串,例如图 1 中在去除公共前缀“2a”后,子串“3bd”和“56f”分别连接在分支节点的“3”和“5”处。特别的,如果一个十六进制串是其它串的一个前缀子串,即该串在扩展节点处会结束存储,那么扩展节点所

连接的分支节点将在 value 值中存储该串,比如图 1 中的“2a88d”串。由于分支节点具有 16 个子节点,因此 MPT 树减少了层高和存储空间。

叶子节点(Leaf Node):作为所有十六进制串存储的终点,叶子节点存储了除去公共前缀部分剩余的子串。因此叶子节点的 next_node 为一个用 null 表示的空节点。

根节点(Root Node):MHT 中根节点用于存储连接在根节点上左右子节点集合的散列值。MPT 树根节点的散列值也与 MHT 的计算方法相同,自底向上计算出所有节点连接子节点集合的散列值,并存在每个节点的 hash 字段中,直至根节点。不同的是 MPT 树根节点通常只连接一个扩展节点或分支节点。

这些节点类型的结合赋予了 MPT 树出色的灵活性和高效性,其能够在有限的内部存储空间中管理大量外部密钥。

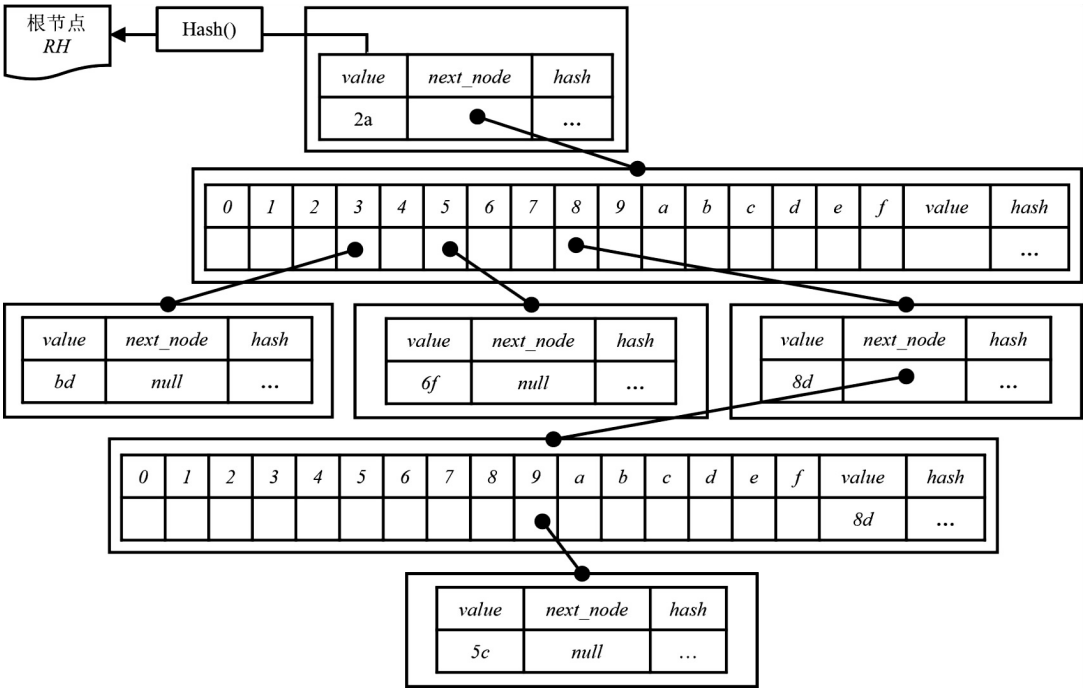

图 1　Merkle Patricia Tree 结构示意图

2　方案设计

2.1　系统模型

如图 2 所示,该文基于 MPT 树设计了一种 TPM 外部密钥的管理方案,可以高效地增加、撤销单一密钥并验证密钥的合法性。TPM 中仅存储了 MPT 树的根节点以节约 TPM 内部存储空间,而完整的 MPT 树则存储在外部的文件中。当 TPM 由主密钥派生出新的子密钥时,将其添加至 MPT 树中,并从下向上逐层按密钥插入路径更新节点的 hash 字段直至根节点。当 TPM 加载密钥时,密钥管理程序则需要沿着密钥验证路径判断密钥是否存在于 MPT 树中,同时依据路径上节点的 hash 字段,逐层重新计算根节点的散列值,并与存储在 TPM 内的根节点值对比,以验证密钥是否合法。需要撤销密钥时,直接验证密钥的搜索路径删除密钥对应的叶子节点,并更新根密钥的散列值。

二叉树密钥管理结构中,因密钥信息仅保存在叶子节点中,因而该类密钥管理的存在性证明需通过遍历寻找密钥对应的叶子节点,并生成由叶节点至根节点的验证路径。由于 MPT 树具有字典树的特性,因此 MPT 树只需依据密钥散列值和每个节点的信息即可生成由根节点至密钥对应叶节点的验证路径。同时,字典树特性使 MPT 树可以支持高效的数据插入、删除、更新等操作,因而文中方案能有效提升密钥管理和验证的效率。

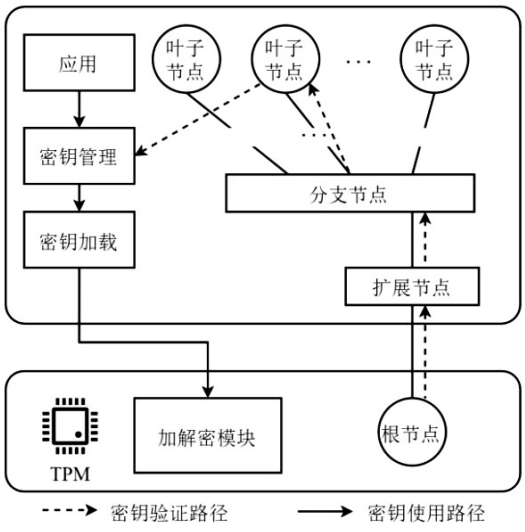

图 2　TPM 外部密钥管理系统模型

表 1 MPT 树密钥管理方案与现有方案对比

	节点类型	字典树特性	高效性
MPT	丰富	√	√
MHT	单一	×	×
CAT	单一	×	×

2.2 威胁模型与设计目标

针对以上系统模型,该文构造了以下几点安全假设和威胁模型:

假设 1 攻击者无法对 TPM 内部存储的任何数据发起篡改攻击,因此 TPM 内部存储的根节点值是安全的。

假设 2 未经身份认证的攻击者无法调用 TPM 的命令接口,亦无法干扰 TPM 的程序执行。

假设 3 选择的散列函数 Hash( )具有单向性与抗碰撞性特征。即给定一个密钥 key 的散列值 H,攻击者无法还原出 key 使得 Hash(key) = H。其次,攻击者难以找到另一个密钥 key′使得 Hash(key′) = Hash(key)。

威胁 1 攻击者视图恢复一个已经被密钥持有者撤销的外部密钥,并在 TPM 中重新加载使用。

威胁 2 攻击者意图篡改一个已经存在于 MPT 树中的密钥,使密钥持有者使用被篡改的密钥执行数据的加解密等安全操作。

基于以上假设和威胁,该文设计的基于 MPT 树的外部密钥管理方法制定了以下设计目标:

目标 1 安全性:能够抵御攻击者对 TPM 外部密钥完整性的破坏,保证外部密钥的安全增加、撤销和验证。

目标 2 高效性:在外部密钥数量规模变大时,仍能高效地增加、撤销和验证密钥。

目标 3 对 TPM 内外部存储空间占用较小。

目标 4 可恢复性:当攻击者对外部密钥存储树进行破坏时,密钥持有者可以根据自身持有的密钥快速恢复仍可以通过 TPM 校验的树结构。

2.3 用于外部密钥管理的 MPT 树

该文使用的可管理 TPM 外部密钥的 MPT 树由根节点、分支节点、扩展节点和叶子节点组成。其中根节点的值为 MPT 树所有节点聚合的散列值 RH,存储在 TPM 内部空间中,其计算方法类似于 MHT 的根节点计算过程。

选取一个散列函数 Hash( )和一个对称加密算法,所有的外部密钥都会被散列函数计算为散列值 hk_i = Hash(key_i), hk_i 位于 TPM 外存储空间的 MPT 树中,并根据密钥的增加、更新和撤销,操作对应的节点。同时,每个密钥将用对称加密算法加密后存储在应用程序中以带使用。MPT 每种节点的结构体如下所示:

```
typedef struct branch_node {
    node   children_node[16];
    byte    value[];
    byte    hash[];
}
typedef struct short_node {
    byte   value[];
    node   next_node;
    byte   hash[];
}
```

其中 branch_node 为分支节点,具有 16 个子节点 children_node[16]和一个 value 值。16 个子节点分别连接匹配完公共前缀后,以 0-9,a-f 开头的子节点。如果有密钥串散列值 hk 在当前分支节点前的扩展节点结束存储,则该分支节点的 value 值直接存储 hk。

而 short_node 可以作为叶子节点也可以作为扩展节点,其节点类型由 next_node 控制。当 next_node 连接一个分支节点时,该节点为扩展节点,其 value 存储经过该节点的所有密钥串的部分公共前缀。当 next_node 为空时,该节点为叶子节点,其 value 存储一个密钥串除去所有公共前缀剩余的子串。

同时,branch_node 和 short_node 中都存在一个字段 hash,与 MHT 类似,该字段用于存储叶子节点 value 对应的散列值,或者存储扩展节点与分支节点所连接子节点集合的散列值,用于校验整棵 MPT 树的完整性。

密钥新增:当新的外部密钥需要插入 MPT 树时,将新密钥摘要与已经存储在 MPT 树中密钥散列值进行公共前缀匹配,查找到可以添加数据的节点,进行密钥增加操作。该操作分为以下三种情况:

(1)如果当前根节点连接为空,则直接构造一个 value 值为 hk,next_node=Ø 的 short_node 节点作为叶子节点连接至根节点,并更新根节点的值为 Hash(hk)。

(2)如果查找到的节点类型是叶子节点,首先计算新密钥散列值与该叶子节点对应密钥散列值的公共前缀,并分为以下情况:

①如果公共前缀就等于叶子节点的 value 值,那么这两个密钥是一样的,说明该密钥已经存在于 MPT 树中,无需对 MPT 树进行额外的操作。

②如果公共前缀不完全匹配,那么就需要把公共前缀提取出来,并构造一个 short_node 节点作为扩展节点连接至根节点,并由扩展节点的 next_node 连接一个新构造的分支节点。同时,分别将新密钥散列值与叶子节点对应的密钥散列值去除公共前缀的部分,

构造两个新的叶子节点。其后,根据两个密钥串散列值除去公共前缀部分的首字母,将两个叶子节点连接在分支节点相应的 children_node 处。最后,逐层向上更新新密钥串插入路径上所有节点存储的 hash 字段,即分支节点的 hash 字段值更新为所连接子节点集合的散列值,扩展节点的 hash 字段更新为所连接分支节点的散列值,直至根节点。

③如果公共前缀匹配的长度为 0,则无法构造一个包含公共前缀的扩展节点,那么直接构造一个分支节点连接至根节点。根据密钥散列值的首字母,将密钥对应的叶子节点作为子节点分别连接至分支节点的 children_node 处。同时,更新所有相关节点的 hash 字段,直至根节点。

(3)如果查找到的节点类型是扩展节点,那么与叶子节点的情况②类似,将公共前缀部分提取成一个新的扩展节点,并连接一个分支节点,像分支节点对应处插入新的密钥串散列值和旧扩展节点去除公共前缀的剩余部分。

(4)如果查找到的节点类型是分支节点,那么直接前往对应的子节点 children_node 继续执行插入操作,直至可以在下层分支节点的 children_node 插入新生成的叶子节点。

上述过程如算法 1 所示,通过对根节点调用 add_node(root_node,hk)并经过插入路径的迭代,最终可以完成新增 hk 和根节点散列值的更新。

算法 1:add_node()用于向 MPT 树新增单个密钥串散列值。

输入:当前遍历至的节点 node,待插入的密钥散列值 byte[] hk。

输出:修改后的当前节点 node。

(1) if node 节点为空 then

(2) 构造 short_node(hk,null,Hash(hk)),并返回

(3) else

(4) if 节点类型为 short_node then

(5) 取 node. value 和 hk 的公共前缀

(6) if 没有公共前缀 then

(7) 构造 branch_node,并连接原节点和新密钥节点,后返回

(8) if 公共前缀长度小于 node. value 长度 then

(9) 将公共前缀构造 short_node,并构造子节点 branch_node,以连接原节点和新密钥节点,后返回

(10) if 公共前缀一致,且 node 是扩展节点 then

(11) 迭代执行 add_node()

(12) End if

(13) else

(14) 前往 branch_node 对应子节点,迭代执行 add_node()

(15) End if

(16) End if

密钥撤销:从根节点开始,通过前缀匹配向下查找,直到找到密钥散列值对应的节点,并删除该节点。如果查找到分支节点仅有两个节点,并且需要删除其中一个节点时,则将该分支节点的父节点由扩展节点转变为叶子节点,并存储两个节点中需要保留的密钥散列值。具体过程如算法 2 所示,调用 delete_node(root_node,hk)可完成密钥串的撤销和根节点散列值的更新。

算法 2:delete_node()用于在 MPT 树中查找并删除单个密钥串散列值。

输入:当前遍历至的节点 node,待删除的密钥散列值 byte[] hk。

输出:修改后的当前节点 node。

(1) if node 是 branch_node then

(2) if branch_node 对应位置连接叶子节点,且叶子节点与 hk 值一致 then

(3) 直接删除叶子节点

(4) else

(5) 前往 branch_node 对应连接的扩展节点,迭代执行 delete_node()

(6) End if

(7) End if

密钥验证:如果已知一个根节点的散列值 RH,在已知整棵 MPT 的情况下,任何密钥使用者都可以提供某个密钥串的存在性证明。密钥使用者只需要提供一个从根节点开始,通过对密钥散列值进行公共前缀匹配到达叶子节点的路径即可。而当攻击者试图使用一个 MPT 树不存的伪造密钥串时,攻击者无法提供一棵与 TPM 内存储的根节点散列值相同,且包含伪造密钥串搜索路径的 MPT 树。因此可以通过 MPT 树保证密钥存在的安全性。通过调用算法 3,当 verify_node(root_node, root_hash, hk)返回 True 时,密钥串有效,否则无效。

算法 3:verify_node()用于验证单个密钥串是否有效。

输入:当前遍历至的节点 node, byte[] hash,待验证密钥 byte[] hk。

输出:密钥校验值 is_valid。

(1) if node 为 branch_node,且节点散列值校验通过 then

(2) if branch_node 对应位置没有子节点 then

(3) is_valid = False

(4) else

(5) 前往 branch_node 对应子节点,迭代执行 verify_node()

(6) End if

(7) else if node 为叶子节点 then

(8) if node. value 与 hk 一致,且散列值校验通过 then

(9) is_valid = True

(10) else

(11) is_valid = False

(12) End if

(13) else if node 为扩展节点,且散列值校验通过 then

(14) 前往子节点,迭代执行 verify_node()

(15) else
(16) is_valid = False
(17) End if

MPT 恢复:基于 MHT 等二叉树的外部密钥管理方案中,所有的密钥散列值均存在叶子节点中,因此当二叉树被攻击者破坏时,如果密钥持有者不能明确所有密钥与原叶子节点的对应关系则难以恢复出仍可以通过 TPM 校验的密钥管理树。但是由于 MPT 树具有字典树的特性,因此每个密钥在其中的存储路径是固定的,所以即使攻击者破坏了部分或全部外部密钥管理的 MPT 树,只要密钥持有者仍保有这些密钥的副本,那么其可以容易地恢复出原始的密钥管理树以供 TPM 使用。

3 安全性证明

存在性证明:假设有一棵 MPT 树,从根节点到叶子节点的路径为 p,该路径上所有节点 d_i 的 value 值可构成一个密钥 key 的散列值。同时,该树每个节点的散列值都是其子节点散列值的 Hash,如式 1 所示。

$$\begin{cases} p = \{d_i \mid i = 1,2,\cdots,m\} \\ \text{Hash(key)} = \{d_i.\text{value} \mid d_i \in p\} \\ d_{i-1}.\text{children} = d_i \\ d_{i-1}.\text{hash} = \text{Hash}(d_i) \end{cases} \tag{1}$$

那么当一个密钥 key 存在于该树中时,必然存在一个到叶子节点的路径 p,使得从叶子节点 d_m 依据 p 逐层向上更新节点散列值直至根节点后,满足新散列值等于根节点散列值 RH,即式 2 所示。

$$\exists p \Rightarrow \text{RH} = \text{Hash}(\cdots\text{Hash}(d_{m-1},\text{Hash}(d_m))) \tag{2}$$

如果不存在符合条件的路径 p,则密钥 key 不存在于该树中。

根据威胁模型及存在性证明,本节具体分析被撤销的外部密钥是否存在安全隐患。

(1)若攻击者获取了被密钥持有者已经在 TPM 撤销的密钥副本,攻击者试图在 MPT 树中恢复该密钥的使用。假设在 TPM 中撤销了该密钥后,存储在 TPM 中根节点值为 RH,那么攻击者需要将密钥副本重新插入外部存储的 MPT 树,并改造 MPT 树使得新树的根节点值 RH′与 RH 相同。由于寻找一棵包含密钥副本,同时根节点值 RH′= RH 的树是困难的,所以攻击者无法恢复已经被密钥持有者在 TPM 中撤销的密钥。

(2)若攻击者试图篡改一个已经存在于 MPT 树中的密钥,使得 TPM 可以加载篡改后的密钥进行数据处理。假设当前存储在 TPM 中的根节点值为 RH,那么攻击者需要选择一个密钥使得该密钥经过散列函数操作后与原密钥具有相同的散列值,即相同的搜索路径。或者攻击者需要构建一棵包含篡改密钥并且根节点值与 RH 相同的 MPT 树。但是根据散列函数的单向性与抗碰撞性特征,对于攻击者来说是难以寻找到符合要求的密钥或者 MPT 来替换存储在 TPM 外部的信息,所以该方案能达到安全目标。

4 性能分析

实验运行在树莓派 4B 开发板上,并配备了实体 TPM 芯片 NationZ Z32H320TC。树莓派 4B 采用四核 64 位 ARM Cortex-A72 架构的 Broadcom BCM2711 处理器,主频为 1.5 GHz,内存为 4 GB。操作系统使用 Linux 内核版本为 4.14.98-v7 的 Raspbian OS。TPM 芯片 Z32H320TC 遵循 TPM 2.0 标准[16]并兼容 TCM 规范[17-18],通过 SPI 接口与树莓派连接,并支持 SHA256、SM3、AES256、SMS4 等多种安全算法,可信软件栈为开源项目 tpm2-tools[19]。

本节实验通过在不同密钥数量规模下,基于 MPT 树的 TPM 外部密钥管理方案新增、撤销和验证单个密钥的时间开销和总存储节点数量,并与基于 Merkle 树和 CAT 的方案进行比较,来说明该方案满足设计目标高效性和较小的空间占用。其中,密钥数量规模分别设置为 2^n, n =1,3,5,7,9,11,13。同时,文中散列函数选择 SHA256,对称加密算法则为 AES256。

图 3 表现了 MPT 树和 MHT 两种密钥管理方式中,新增单个密钥的时间开销与已有密钥规模的关系。

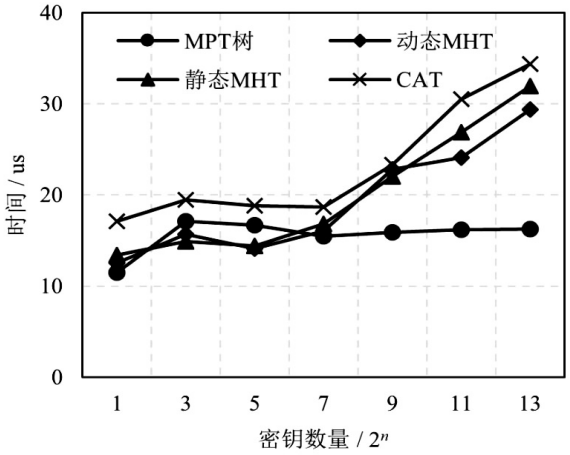

图 3 密钥新增时间开销比较

由于 MHT 中插入一个包含新密钥的叶子节点后,需要自底向上更新每一层相关节点的散列值直至根节点。因此,随着密钥规模变大,MHT 的层数也逐渐变多,而需要计算散列函数的次数与 MHT 层数相同。而在 CAT 方案中,由于左叶节点保存密钥散列值,而右叶节保存密钥变色龙散列值,因此相同密钥数量情况下,CAT 层数比 MHT 层数多 1。在 MPT 树中,散列函数的计算次数与密钥的插入路径相关。由于

MPT 树具有扩展节点和 16 个子节点的分支节点,因此密钥插入 MPT 树时所经过的节点数相较于 MHT 和 CAT 的层数大大减少。实验结果表明,MPT 树中不同密钥数量规模下,新增一个密钥耗时较为稳定,约为 15.85 μs。而当 n >9 时,基于 MHT 树的方案增加一个密钥的耗时逐渐增大,例如当 n =9 时,动态 MHT 耗时为 22.83 μs,而静态 MHT 耗时为 22.03 μs;当 n =13 时,动态 MHT 耗时为29.35 μs,静态 MHT 耗时为 31.96 μs。同时,CAT 方案下时间消耗从 17.1 μs 增长至 34.35 μs。所以,MPT 树新增单个密钥消耗的时间与密钥规模无关,而其余方案效率均随密钥增多而降低。

同理,为了维护二叉树的平衡性,在 MHT 和 CAT 中撤销一个密钥的方式为,将该密钥所在的叶子节点值更新成原密钥联合撤销标志的散列值,并逐层重计算直至根节点的散列值。而 MPT 树只需根据搜索路径直接删除密钥对应的叶子节点,并更新路径上所有节点的散列值,即可完成密钥的撤销。如图 4 所示,在不同密钥数量规模下,MPT 树撤销一个密钥耗时约为 3.21 μs;而当 n =5 时,动态 MHT 用时 11.46 μs,静态 MHT 需用时 18.33 μs,CAT 用时 17.21 μs;当 n =13 时,动态 MHT 用时 33.15 μs,静态 MHT 需要 34.21 μs,CAT 则需 35.42 μs。与密钥新增效率相似,MPT 树撤销密钥的效率在密钥规模变大时仍较为稳定,而其他方案逐渐降低。

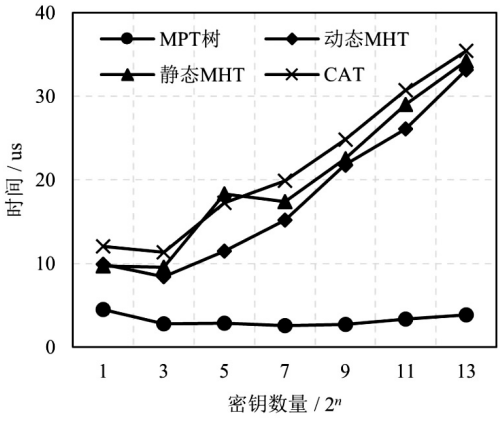

图 4　密钥撤销时间开销比较

密钥验证时,MHT 和 CAT 需要根据密钥所在的叶子节点位置生成验证路径,并根据路径计算出新的根节点散列值与 TPM 内存储的值进行比较。而 MPT 树具有字典树的特性,因此可以直接依据密钥的搜索路径寻找相应叶子节点后,并重新计算路径上所有节点至根节点的散列值,与 TPM 内的根节点比较后判断密钥是否有效。图 5 表明,不同密钥数量规模中,MPT 树验证单密钥有效性耗时约为 8.25 μs;而在 MHT 和 CAT 中,随密钥数量规模变大,验证单密钥的时间也逐渐增大,当 n =7 时,动态 MHT 耗时为 16.71 μs,静态 MHT 耗时为 16.41 μs,CAT 用时 18.81 μs;当 n = 13 时,动态 MHT 用时 30.77 μs,静态 MHT 耗时为 30.88 μs,CAT 为 32.49 μs。可得,MPT 树密钥验证效率不受密钥规模影响,而其他方案在密钥规模增大时效率随之降低。

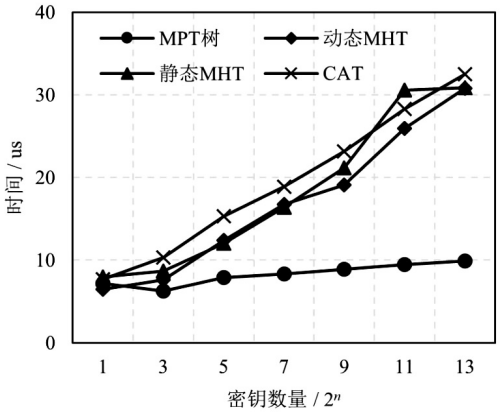

图 5　密钥验证时间开销比较

在树的大小与密钥数量规模方面,由于 MHT 是一棵二叉树,因此当 MHT 有 2^n 个叶子节点时,其总节点数为 $2^{n+1}-1$。而 MPT 树具有包含 16 个子节点的分支节点,因此 MPT 树的存储规模大大降低。图 6 显示了不同密钥数量下,MPT 树与 MHT 所具有的节点数量的变化。

实验结果表明,当 n = 11 时,MPT 树需要 2 713 个节点,而在该密钥规模下动态 MHT 和静态 MHT 均为满二叉树,均需 4 095 个节点;当 n = 13 时,MPT 树节点数为 11 431 个,而动态和静态 MHT 为16 383个。由于 CAT 中左右两个叶节点表示了一个同密钥的有效性,因此 CAT 占用空间最大,当 n =13 时,CAT 需要使用 32 767 个节点。

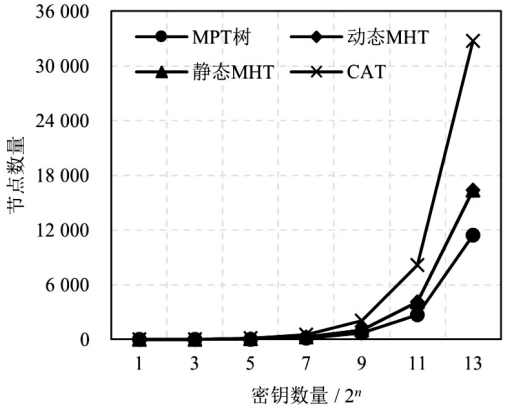

图 6　存储节点数量比较

综上,在大规模 TPM 外部密钥管理场景下,基于 MPT 树的方案相较于 MHT 和 CAT 具有更稳定的性能和更小的内存占用空间。

5 结束语

该文提出了基于 MPT 树的 TPM 外部密钥管理方案,并在树莓派 4B 开发板上结合 TPM 芯片 Z32H330TC 实现了对单一密钥的新增、撤销和验证等功能。借助 MPT 树多种节点的灵活配合及其字典树的特性,该方案可以在较小的存储空间中高效地管理大规模密钥。随着近些年可信执行环境(Trusted Execution Environment,TEE)技术[20]的新起,该方案亦可用于对 TEE 外部加密文件的版本管理等功能[21-22],有效保证 TEE 处理外部文件的完整性[23]。

参考文献:

[1] 冯登国,刘敬彬,秦 宇,等. 创新发展中的可信计算理论与技术[J]. 中国科学:信息科学,2020,50(8):1127-1147.

[2] JACOB H N,WERLING C,BUHREN R,et al. faulTPM:exposing AMD fTPMs′ deepest secrets[C]//2023 IEEE 8th European symposium on security and privacy (EuroS&P). Delft:IEEE,2023:1128-1142.

[3] 高 爽. 主动配电网信息安全防护关键技术研究[D]. 北京:华北电力大学,2019.

[4] 张 磊. 电力物联网下的采集终端可信计算技术研究[J]. 电子元器件与信息技术,2019,3(12):113-116.

[5] KASHIF U A,MEMON Z A,GHANGHRO S A,et al. Centralized accessibility of VM for distributed trusted cloud computing[C]//2023 4th international conference on computing,mathematics and engineering technologies (iCoMET). Sukkur:IEEE,2023:1-6.

[6] WANG J,XIAO F,HUANG J,et al. A security-enhanced vTPM 2.0 for cloud computing[C]//Information and communications security:19th international conference, ICICS 2017. Beijing:Springr,2018:557-569.

[7] 田洪亮. 云环境下基于可信硬件的数据机密性保护技术研究[D]. 北京:清华大学,2019.

[8] KATZENBEISSER S,KURSAWE K,STUMPF F. Revocation of TPM keys[C]//Proceedings of the 2nd international conference on trusted computing. Oxford:Springer-Verlag,2009:120-132.

[9] 余发江,陈宇驰,张焕国. 具备撤销单一密钥功能的 TPM 动态密钥管理机制[J]. 清华大学学报:自然科学版,2020,60(6):464-473.

[10] 余发江,申 淦,张焕国. 基于 Merkle 树的 TPM 单一密钥撤销[J]. 电子学报,2023,51(4):792-800.

[11] WU H,PENG Z,GUO S,et al. VQL:efficient and verifiable cloud query services for blockchain systems[J]. IEEE Transactions on Parallel and Distributed Systems,2021,33(6):1393-1406.

[12] LU Z,WANG Q,QU G,et al. A blockchain-based privacy-preserving authentication scheme for VANETs[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems,2019,27(12):2792-2801.

[13] 国民技术股份有限公司. Z32H330TC 可信计算芯片[EB/OL]. [2023-11-10]. https://www.nationstech.com/Z32H330TC/.

[14] LIU X,CHANG Y,ZHANG H,et al. Power data integrity verification method based on chameleon authentication tree algorithm and missing tendency value[J]. Energy Harvesting and Systems,2024,11(1):20230067.

[15] TIAN G,WEI J,MIAO M,et al. Blockchain-based compact verifiable data streaming with self-auditing[J]. IEEE Transactions on Dependable and Secure Computing,2023,Early Access:1-14.

[16] ZHANG Q,ZHAO S,QIN Y,et al. Formal analysis of TPM2.0 key management APIs[J]. Chinese Science Bulletin,2014,59(32):4210-4224.

[17] 王 冠,袁华浩. 基于可信根服务器的虚拟 TCM 密钥管理功能研究[J]. 信息网络安全,2016(4):17-22.

[18] 王 浩,张明利,王鹏彰. 基于 TCM 的安全增强机制在电力控制保护设备上的应用[J]. 工业控制计算机,2022,35(2):39-40.

[19] TPM2 Software Community. Linux TPM2 & TSS2 software[EB/OL]. [2023-11-10]. https://github.com/tpm2-software.

[20] EKBERG J E,KOSTIAINEN K,ASOKAN N. Trusted execution environments on mobile devices[C]//Proceedings of the 2013 ACM SIGSAC conference on computer & communications security. Berlin:ACM,2013:1497-1498.

[21] NIU J,PENG W,ZHANG X,et al. Narrator:secure and practical state continuity for trusted execution in the cloud[C]//Proceedings of the 2022 ACM SIGSAC conference on computer and communications security. Los Angeles:ACM,2022:2385-2399.

[22] VAN DIJK M,RHODES J,SARMENTA L F G,et al. Offline untrusted storage with immediate detection of forking and replay attacks[C]//Proceedings of the 2007 ACM workshop on scalable trusted computing. Alexandria Virginia:ACM,2007:41-48.

[23] LU D,SHI M,MA X,et al. Smaug:a TEE-assisted secured SQLite for embedded systems[J]. IEEE Transactions on Dependable and Secure Computing,2023,20(5):3617-3635.