软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(8):2287−2313 [doi: 10.13328/j.cnki.jos.005761]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 Duplication Authority 的 TPM2.0 密钥迁移协议*

谭 良[1,2], 宋 敏[1]

[1](四川师范大学 计算机科学学院,四川 成都 610101)

[2](中国科学院 计算技术研究所,北京 100190)

通讯作者: 谭良, E-mail: tanliang2008cn@126.com

摘 要: 《TPM-Rev-2.0-Part-1-Architecture-01.38》国际标准允许用户基于密钥复制接口设计迁移协议,此复制接口通过 innerwrap 和 outerwrap 为密钥迁移提供机密性、完整性和认证性.但研究发现,基于该复制接口来设计密钥迁移协议存在 3 个问题:其一是缺少交互双方 TPM 的相互认证,会导致密钥能够在敌手和 TPM 间迁移;其二是当迁移密钥的属性 *encryptedDuplication*=0 且新父密钥的句柄 *newParentHandle=TPM_RH_NULL* 时,复制接口不能实施 innerwrap 和 outerwrap,迁移密钥将以明文传输而造成泄露;其三当新父密钥是对称密钥时,innerwrap 中的对称加密密钥以及 outerwrap 中的密钥种子如何在源 TPM 与目标 TPM 之间安全交换,《TPM-Rev-2.0-Part-1-Architecture-01.38》并没有给出具体的解决办法.针对上述问题,提出了基于 Duplication Authority 的密钥迁移协议.该协议以 Duplication Authority 为认证和控制中心,将密钥迁移过程分为初始化阶段、认证和属性获取阶段以及控制和执行阶段.Duplication Authority 通过判定密钥的复制属性和类型、新父密钥的密钥类型和句柄类型来决定迁移流程.考虑了各种合理的属性组合,共设计了 12 种迁移流程.最后对该协议进行了安全分析和实验验证,结果显示,该协议不仅完全满足《TPM-Rev-2.0-Part-1-Architecture-01.38》规范,而且可以保证迁移密钥的完整性、机密性和认证性.

关键词: 可信计算;可信平台模块;密钥层次结构;密钥复制;密钥迁移

中图法分类号: TP309

中文引用格式: 谭良,宋敏.基于 Duplication Authority 的 TPM2.0 密钥迁移协议.软件学报,2019,30(8):2287−2313. http://www.jos.org.cn/1000-9825/5761.htm

英文引用格式: Tan L, Song M. TPM2.0 key migration-protocol based on duplication authority. Ruan Jian Xue Bao/Journal of Software, 2019,30(8):2287−2313 (in Chinese). http://www.jos.org.cn/1000-9825/5761.htm

TPM2.0 Key Migration-protocol Based on Duplication Authority

TAN Liang[1,2], SONG Min[1]

[1](School of Computer Science, Sichuan Normal University, Chengdu 610101, China)

[2](Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: International Standard "TPM-Rev-2.0-Part-1-Architecture-01.38" allows users to design a migration protocol based on the duplication interface which provides confidentiality, integrity, and authentication for key migration by innerwrap and outerwrap. However, the researchs have found that there are three problems, one is the lack of mutual authentication between the two parties of the interaction

* 基金项目: 国家自然科学基金(61373162); 四川省科技厅重点研发项目(19ZDYF1082); 可视化计算与虚拟现实四川省重点实验室项目(KJ201402)

Foundation item: National Natural Science Foundation of China (61373162); Key Research and Development Project of Science and Technology Bureau, Sichuan Province (2014GZ0007); Project of Sichuan Provincial Key Laboratory of Visual Computing and Virtual Reality (KJ201402)

本文由"面向自主安全可控的可信计算"专题特约编辑张焕国教授推荐.

收稿时间: 2018-05-25; 修改时间: 2018-09-21; 采用时间: 2018-12-13; jos 在线出版时间: 2019-03-28

CNKI 网络优先出版: 2019-03-29 09:16:26, http://kns.cnki.net/kcms/detail/11.2560.TP.20190329.0915.012.html

TPM, which results in the transfer of keys between adversaries and TPM. The other is that when the property of the duplication key *encryptedDuplication*=0 and the new parent key handle *newParentHandle=TPM_RH_NULL*, the duplication interface can not implement innerwrap and outerwrap, the migration key will be transmitted in clear text. The third is that how are the symmetric encryption key in innerwrap and the seed in outerwrap exchanged securely between the source TPM and the target TPM when the new parent key is a symmetric key. "TPM-Rev-2.0-Part-1-Architecture-01.38" did not give a specific solution. In order to solve the above problems, this study proposes a transfer protocol based on Duplication Authority which uses as the authentication and control center, and the protocol is divided into three phases: initialization phase, authentication and attribute acquisition phase, and control and execution phase. Duplication Authority determines the migration process by the migration key's duplication attributes and types, the key type and handle type of the new parent key. A combination of various compliance attributes was considered and a total of 12 migration processes were designed. Finally, the protocol was analyzed by security and experiments, the results show that the protocol is not only fully compliant with the "TPM-Rev-2.0- Part-1-Architecture-01.38" specification but also meets the requirements of integrity, confidentiality, and authenticity for key migration.

Key words: trusted computing; trusted platform module; key hierarchy; key duplication; key migration

可信计算技术的基本思想是:在通用计算平台上嵌入一个防篡改的硬件可信安全芯片,利用芯片的安全特性保证系统按照预期的行为执行,从根本上提高终端的安全性[1].TPM(trusted platform module)[2]是国际广泛使用的是符合可信平台模块标准的安全芯片,具有密码学功能和受保护的存储空间,能够为可信计算平台提供密钥管理、平台数据保护、完整性存储与报告、身份标识等功能[2−4].

密钥管理是 TPM 非常重要的功能,它是 TPM 能够有效地提供其他各项功能的前提和基础.为了满足密钥的安全存储、分发和备份,TPM 采用层次型的存储保护体系,并提供密钥迁移(复制)接口.TPM1.1[5]规范中定义的密钥迁移接口是 *TPM_AuthorizeMigrationKey*,*TPM_CreateMigratedBlob* 和 *TPM_convertMigratedBlob* 等;TPM1.2[6]规范定义的密钥迁移接口是 *TPM_AuthorizeMigratinKey*(),*TPM_CMK_ApproveMA*(),*TPM_CMK_CreateKey*(),*TPM_CMK_CreateTicket*(),*TPM_CMK_CreateBlob*()和 *TPM_CMK_ConvertMigration*()等;TPM2.0[7]规范定义的密钥复制接口是 *TPM2_Duplicate*()和 *TPM2_Import*().通常,用户或上层应用可以通过以上接口设计密钥迁移协议,将源 TPM 中的密钥迁移到目的 TPM 中,以实现 TPM 芯片间密钥的共享.为了保证整个迁移过程的安全,需要提供机密性、完整性和认证性.

然而,TPM2.0 的密钥迁移协议设计会更加复杂.一方面,由于 TPM2.0 已支持对称密钥对象,使得在设计密钥迁移协议时需要考虑更多的迁移组合.因为迁移密钥既可以是对称密钥也可以是非对称密钥,而新父密钥也可以是对称密钥或非对称密钥,不同的迁移需求组合在协议设计过程中需要进行不同的设计.另一方面,TPM2.0 是通过迁移密钥对象的复制属性(*fixedTPM*,*fixedParen*,*encryptedDuplication*)和新父密钥的 *newParentHandle* 类型来决定迁移方式和迁移过程.不同的组合,其迁移方式和迁移过程是不同的,特别是在密钥复制过程中是否进行 *innerwrap* 和 *outerwrap*,是保证密钥迁移的机密性、完整性和认证性的关键.通过进一步研究发现,基于密钥复制接口的密钥迁移协议至少存在 3 个问题:一是缺少交互双方 TPM 的相互认证,会导致密钥能够在敌手和 TPM 间迁移;二是当迁移密钥的属性 *encryptedDuplication*=0 且新父密钥的句柄 *newParentHandle*=*TPM_RH_NULL* 时,复制接口不能实施 *innerwrap* 和 *outerwrap*,迁移密钥将以明文传输而造成泄露;三是当新父密钥是对称密钥时,*innerwrap* 中的对称加密密钥以及 *outerwrap* 中的密钥种子如何在源 TPM 与目标 TPM 之间安全交换,《TPM-Rev-2.0-Part-1-Architecture-01.38》并没有给出具体的解决办法.

为此,本文提出了基于 Duplication Authority 的密钥迁移协议.该协议以 Duplication Authority 为认证和控制中心,通过判定迁移密钥的复制属性、新父密钥的密钥类型和句柄类型来决定迁移流程.最后,对该协议进行了安全分析和实验验证.

本文第 1 节简介 TPM 密钥迁移的相关背景知识.第 2 节介绍现有的 TPM 密钥迁移协议并指出其存在的问题.第 3 节详述本文提出的基于 Duplication Authority 的密钥迁移协议.第 4 节对提出的协议进行分析.第 5 节是实验验证和性能对比分析.第 6 节介绍可信安全芯片密钥迁移的相关研究工作.第 7 节总结全文.

1 背景知识

本节从 TPM 2.0 的密钥类型和结构、密钥对象管理保护体系、密钥迁移类接口这 3 个方面介绍本文的背景知识.

1.1 TPM2.0的密钥类型和结构

相比于 TPM1.2,TPM2.0 的密钥类型和结构发生了较大的变化.下面我们对此进行详细的介绍.

1.1.1 TPM2.0 的密钥类型

1. 按照密钥功能组合分类

TPM2.0 中密钥对象的基本属性包括:

- Restricted Attribute:专用属性,表明该密钥只能对特定对象进行操作.
- Sign Attribute:签名属性,表明该密钥对象是否可以用于签名.
- Decrypt Attribute:机密属性,表明该对象是否可以用于加解密.

根据以上 3 个属性,TPM2.0 将密钥对象分为 8 类,见表 1.

Table 1 Key classification of TPM 2.0

表 1 TPM2.0 的密钥分类

Sign	Decrypt	Restricted	功能
0	0	0	表明该密钥对象是一个可以用 *TPM2_Unseal*()接口解封的数据块
0	0	1	不允许这样设置.无此类密钥对象
0	1	0	该密钥可用于任何需要加解密的操作.但该密钥不能是存储密钥
0	1	1	无论是对称密钥还是非对称密钥,拥有此属性表明该密钥是一个存储父密钥.该密钥只用于默认的方案和模式
1	0	0	表明该密钥可用于任何签名操作包括 quote,certify 和 sign.对于拥有 *TPM_ALG_KEYEDHASH* 属性的对象,可以用该密钥生成 HMAC 消息码
1	0	1	表明该密钥只能对 TPM 产生的任何消息摘要进行签名,包括 Quoting,certifying 和 signing.该密钥只用于默认的方案和模式
1	1	0	一种通用的密钥,只要对象相关的密钥算法兼容,就可以使用该密钥进行加密和签名.但该密钥不能是存储密钥
1	1	1	目前暂无

这 8 类密钥与 TPM1.2 中 7 种类型密钥(签注密钥(endorsement key,简称 EK)、存储密钥(storage key,简称 SK)、身份认证密钥(attestation identity key,简称 AIK)、签名密钥(signing key)、绑定密钥(binding key)、继承密钥(legacy key)和验证密钥(authentication keys))的对应关系见表 2.

Table 2 Key type correspondence between TPM1.2 and TPM2.0

表 2 TPM1.2 与 TPM2.0 的密钥类型对应关系

TPM1.2 中的密钥类型	Sign	Decrypt	Restricted	说明
TPM_KEY_SIGNING	1	0	0	在 TPM1.2 中,该类型密钥只用于默认的方案和模式.在 TPM2.0 中,该方案在命令定义,既可以对 TPM 产生的数据签名,又可以对外部数据签名
TPM_KEY_STORAGE	0	1	1	该类密钥 TPM1.2 和 TPM2.0 用法一致
TPM_KEY_IDENTITY	1	0	1	在 TPM1.2 中,此类密钥受到高度限制,例如,不能对非 TPM 产生的数据签名.在 TPM2.0 中,此类密钥不仅可以对 TPM 产生的数据进行签名,而且还可用于其他默认的方案和模式
TPM_KEY_AUTHCHANGE	–	–	–	TPM2.0 无此类密钥,TPM1.2 现已弃用
TPM_KEY_BAND	0	1	0	此类密钥与 TPM1.2 中的绑定密钥基本一致.在 TPM1.2 使用 *TPM_Unbind*()的地方,TPM2.0 使用 *TPM2_RAS_decrypt*()
TPM_KEY_LEGACY	1	1	0	TPM1.2 和 TPM2.0 用法一致,此类密钥的使用唯一受到密钥体系特性的限制
TPM_KEY_MIGRATE	0	1	1	TPM1.2 和 TPM2.0 的用法基本一致,如果新父密钥允许,源存储密钥可以是重新封装迁移的对象
Sealed data	1	1	1	包含用户定义的数据块

由此可见,TPM2.0的密钥类型更精确,其中,对应于TPM1.2中的*TPM_KEY_SIGNING*,*TPM_KEY_IDENTITY*这两类密钥功能略有增加.

2. 按照密钥复制属性分类

TPM2.0 中,密钥对象除了基本属性外,还包括一些其他属性,如 *fixedTPM* 和 *fixedParent*,*stclear*,*sensitiveDataOrigin*,*userWithAuth*,*adminWithPolicy*,*noDA* 和 *encryptedDuplication*.根据 *fixedTPM* 和 *fixedParent* 的属性组合,可以将密钥对象分为可复制密钥和不可复制密钥.如表 3 所示.本文所指的可复制密钥,就是指 *fixedTPM*=0 和 *fixedParent*=0 的密钥.

Table 3 Key classification for duplication

表 3 密钥可复制分类表

fixedTPM	*fixedParent*	描述
0	0	密钥可以被复制
0	1	密钥跟随父密钥复制
1	0	无此类组合密钥
1	1	不可复制密钥

1.1.2 TPM2.0 的密钥结构

TPM2.0 中,密钥对象的基本结构包括 3 个域:*PublicArea*,*SensitiveArea* 或 PrivateArea,对 TPM2.0 内部的任意密钥 *k*,表示为 *k*=(*PublicArea*,*SensitiveArea*);外部的任意密钥,通常表示为 *k*=(*PublicArea*,*PrivateArea*),其中,

- *Public Area*

1) *type*:密钥类型.
2) *nameAlg*:此密钥支持的密码算法.
3) *objectAttributes*:密钥属性,包括功能(*Sign*,*Decrypt* 和 *Restricted*)、授权(*userWithAuth*,*adminWithPolicy* 和 *noDA*)、复制(*fixedTPM*,*fixedParent* 和 *encryptedDuplication*)、生成方式(*sensitiveDataOrigin*)以及重置(*stclear*).
4) *authPolicy*:授权策略.
5) *parameters*:此类密码算法的参数.
6) *unique*:非对称密钥此项代表公钥;对称密钥此项代表其 *SensitiveArea* 的摘要值.

- *Sensitive Area*

1) *sensitiveType*:敏感数据类型.
2) *authValue*:授权值.
3) *seedValue*:对于对称的和非对称的存储密钥,由该值产生保护 child 对象的密钥.对于非对称密钥的非存储密钥当前无用.对于其他对象,该值与 *sensitive* 一起 HASH 产生 *unique* 摘要值.
4) *sensitive*:敏感参数值.对于非对称密钥,此值表示私钥:对于对称密钥,此值就是 *key*;对于消息码,此值就是 *key*;对于数据对象,此值就是敏感数据.

- *Private Area*

1) *encypted sensitive area*:此密文是由父密钥的 *seedValue* 产生的 *key* 对 *sensitive area* 加密的值.
2) HMAC 消息码:此消息码是由父密钥的 *seedValue* 产生的 *key* 对 *sensitive area* 进行 HMAC 的值.

在创建不同类型的对象时,可以调用 3 个接口:*TPM2_CreatePrimary*(),*TPM2_Create*() 和 *TPM2_CreateLoaded*().创建的对象类型依赖于输入参数 *ParentHandle* 的类型.函数执行成功会返回 *PublicArea* 和 *SensitiveArea*,前两个接口还会返回 *TPMS_CREATION_DATA* 类的 *creationData*.

1.2 TPM2.0密钥对象管理存储保护体系

TPM2.0 可通过 *TPM2_CreatePrimary*,*TPM2_Create* 和 *TPM2_CreateLoaded* 生成种子密钥对象、普通密钥对象和派生密钥对象,所有的密钥对象形成一棵密钥树,其中,种子密钥对象一般作为根密钥保护所在层次的子

密钥,如图 1 所示.在此密钥树中,TPM2.0 将密钥对象分为可复制密钥对象、可跟随父密钥复制密钥对象以及不可复制密钥对象.不可复制密钥对象只能与原 TPM 芯片绑定,不能被复制;而可复制密钥对象可以复制到其他 TPM 中使用,而可跟随父密钥复制密钥对象的存在使得在密钥树中一次能复制一棵子树,比 TPM1.2 中的密钥迁移更灵活,效率更高.

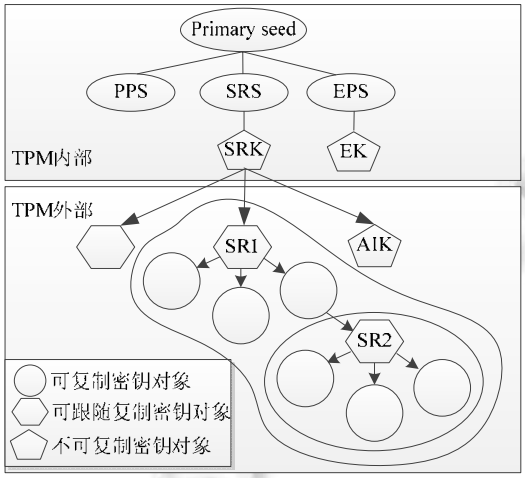

Fig.1 TPM2.0 key protect tree

图 1 TPM2.0 的密钥树

在 TPM2.0 密钥树中,存储父密钥采用对称加密方法保护孩子密钥,加密密钥由父密钥的密钥种子 *seedValue* 产生,加密算法由父密钥的 *nameAlg* 指定.这一点与 TPM1.2 中用非对称密钥的私钥对孩子的私钥进行加密保护不同,因此在 TPM2.0 的密钥树中,无论是对称密钥还是非对称密钥,只要是存储密钥,都可以作为父节点.如图 2 所示.

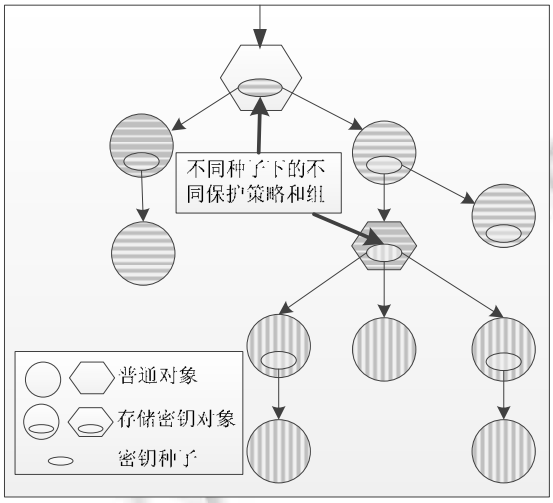

Fig.2 Key hierarchy protection model of TPM2.0

图 2 TPM2.0 的密钥层次保护模型

1.3 TPM2.0密钥对象复制接口

TPM2.0 完整的密钥复制流程是将源 TPM 的密钥对象复制到目标 TPM,因此,密钥复制应该包括两个接口:

其一是复制数据的生成;其二是复制数据的加载.

- 接口 1:复制数据生成接口:

TPM2_Duplication(*objectHandle*,*newParentHandle*,*encrptionKeyIn*,*symmetricAlg*).

其中,*objectHandle* 为复制密钥的句柄;*newParentHandle* 是新父密钥句柄;*encryptionKeyIn* 是 *innerwrap* 加密密钥,该密钥或由 caller 传入,或是由 TPM 产生;*symmetricAlg* 是对称加密算法.该函数执行后返回 3 个值:其一是 *encryptionKeyOut*,*encryptionKeyOut* 返回的是由 TPM 产生的内部加密密钥,如果 TPM 没产生内部加密密钥,该值返回 *null*;其二是 *duplicate*,*duplicate* 是复制数据,封装了被复制密钥的 *Sensitive Area*;最后是 *outSymSeed*, *outSymSeed* 是 *outerwrap* 的密钥种子,由它可以产生外部对称加密的密钥.

该接口的执行过程如下.

(1) 检查迁移密钥的属性 *fixedTPM* 和 *fixedParent*:如果设置不是(0,0),就结束复制过程.

(2) 检查迁移密钥的属性 *encryptedDuplication*:

- 如果设置为 1,则判断 *newParentHandle* 是否为 *TPM_RH_NULL*:如果为 *TPM_RH_NULL*,则结束复制过程;否则,转到步骤(3);
- 如果设置为 0,则转到步骤(4).

(3) 执行 *innerwrap*,用 *encrptionKeyIn* 对复制密钥的 *sensitiveArea* 进行加密,生成 *encSensitive*.

(4) 执行 *outerwrap*,用密钥种子 *seed* 生成加密密钥和 HMAC 密钥,对 *encSensitive* 进行加密和 HAMC 运算,得到 *dupSensitive* 和 *outerHMAC*.

- 接口 2:复制数据导入接口:

TPM2_Import(*newparentHandle*,*encryptionKey*,*duplicate*,*inSymSeed*).

其中,*newparentHandle* 是新父密钥句柄;*encryptionKey* 是源 TPM 内的 *innerwrap* 密钥,其值由 *TPM2_Duplicate* 的返回值 *encryptionKeyOut* 提供;*duplicate* 是复制数据,此值由 *TPM2_Duplicate* 的返回值 *duplicate* 提供; *inSymSeed* 是源 TPM 内的 *outerwrap* 密钥种子,此值由 *TPM2_Duplicate* 的返回值 *outSymSeed* 提供.

该接口的执行过程如下:

(1) 检查复制密钥的属性 *fixedTPM* 和 *fixedParent*:如果设置不是(0,0),就结束导入过程.

(2) 检查新父密钥是否为存储密钥:如不是,结束导入过程.

(3) 检查 *innerwrap* 的加密密钥 *encryptionKey* 是否正确:如果不正确,结束导入过程.

(4) 检查 *outerwrap* 的密钥种子 *inSymSeed* 是否正确:如果不正确,结束导入过程.

(5) 由 *inSymSeed* 和 *newparentHandle* 恢复出源 TPM 的 *outerwrap* 的 HAMC 密钥,通过 *outerHAMC* 验证 *dupsensitive* 及其 *name* 的真实性和完整性;然后再恢复出 *outerwrap* 的对称密钥,对 *dupsensitive* 解密得到 *encsensitive*.

(6) 用 *encryptionKey* 对 *encsensitive* 进行解密,得到被复制密钥的 *sensitive*;并对 *sensitive* 及其 *name* 进行完整性校验.

通过对 *TPM2_Duplication* 接口和 *TPM2_Import* 接口的分析可知:采用 TPM2.0 的密钥复制接口来设计密钥迁移协议,需要将新父密钥从目标 TPM 传递到源 TPM,源 TPM 调用 *TPM2_Duplication* 接口得到被迁移密钥的复制数据,目标 TPM 调用 *TPM2_Import* 将复制数据载入.基本的流程如下.

(1) 目标 TPM 将新父密钥传递给源 TPM.

(2) 源 TPM 调用 *TPM2_Duplication* 接口,根据迁移密钥的复制属性(*fixedTPM*,*fixedParen*, *encryptedDuplication*)和新父密钥的 *newParentHandle* 类型实施 *innerwrap* 和 *outerwrap*,得复制数据.

(3) 将复制数据传递给目标 TPM,目标 TPM 调用 *TPM2_Import* 将被迁移密钥加载到新父密钥下.

1.4 innerwrap和outerwrap过程

由第 1.3 节可以看出,在利用 *TPM2_Duplication* 接口和 *TPM2_Import* 接口进行密钥迁移时,复制过程的安全性不仅依赖 *innnerwrap*,而且依赖 *outerwrap*.下面我们对 *innnerwrap* 和 *outerwrap* 进行分析.

1.4.1 *innerwrap* 过程

通过对《TPM-Rev-2.0-Part-1-Architecture-01.38》和《TPM-Rev-2.0-Part-3-Commands-01.38-code》分析发现,密钥复制过程中是否进行 *innerwrap* 是由迁移密钥的属性 *encryptedDuplication* 和新父密钥的密钥句柄类型决定,只有当 *encryptedDuplication*=1 且 *newParentHandle*!=*TPM_RH_NULL* 时,*innerwrap* 才能发生. *innerwrap* 过程中需要的对称加密密钥由 caller 决定,caller 可以选择输入,也可以选择由 TPM 自行产生.

innerwrap 可以在复制过程中为迁移密钥提供完整性和机密性,包括两个步骤:首先是用复制密钥的 Hash 算法计算复制密钥 *sensitive* 和 *name* 的哈希值 *innerIntegrity*,保证复制密钥的完整性;然后,用某对称加密算法的 CBF 模式对 *innerIntegrity*||*sensitive* 进行加密,得到 *encSensitive*,其中,需要的对称加密算法和密钥由 caller 将其作为参数传入 *TPM2_Duplication*.

从以上过程可以看出,源 TPM 要完成 *innerwrap*,需要确保加密密钥安全地传递到目的 TPM.因此,如何将此加密密钥安全地传递到目的 TPM,是需要考虑的重要问题.需要考虑如下 3 种情况.

(1) 如果新父密钥是非对称密钥,无论迁移密钥是对称密钥还是非对称密钥,*innerwrap* 中需要的对称加密密钥可以由源 TPM 产生或由 caller 输入,且可以采用《TPM-Rev-2.0-Part-1-Architecture-01.38》中 B.10.3 或 C.6.3 中规定的算法进行保护交换.

(2) 如果新父密钥是对称密钥,而复制密钥是非对称密钥,则 *innerwrap* 中需要的对称加密密钥可以由目的 TPM 产生,且可以采用《TPM-Rev-2.0-Part-1-Architec-ture-01.38》中 B.10.3 或 C.6.3 中规定的算法进行保护交换.

(3) 如果新父密钥是对称密钥,而迁移密钥也是对称密钥,则 *innerwrap* 中需要的密钥无论是由源 TPM 产生或目的 TPM 产生或 caller 输入,《TPM-Rev-2.0-Part-1-Architecture-01.38》中还未给出此密钥的保护交换办法.

1.4.2 outerwrap 过程

通过对《TPM-Rev-2.0-Part-1-Architecture-01.38》和《TPM-Rev-2.0-Part-3-Commands-01.38-code》的分析发现,密钥复制过程中是否进行 *outerwrap* 是由新父密钥的密钥句柄类型决定,当 *newParentHandle*!=*TPM_RH_NULL* 时,*outerwrap* 就会发生;当 *newParentHandle=TPM_RH_NULL* 时,*outerwrap* 不会发生.究其原因是:*outerwrap* 过程主要由新父密钥控制,如果新父密钥的句柄为 *TPM_RH_NULL*,则新父密钥不能为 *outerwrap* 提供需要的算法及参数.

outerwrap 可以在复制过程中为迁移密钥提供机密性、完整性以及对新父密钥的认证性,包括两个步骤.

- 一是对 *encSensitive* 进行加密,具体过程为:首先,获得一个密钥种子 *seed*;然后,将此密钥种子、新父密钥的 *npNameAlg* 和迁移密钥的 *Name* 作为 *KDFa*()的参数,产生对称加密密钥;最后,采用新父密钥的对称加密算法对 *encSensitive* 进行加密,得到 *dupSensitive*.
- 二是对 *dupSensitive* 和 *Name* 进行 HAMC 运算,产生消息码 *outerHMAC*.具体过程为:首先,用 *Seed* 和新父密钥的 *npNameAlg* 作为 *KDFa*()的参数,产生 HAMC 密钥;然后,采用新父密钥的 HMAC 算法进行运算,获得消息码 *outerHMAC*.

从以上过程可以看出,源 TPM 要完成 *outerwrap* 不仅需要获得新父密钥的相关参数,而且需要确保密钥种子 *seed* 能在源 TPM 和目的 TPM 之间安全地交换.需要考虑的 3 种情况与 *innerwrap* 要考虑的 3 种情况一致.

2 问题分析

本节先基于 TPM2.0 的密钥复制接口设计最初的密钥迁移协议,然后详细分析其存在的问题[8].

该初始迁移协议包括 6 个参与实体:源 TPM、源 TPM 的所有者、目标 TPM、目标 TPM 的所有者、源 TPM 所在的主机、目标 TPM 所在的主机,其中,前 4 个实体是可信的,而源和目标 TPM 所在的主机被认为是不可信的.为了叙述方便,我们定义如下符号及函数,见表 4.

Table 4 Symbols and functions

表 4 符号及函数

符号/函数	描述
T_S,O_S,H_S	源 TPM、源 TPM 的所有者、源 TPM 所在主机
T_D,O_D,H_D	目标 TPM、目标 TPM 的所有者、目标 TPM 所在主机
newParent,*newParentHandle*	新父密钥,新父密钥句柄
Object,*objectHandle*	复制密钥,复制密钥句柄
encryptionKeyin	*innerwrap* 的对称密钥
Seed	*outerwrap* 的密钥种子
RSA_OAEP, *ECC_ECDH*	RAS 和 ECC 加密算法
symmetricAlg	源 TPM 和目标 TPM 都支持的对称加密算法

2.1 基于密钥复制接口的密钥迁移协议

基于第 1 节介绍的背景知识,并根据《TPM-Rev-2.0-Part-1-Architecture-01.38》和《TPM-Rev-2.0-Part-3-Commands-01.38-code》规范的要求,我们设计出基于复制接口的密钥迁移协议,其流程如下.

1. $H_D \rightarrow H_S$:*newParent*.*publicAera*,*newparentHandle*,*symmetrical*.
2. O_S:

(1) 获得 *objectHandle*,*newparentHandle*.

(2) 决定 *encryptionKeyin* 的产生方式,可以输入,也可以由 TPM 内部产生.

(3) 调用接口 *TPM2_Duplication*(*objectHandle*,*newParentHandle*,*encrptionKeyIn*,*symmetricAlg*).

3. T_S:执行复制过程:

(1) 检查迁移密钥的属性 *fixedTPM* 和 *fixedParent*:如果设置不是(0,0),就结束复制过程,转到步骤 10.

(2) 检查迁移密钥的属性 *encryptedDuplication*:

- 如果设置为 1,则判断 *newParentHandle* 是否为 *TPM_RH_NULL*:如果为 *TPM_RH_NULL*,则结束复制过程,转到步骤 10;否则转到步骤 3 中的(3);
- 如果设置为 0:*newParentHandle* 不为 *TPM_RH_NULL*,则转到步骤 3.(5);否则,转向步骤 3.(7).

(3) 执行 *innerwrap*,由 caller 输入或 TPM 产生 *encrptionKeyIn*,并用 *encrptionKeyIn* 对迁移密钥的 *sensitiveArea* 进行加密,生成 *encSensitive*.

(4) 对 *encrptionKeyIn* 进行保护,分为两种情况:

- 如果新父密钥是非对称密钥,如 RSA 或 ECC 密钥,则用新父密钥的公钥加密 *encrptionKeyIn*,即 *CencrptionKeyIn*=*RSA_OAEP*(*newParentHandle*,*encrptionKeyIn*)或 *ECC_ECDH*(*newParentHandle*,*encrptionKeyIn*),并将 *sysmetrickey*=*CencrptionKeyIn*;
- 如果新父密钥是对称密钥,则直接 *sysmetrickey*=*encrptionKeyIn*.

(5) 执行 *outerwrap*,由 TPM 产生密钥种子 *seed*,用密钥种子 *seed* 生成加密密钥和 HMAC 密钥,对 *encSensitive* 进行加密和 HAMC 运算,得到 *dupSensitive* 和 *outerHMAC*.

(6) 对 *Seed* 进行保护,分两种情况:

- 如果新父密钥是非对称密钥,如 RSA 或 ECC 密钥,则用新父密钥的公钥加密 *Seed*,即 *CSeed*=*RSA_OAEP*(*newParentHandle*,*Seed*)或 *ECC_ECDH*(*newParentHandle*,*Seed*),并将 *sysmetricSeed*=*Cseed*;
- 如果新父密钥是对称密钥,则直接 *sysmetricSeed*=*Seed*.

结束复制过程,转到步骤 4.

(7) 对复制密钥既不进行 *innerwrap*,也不进行 *outerwrap*,将 *dupSensitive*=*sensitiveArea*,*sysmetrickey*=*NULL*,*sysmetricSeed*=*NULL*.

4. $T_S \rightarrow O_S$:*dupSensitive*,*outerHMAC*,*sysmetrickey*,*sysmetricSeed*.
5. $O_S \rightarrow H_S$:*dupSensitive*,*outerHMAC*,*sysmetrickey*,*sysmetricSeed*.

6. $H_S \rightarrow H_D$:*dupSensitive*,*outerHMAC*,*sysmetrickey*,*sysmetricSeed*.
7. $H_D \rightarrow O_D$:*dupSensitive*,*outerHMAC*,*sysmetrickey*,*sysmetricSeed*.
8. $O_D \rightarrow T_D$:*dupSensitive*,*outerHMAC*,*sysmetrickey*,*sysmetricSeed*.
9. T_D:执行导入过程:

(1) 检查迁移密钥的属性 *fixedTPM* 和 *fixedParent*:如果设置不是(0,0),就结束导入过程,转到步骤 10.

(2) 检查新父密钥是否为存储密钥:如不是,结束导入过程,转到步骤 10.

(3) 根据 *sysmetricSeed* 参数是否为 *NULL* 来判断是否进行了 *outerwrap*.

- 如果为 *NULL*,则转到下一步;
- 否则,直接得到 *seed* 或用私钥解密 *sysmetricSeed* 得到 *seed*,按照 *symmetricAlg* 算法生成 HMAC 密钥 *HMACkey*,并对 *dupSensitive*||*name* 进行 HAMC 运算,将得到的值与 *outerHMAC* 比较:如果不相等,终止导入过程,转到步骤 10;如果相等,则用 *seed* 生成对称加密密钥 *symkey* 并解密 *dupSensitive*,得到 *encSensitive*.

(4) 根据 *sysmetrickey* 参数是否为 *NULL* 来判断是否进行了 *innerwrap*.

- 如果为 *NULL*,则转到步骤 10;
- 否则,直接得到 *encrptionKeyIn* 或用新父密钥的私钥解密 *sysmetrickey* 得到 *encrptionKeyIn*,用 *symmetricAlg* 算法对 *encSensitive* 进行解密,得到 *Sensitive* 和 *name*,用 Hash 算法对 *Sensitive*||*name* 进行完整性验证:验证通过,则表明迁移成功,转到步骤 10;验证不通过,则表明迁移不成功,转到步骤 10;

10. 结束迁移过程.

2.2 存在问题

从第 2.1 节可以看出,基于 TPM2.0 密钥复制接口设计的密钥迁移协议存在如下安全问题.

- 问题 1:该协议缺少源 TPM 和目标 TPM 间的身份认证,导致密钥能够在敌手和 TPM 间迁移.存在着如下两种情况:(1) 源 TPM 不能认证 *newParent* 是否是目标 TPM 的密钥,导致敌手可以用其控制的密钥迁移源 TPM 的密钥,并获得密钥明文;(2) 目标 TPM 不能认证迁移数据是否来自源 TPM,使敌手可以将其控制的密钥迁移到目标 TPM 中.
- 问题 2:当复制密钥的属性 *encryptedDuplication*=0 且新父密钥的句柄 *newParentHandle*=*TPM_RH_NULL* 时,复制接口不能实施 *innerwrap* 和 *outerwrap*,迁移密钥将以明文传输而造成泄露.
- 问题 3:当新父密钥是对称密钥时,*innerwrap* 中的对称加密密钥以及 *outerwrap* 中的密钥种子如何在源 TPM 与目标 TPM 之间安全交换,《TPM-Rev-2.0-Part-1-Architecture-01.38》并没有给出具体的解决办法.

另外,第 2.1 节的密钥迁移协议中复制流程比较复杂,其中有几个关键的因素决定复制流程:首先,密钥能否复制是由复制密钥的属性(*fixedTPM*,*fixedParent*)决定;其次,复制过程中是否实施 *innerwrap* 由复制密钥的 *encryptedDuplication* 属性决定;第三,复制过程中是否实施 *outerwrap* 由新父密钥的句柄类型决定;第四,*innerwrap* 中对称密钥和 *outerwrap* 中密钥种子的保护交换还依赖新父密钥的密钥类型.不同的属性值决定了不同的复制流程,在所有的流程中,部分流程的输出结果是存在安全隐患的.由于所有的判断均在 TPM 内部进行,在执行完成之前,外界无法知道 *TPM2_Duplication*()的输出结果.为了保证复制过程的安全,需要提前掌握复制流程.因此,外界需要一个控制中心,提前获知迁移密钥属性(*fixedTPM*,*fixedParent*,*encryptedDuplication*)、新父密钥类型和新父密钥句柄类型,从而提前掌握复制流程,并对复制过程的输出结果采取合理的保护措施.

3 基于 Duplication Authority 的密钥迁移协议

根据第 2 节的分析,我们提出基于 Duplication Authority 的密钥迁移协议,该协议以 Duplication Authority 为控制和认证中心,负责对源 TPM 和目标 TPM 进行认证并控制复制流程.该协议包括 3 个阶段:初始化阶段、认

证和属性获取阶段以及控制和执行阶段.鉴于篇幅和突出重点,在该协议描述过程中,我们仅考虑 TPM 与 Duplication Authority 以及源 TPM 与目标 TPM 之间的直接信息交互.

3.1 初始化阶段

初始化阶段的主要任务是源和目的 TPM 到 Duplication Authority 注册并对 TPM 进行认证,流程如图 3 所示.我们用 DA 代表 Duplication Authority,(K_{pub_DA},K_{pri_DA})是 Duplication Authority 的一对非对称密钥,$Cert_{DA}$表示该公钥的证书.注册过程是 TPM 将自己的身份标识(ID_{TPM})注册到 DA 的数据库中,$E(\cdot)$为加密函数,$Sign(\cdot)$为签名函数,$Verify(\cdot)$为验证函数,RA为注册标志位,$Cert_{EK}$为 TPM 的背书证书,N_{tpm}为 TPM 产生的随机数,N_{DA0}为 DA 产生的随机数.具体的交互过程如下.

1. $TPM\rightarrow DA$:$E(ks,[Cert_{EK}||ID_{TPM}||Sign_{pri\_EK}(ID_{TPM})||N_{tpm}]),E(Cert_{DA},ks)$.
2. $DA\rightarrow TPM$:$E(ks,RA||N_{tpm}||N_{DA0})$.
3. $TPM\rightarrow DA$:$E(ks,N_{DA0})$.

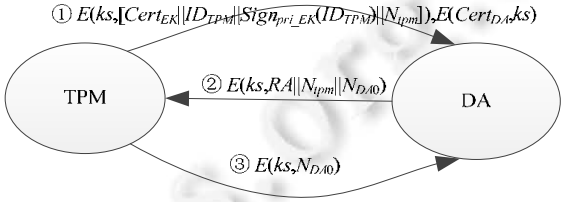

Fig.3 Initialization phase

图 3 初始化阶段

下面我们对上述每一步进行详细解释,具体如下.

- 步骤 1:TPM 向 DA 发送注册信息,发送的具体内容:$E(ks,[Cert_{EK}||ID_{TPM}||Sign_{pri\_EK}(ID_{TPM})||N_{tpm}])||E(Cert_{DA},ks)$.DA 接收到 TPM 发来的注册信息后,首先用 K_{pri_DA} 解密得 ks,并用 ks 解密得 $Cert_{EK}||ID_{TPM}||Sign_{pri\_EK}(ID_{TPM})||N_{tpm}$;然后,$Verify(Cert_{EK})$和 $Verify(Sign_{pri\_EK}(ID_{TPM}))$,验证通过置 RA=1,否则置 RA=0.
- 步骤 2:TPM 接收到 DA 的返回密文 $E(ks,RA||N_{tpm}||N_{DA0})$,用 ks 解密,读取 RA 的值,验证 N_{tpm},如果验证通过,返回 $E(ks,N_{DA0})$.
- 步骤 3:DA 解密得到 N_{DA0},验证 N_{DA0},如果通过,表明 TPM 已经知道注册成功,则将 $Cert_{EK}||ID_{TPM}$ 存入注册数据库.

3.2 认证和属性获取阶段

认证和属性获取阶段包括两个主要任务:其一是源 TPM 与目标 TPM 交互双方基于 Duplication Authority 进行认证;其二是为控制后期复制过程,DuplicationAuthority 通过和目标 TPM 与源 TPM 的交互,获得复制密钥和新父密钥的属性.为了叙述方便,我们假设具体的应用场景是:T_S 和 T_D 均是在 DA 中注册的 TPM,当前要将 T_S 中 *objectHandle* 所指向的密钥对象迁移到 T_D 中 *newParentHandle* 所指向的新父密钥对象下.我们用标志位 S 表示对称密钥、非对称密钥或非密钥对象,S=0 表示对称密钥,S=1 表示非对称密钥,S=−1 为非密钥对象;用 *AlgParameter* 表示密钥对象支持的算法及相关参数,流程如图 4 所示.具体的交互过程如下.

1. $T_D\rightarrow DA$:$E(ks_1,[\ ID_{T_D}\ ||newParentHandle||newParentHandle\rightarrow S||newParentHandle\rightarrow AlgParameter||\ ID_{T_S}\ ||\ objectHandle||\ N_{T_D}\ ]),E(Cert_{DA},ks_1)$.
2. $DA\rightarrow T_S$:$DRequest,Sign(K_{pri\_DA},DRequest||N_{DA1})$.
3. $T_S\rightarrow DA$:$E(ks_2,[\ ID_{T_S}\ ||N_{DA}||\ N_{T_S}\ ]),E(Cert_{DA},ks_2)$.
4. $DA\rightarrow T_S$:$E(ks_2,[\ ID_{T_D}\ ||objectHandle||newParentHandle\rightarrow AlgParameter||\ N_{T_D}\ ||N_{DA1}||\ N_{T_S}\ ])$.
5. $T_S\rightarrow DA$:$E(ks_2,[fixedTPM||fixedParent||encryptedDuplication||objectHandle\rightarrow S||\ N_{T_D}\ ||N_{DA1}||\ N_{T_S}\ ])$.
6. $DA\rightarrow T_D$:$E(ks_1,[\ N_{T_D}\ ||N_{DA1}||\ N_{T_S}\ ])$.

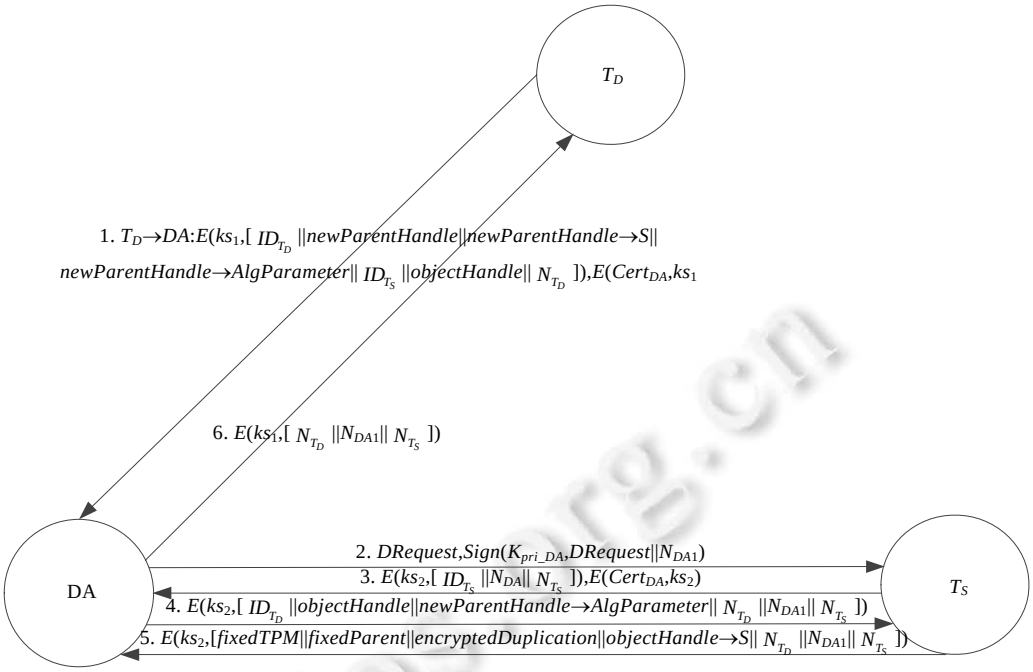

Fig.4 Authentication and attribute acquisition process

图 4 认证和属性获取阶段

下面,我们对上述每一步进行详细解释.具体如下.

- 步骤 1:T_D 向 DA 发送迁移请求秘密信息.DA 用私钥解密 $E(Cert_{DA},ks_1)$得 ks_1,并用 ks_1 解密 T_D 发送的具体内容,得到目标 TPM 的 ID、新父密钥的句柄 *newParentHandle*、新父密钥标志位 *S*、新父密钥的算法及相关参数 *AlgParameter*、源 TPM 的 ID 以及现时 N_{T_D} 等.DA 验证 T_D 的 ID_{T_D} 和 T_S 的 ID_{T_S},确认是否均为合法的 TPM,如果不合法则终止.
- 步骤 2:DA 根据 ID_{T_S} 向源 TPM 发送复制请求 *DRequest*,并对 *DRequest* 和 N_{DA1} 进行了签名.签名的目的是为了保证复制请求的真实性和认证性.
- 步骤 3:T_S 用 $Cert_{DA}$ 验证复制请求后,确认此信息来自 DA,并向 DA 发送共享密钥 ks_2 和现时 N_{T_S}.
- 步骤 4:DA 解密获得 ks_2,用 ks_2 加密目的 TPM 的 ID、复制密钥的句柄 *objectHandle*、新父密钥的算法以及现时 N_{T_D} ,N_{DA1} 和 N_{T_S} 并发送给 T_S.
- 步骤 5:T_S 解密后,获知有 T_D 要迁移 *objectHandle* 所指向的密钥对象到其 *newParentHandle* 所指向的新父密钥对象下.由于目标 TPM 的 ID 来自 DA,因此 T_S 认为 T_D 是合法可信的 TPM,于是在确认自己支持新父密钥的算法及参数后,向 DA 发送了 *objectHandle* 所指向的密钥对象的复制属性 *fixedTPM*, *fixedParent*,*encryptedDuplication* 以及密钥类型和现时 N_{T_D} ,N_{DA1} 和 N_{T_S}.
- 步骤 6:DA 向 T_D 发回现时 N_{T_D} ,N_{DA1} 和 N_{T_S} ,表明迁移准备就绪.

至此,不仅 DA 获得了在复制过程中需要的复制密钥和新父密钥的相关属性,而且 T_S 和 T_D 得到相互认证.

3.3 控制和执行阶段

控制和执行阶段包括两个主要任务:其一是 DA 根据复制密钥和新父密钥的相关属性控制和选择具体的复制流程;其二是迁移过程的具体执行.

上一阶段结束后,DA 已经获得了复制密钥和新父密钥的相关属性,包括 *fixedTPM*,*fixedParent*, *encryptedDuplication*,*newParentHandle*,*objecthandle→S*,*newParentHandle→S* 等,这些属性的取值一共有 96 种组

合,但大部分组合是无效组合,DA 可以根据这些属性组合对执行过程进行控制.我们经过分析,有效组合一共有 12 种.为了方便叙述,我们用 *EndEcho* 表示复制过程终止.限于篇幅,各种情况的密钥迁移流程图略.具体的 12 种情况如下.

情况 1:当 *fixedTPM*!=0 或 *fixedParent*!=0 时,具体流程如下.

(1) $DA \rightarrow T_D$:$E(ks_1,[EndEcho\| N_{T_D} \|N_{DA1}\| N_{T_S}])$.

(2) $DA \rightarrow T_S$:$E(ks_2,[EndEcho\| N_{T_D} \|N_{DA1}\| N_{T_S}])$.

下面,我们对上述每一步进行详细解释.具体如下.

- 步骤 1:DA 向 T_D 发送 *EndEcho*,结束密钥迁移过程.
- 步骤 2:DA 向 T_S 发送 *EndEcho*,结束密钥迁移过程.

本属性组合表明:只有当迁移密钥的 *fixedTPM* 和 *fixedParent* 属性都置 0 时,复制及迁移过程才可继续进行.

情况 2:当 *fixedTPM*=0,*fixedParent*=0,*encryptedDuplication*=1,*newParentHandle*=*TPM_RH_NULL* 时,具体流程如下.

(1) $DA \rightarrow T_D$:$E(ks_1,[EndEcho\| N_{T_D} \|N_{DA1}\| N_{T_S}])$.

(2) $DA \rightarrow T_S$:$E(ks_2,[EndEcho\| N_{T_D} \|N_{DA1}\| N_{T_S}])$.

下面,我们对上述每一步进行详细解释.具体如下.

- 步骤 1:DA 向 T_D 发送 *EndEcho*,结束密钥迁移过程.
- 步骤 2:DA 向 T_S 发送 *EndEcho*,结束密钥迁移过程.

本属性组合表明:在密钥复制过程中,如果只进行 *innerwrap* 是不行的,复制及迁移过程会终止.

情况 3:当 *fixedTPM*=0,*fixedParent*=0,*encryptedDuplication*=1,*newParentHandle*!=*TPM_RH_NULL*,*objecthandle*→*S*=1,*newParentHandle*→*S*=1 时,具体流程如下.

(1) $DA \rightarrow T_S$:$Sign(K_{pri\_DA},[ID_{T_D}\|newParentHandle\|newParentHandle \rightarrow S\|newParentHandle \rightarrow AlgParameter\| N_{T_D} \|N_{DA1}\| N_{T_S}]),E(ks_2,[ID_{T_D}\|newParentHandle\|newParentHandle \rightarrow S\|newParentHandle \rightarrow AlgParameter\| N_{T_D} \|N_{DA1}\| N_{T_S}])$.

(2) $T_S \rightarrow T_D$:*dupSensitive*,*outerHMAC*,*CencryptionKeyout*,*CSeed*,*encRng*.

(3) $T_D \rightarrow T_S$:*encRng*.

(4) $T_S \rightarrow DA$:*encRng*.

下面,我们对上述每一步进行详细解释.具体如下.

- 步骤 1:T_S 接收到 DA 的消息,解密得到目标 TPM 的 ID、新父密钥的句柄 *newParentHandle*、新父密钥标志位 *S* 的值、新父密钥的算法及相关参数 *AlgParameter* 以及现时 N_{T_D},N_{DA1} 和 N_{T_S}.然后用 $Cert_{DA}$ 验证签名,再验证现时 N_{T_D},N_{DA1} 和 N_{T_S}.所有的验证通过,T_S 将进行如下计算.
  1) 产生 *innerwrap* 需要的对称加密密钥 *encryptionKeyIn*(可由 T_S 的 O_S 输入).
  2) 计算 $innerIntegrity=\boldsymbol{H}_{object.nameAlg}(object.sensitive\|object.name)$.
  3) 计算 $encSensitive=\boldsymbol{CFB}_{object.symAlg}(encryptionKeyIn,0,innerIntegrity\|object.sensitive)$.
  4) 根据 *AlgParameter*,计算 *CencryptionKeyIn*=*RSA_OAEP*(*newParentHandle*→*PubliceArea*,*encryptionKeyIn*)或 *CencryptionKeyIn*=*ECC_ECDH*(*newParent Handle*→*PubliceArea*,*encryptionKeyIn*).
  5) 产生 *outerwrap* 需要的密钥种子 *seed*.
  6) 计算 $symKey=\boldsymbol{KDFa}(AlgParameter,seed,\text{“STORAGE”},Name,NULL,bits)$.
  7) 计算 $dupSensitive=\boldsymbol{CFB}_{AlgParameter}(symKey,0,encSensitive)$.
  8) 计算 $encRng=\boldsymbol{CFB}_{AlgParameter}(symKey,0,[N_{T_D}\|N_{DA}\|N_{T_S}])$.
  9) 计算 $HMACkey=\boldsymbol{KDFa}(AlgParameter,seed,\text{“INTEGRITY”},NULL,NULL,bits)$.
  10) 计算 $outerHMAC=\boldsymbol{HMAC}_{npNameAlg}(HMACkey,dupSensitive\|objecthandle \rightarrow Name)$.

  以上过程实际上是调用 *TPM2_Duplicate*(*objectHandle*,*newParentHandle*,*encryptionKeyIn*,*symmetric*

Alg)进行密钥复制,对复制密钥实施 *innerwrap* 和 *outerwrap*.

- 步骤 2:T_D 收到 T_S 传来的 *dupSensitive*,*outerHMAC*,*CencryptionKeyout*,*CSeed* 和 *encRng*,进行如下计算.
  1) 用新父密钥的私钥解密 *CSeed*,得到 *Seed*.
  2) 用同样的算法生成 *HMACkey* 和 *symKey*.
  3) 用 *symKey* 解密 *encRng*,验证随机数 N_{T_D} ,N_{DA1} 和 N_{T_S} ;然后,用 *HMACkey* 验证 *outerHMAC*.如果验证都通过,用 *symKey* 对 *dupSensitive* 进行解密,得到 *encSensitive*.
  4) 用新父密钥的私钥解密 *CencryptionKeyout* 得到 *encryptionKeyout*,用 *encryption-Keyout* 解密 *encSensitive* 得到 *innerIntegrity* 和 *Sensitive*,用同样的 ***H*** 算法验证 *object.sensitive*||*object.name* 的完整性.如果验证通过,则复制成功.

  以上过程实际上是调用 *TPM2_Import*(*newparentHandle*,*CencryptionKeyout*,*dupSenstive*,*CSeed*)进行复制密钥导入.
- 步骤 3:T_D 给 T_S 返回 *encRng*,T_S 解密验证随机数.如果相同,表明迁移成功.
- 步骤 4:T_S 给 DA 返回 *encRng*,DA 解密验证随机数.如果相同,表明迁移成功,结束迁移过程.

情况 4: 当 *fixedTPM*=0,*fixedParent*=0,*encryptedDuplication*=1,*newParentHandle*!=*TPM_RH_NULL*,*object handle*→*S*=1,*newParentHandle*→*S*=0 时,其具体流程与情况 3 唯一的不同是新父密钥为对称密钥, *innerwrap* 所需的 *encryptionKeyin* 和 *outerwrap* 所需的 *Seed* 只能在 T_D 中产生,才能利用《TPM-Rev-2.0-Part-1-Architecture-01.38》中规定的 *RSA_OAEP* 或 *ECC_ECDH* 算法对 *encryptionKeyin* 和 *Seed* 进行保护交换.我们用标志位 *Migrate_object.publicArea_to_*T_D 表示 DA 要求 T_S 将复制密钥的 *publicArea* 传递给 T_D.具体流程如下.

(1) $DA \rightarrow T_S: Sign(K_{pri\_DA}, [\ ID_{T_D}\ || newParentHandle || newParentHandle \rightarrow S || newParentHandle \rightarrow AlgParameter || Migrate\_object.publicArea\_to\_T_D || N_{T_D}\ || N_{DA} || N_{T_S}\ ]), E(ks_2, [\ ID_{T_D}\ || newParentHandle || newParentHandle \rightarrow S || newParentHandle \rightarrow AlgParameter || Migrate\_object.publicArea\_to\_T_D || N_{T_D}\ || N_{DA1} || N_{T_S}\ ])$.

(2) $T_S \rightarrow DA: E(ks_2, [object.publicArea || N_{T_D}\ || N_{DA1} || N_{T_S}\ ])$.

(3) $DA \rightarrow T_D: E(ks_1, [object.publicArea || N_{T_D}\ || N_{DA1} || N_{T_S}\ ])$.

(4) $T_D \rightarrow T_S$:*CencryptionKeyout*,*Cseed*.

(5) $T_S \rightarrow T_D$:*dupSensitive*,*outerHMAC*,*encRng*.

(6) $T_D \rightarrow T_S$:*encRng*.

(7) $T_S \rightarrow DA$:*encRng*.

下面,我们对上述每一步进行详细解释.具体如下.

- 步骤 1:与情况 3 基本相同的步骤一基本相同,不同的是在步骤一中,DA 发送给 T_S 消息增加了标志位 *Migrate_object.publicArea_to_*T_D.
- 步骤 2、步骤 3:T_S 将复制密钥的公钥部分通过 DA 传给 T_D.
- 步骤 4:由 T_D 产生 *encryptionKeyin* 和 *Seed*,用复制密钥的公钥加密得到 *CencryptionKeyin* 和 *CSeed* 并传递给 T_S.
- 步骤 5:与情况 3 的步骤 2 基本一致,T_S 调用 *TPM2_Duplicate*(*objectHandle*,*newParentHandle*,*encryptionKeyIn*,*symmetricAlg*)进行密钥复制,对复制密钥实施 *innerwrap* 和 *outerwrap*.由于 *encryptionKeyin* 和 *Seed* 已经在 T_D,因此发送给 T_D 的数据就只有 *dupSensitive*,*outerHMAC* 和 *encRng*.
- 步骤 6:与情况 3 的步骤 3 一致.
- 步骤 7:与情况 3 的步骤 4 一致.

情况 5: 当 *fixedTPM*=0,*fixedParent*=0,*encryptedDuplication*=1,*newParentHandle*!=*TPM_RH_NULL*,*object handle*→*S*=0,*newParentHandle*→*S*=1 时,其具体流程与情况 3 一致,略.

情况 6: 当 *fixedTPM*=0,*fixedParent*=0,*encryptedDuplication*=1,*newParentHandle*!=*TPM_RH_NULL*,*object*

handle→S=0,*newParentHandle→S*=0时,其具体流程与前面所有的情况均不相同.本情况的复制密钥和新父密钥都是对称密钥,*encryptionKeyin* 和 *Seed* 无论在 T_S 或 T_D 产生,都无法用《TPM-Rev-2.0-Part-1-Architecture- 01.38》中规定的 *RSA_OAEP* 或 *ECC_ECDH* 算法进行保护交换.为了防止 *encryptionKeyin* 和 *Seed* 泄露,我们设计在 DA 的控制下用 DH 算法在 T_S 和 T_D 之间交换共享密钥作为 *encryptionKeyin* 和 *Seed*.因此,我们用标志位 *Pro_DH_encryptionKeyin_and_Seed* 表明此需求,且 T_S 和 T_D 双方已选择一共同的素数 q 以及 q 的一个原根 a.具体流程如下.

(1) $DA\rightarrow T_S$:*Sign*(K_{pri_DA},[ID_{T_D} ||*newParentHandle*||*newParentHandle→S*||*newParentHandle→AlgParameter*||*Pro_DH_encryptionKeyin_and_Seed*|| N_{T_D} ||N_{DA1}|| N_{T_S}]),*E*(ks_2,[ID_{T_D} ||*newParentHandle*||*newParentHandle→S*||*newParentHandle→AlgParameter*||*Pro_DH_encryptionKeyin_and_Seed*|| N_{T_D} ||N_{DA1}|| N_{T_S}]).

(2) $T_S\rightarrow DA$:*E*(K_{s2},[ID_S||Y_S|| N_{T_D} ||N_{DA1}|| N_{T_S}]).

(3) $DA\rightarrow T_D$:*E*(K_{s1},[ID_S||Y_S|| N_{T_D} ||N_{DA1}|| N_{T_S} ||*Sign*(*pri_DA*,[ID_S||Y_S|| N_{T_D} ||N_{DA1}|| N_{T_S}])]).

(4) $T_D\rightarrow DA$:*E*(K_{s1},[ID_D||Y_D|| N_{T_D} ||N_{DA1}|| N_{T_S}]).

(5) $DA\rightarrow T_S$:*E*(K_{S2},[ID_D||Y_D|| N_{T_D} ||N_{DA1}|| N_{T_S} ||*Sign*(*pri_DA*,[ID_D||Y_D|| N_{T_D} ||N_{DA1}|| N_{T_S}])]).

(6) $T_S\rightarrow T_D$:*dupSensitive*,*outerHMAC*,*encRng*.

(7) $T_D\rightarrow T_S$:*encRng*.

(8) $T_S\rightarrow DA$:*encRng*.

下面,我们对上述每一步进行详细解释.具体如下.

- 步骤 1:与情况 3 的步骤一基本相同,不同的是 DA 发送给 T_S 消息增加了标志位 *Pro_DH_encryptionKeyin_and_Seed*.

以下步骤 2~步骤 5 就是防中间人攻击的 DH 算法.

- 步骤 2:T_S 计算 $Y_s = a^{X_s} \mod q$,其中,X_s 是 T_S 的任选素数,$X_s<q$.然后,把 $ID_S||Y_s$ 发送给 DA.
- 步骤 3:DA 对 $ID_S||Y_s$ 进行签名 *Sign*(*pri_DA*,[ID_S||Y_S]),并一起发送给 T_D.
- 步骤 4:T_D 验证 DA 对 T_S 的签名 *Verify*(*pub_DA* ,*Sign* (*pri_DA*,[ID_S||Y_S])),然后任选素数 X_D,且 $X_D<q$,并计算 $K_0 = Y_S^{X_D} \mod q$,接下来在计算 $Y_D = a^{X_D} \mod q$,然后将 $ID_D||Y_D$ 发送给 DA.
- 步骤 5:DA 对 $ID_D||Y_D$ 进行 *Sign*(*pri_DA*,[ID_D||Y_D]),并一起发送给 T_S,T_S 接收到 Y_D 后,计算:

$$K_0 = Y_S^{X_D} \mod q.$$

- 步骤 6:与情况 3 的步骤 2 基本一致,T_S 调用 *TPM2_Duplicate*(*objectHandle*,*newParentHandle*,*encryptionKeyIn*,*symmetricAlg*)进行密钥复制,对复制密钥实施 *innerwrap* 和 *outerwrap*,K_0 既作为 *innerwrap* 的 *encryptionKeyin*,又作为 *outerwrap* 的 *Seed*.由于 K_0 在 T_S 和 T_D 双方都存在,因此发送给 T_D 的数据就只有 *dupSensitive*,*outerHMAC* 和 *encRng*.
- 步骤 7:与情况 3 的步骤 3 一致.
- 步骤 8:与情况 3 的步骤 4 一致.

情况 7: 当 *fixedTPM*=0,*fixedParent*=0,*encryptedDuplication*=0,*newParentHandle*!=*TPM_RH_NULL*,*objecthandle→S*=1,*newParentHandle→S*=1 时,与情况 3 的流程基本一致,只是在步骤 1 中不进行 *innerwrap*,即没有情况 3 步骤 1 中的步骤 2)~步骤 4),在步骤 2 中也就不进行 *innerwrap* 密钥 *encryptionKeyin* 的保护交换.具体流程如下.

(1) $DA\rightarrow T_S$:*Sign*(K_{pri_DA},[ID_{T_D} ||*newParentHandle*||*newParentHandle→S*||*newParentHandle→AlgParameter*|| N_{T_D} ||N_{DA1}|| N_{T_S}]),*E*(ks_2,[ID_{T_D} ||*newParentHandle*||*newParentHandle→S*||*newParentHandle→AlgParameter*|| N_{T_D} ||N_{DA1}|| N_{T_S}]).

(2) $T_S\rightarrow T_D$:*dupSensitive*,*outerHMAC*,*CSeed*,*encRng*.

(3) $T_D \rightarrow T_S$:*encRng*.

(4) $T_S \rightarrow DA$:*encRng*.

情况 8: 当 *fixedTPM*=0,*fixedParent*=0,*encryptedDuplication*=0,*newParentHandle*!=*TPM_RH_NULL*,*object handle*→*S*=1,*newParentHandle*→*S*=0 时,与情况 4 基本一致.只是在步骤 4 中只传输 *CSeed*,这是因为本情况不进行 *innerwrap*,也就不进行 *innerwrap* 密钥 *encryptionKeyin* 的保护交换.具体流程如下.

(1) $DA \rightarrow T_S$:*Sign*(K_{pri_DA},[ID_{T_D} ||*newParentHandle*||*newParentHandle*→*S*||*newParentHandle*→*AlgParameter*|| *Migrate_object.publicArea_to_*T_D|| N_{T_D} ||N_{DA1}|| N_{T_S}]),*E*(ks_2,[ID_{T_D} ||*newParentHandle*||*newParentHandle*→*S*||*newParentHandle*→*AlgParameter*||*Migrate_object.publicArea_to_*T_D|| N_{T_D} ||N_{DA1}|| N_{T_S}]).

(2) $T_S \rightarrow DA$:*E*(ks_2,[*object.publicArea*|| N_{T_D} ||N_{DA1}|| N_{T_S}]).

(3) $DA \rightarrow T_D$:*E*(ks_1,[*object.publicArea*|| N_{T_D} ||N_{DA1}|| N_{T_S}]).

(4) $T_D \rightarrow T_S$:*Cseed*.

(5) $T_S \rightarrow T_D$:*dupSensitive*,*outerHMAC*,*encRng*.

(6) $T_D \rightarrow T_S$:*encRng*.

(7) $T_S \rightarrow DA$:*encRng*.

情况 9: 当 *fixedTPM*=0,*fixedParent*=0,*encryptedDuplication*=0,*newParentHandle*!=*TPM_RH_NULL*,*object handle*→*S*=0,*newParentHandle*→*S*=1 时,与情况 7 一致,略.

情况 10: 当 *fixedTPM*=0,*fixedParent*=0,*encryptedDuplication*=0,*newParentHandle*!=*TPM_RH_NULL*,*object handle*→*S*=0,*newParentHandle*→*S*=0 时,与情况 6 基本相同,都面对的是复制密钥和新父密钥是对称密钥的情况.不同的是,本情况不需进行 *innerwrap*.因此在步骤 1 中,原请求 T_S 产生 *innerwarp* 密钥和 *outerwrap* 密钥种子的标志 *Pro_DH_encryptionKeyin_and_Seed* 变为了只产生 *outerwrap* 密钥种子的标志 *Pro_DH_Seed*,相应地,用 DH 算法使得 T_S 和 T_D 共享会话密钥 K_0 只作为 *Seed* 进行 *outerwarp*.具体流程如下.

(1) $DA \rightarrow T_S$:*Sign*(K_{pri_DA},[ID_{T_D} ||*newParentHandle*||*newParentHandle*→*S*||*newParentHandle*→*AlgParameter*|| *Pro_DH_Seed*|| N_{T_D} ||N_{DA1}|| N_{T_S}]),*E*(ks_2,[ID_{T_D} ||*newParentHandle*||*newParentHandle*→*S*||*newParentHandle*→*AlgParameter*||*Pro_DH_Seed*|| N_{T_D} ||N_{DA1}|| N_{T_S}]).

(2) $T_S \rightarrow DA$:*E*(K_{s2},[ID_S||Y_s|| N_{T_D} ||N_{DA1}|| N_{T_S}]).

(3) $DA \rightarrow T_D$:*E*(K_{s1},[ID_S||Y_s|| N_{T_D} ||N_{DA1}|| N_{T_S} ||*Sign*(*pri_DA*,[ID_S||Y_s|| N_{T_D} ||N_{DA1}|| N_{T_S}])]).

(4) $T_D \rightarrow DA$:*E*(K_{s1},[ID_D||Y_D|| N_{T_D} ||N_{DA1}|| N_{T_S}]).

(5) $DA \rightarrow T_S$:*E*(K_{S2},[ID_D||Y_D|| N_{T_D} ||N_{DA1}|| N_{T_S} ||*Sign*(*pri_DA*,[ID_D||Y_D|| N_{T_D} ||N_{DA1}|| N_{T_S}])]).

(6) $T_S \rightarrow T_D$:*dupSensitive*,*outerHMAC*,*encRng*.

(7) $T_D \rightarrow T_S$:*encRng*.

情况 11: 当 *fixedTPM*=0,*fixedParent*=0,*encryptedDuplication*=0,*newParentHandle*=*TPM_RH_NULL*,*object handle*→*S*=1,*newParentHandle*→*S*=−1 时.本情况面对的是复制密钥是非对称密钥而新父密钥是非层次结构的非密钥对象,如密钥种子、空授权对象等.显然,这种情况密钥在复制过程中既不进行 *innerwrap*,也不进行 *outerwrap*,复制密钥将以明文在 T_D 和 T_S 之间传输,存在泄露的可能.为了防止泄露,在 DA 的控制下,T_D 和 T_S 通过 DH 算法而共享会话密钥,用该密钥对复制密钥加密.具体流程如下.

(1) $DA \rightarrow T_S$:*Sign*(K_{pri_DA},[ID_{T_D} ||*newParentHandle*||*newParentHandle*→*S*||*newParentHandle*→*AlgParameter*|| *Pro_DH_KEY*|| N_{T_D} ||N_{DA1}|| N_{T_S}]),*E*(ks_2,[ID_{T_D} ||*newParentHandle*||*newParentHandle*→*S*||*newParentHandle*→*AlgParameter*||*Pro_DH_KEY*|| N_{T_D} ||N_{DA1}|| N_{T_S}]).

(2) $T_S \rightarrow DA$:*E*(K_{s2},[ID_S||Y_s|| N_{T_D} ||N_{DA1}|| N_{T_S}]).

(3) $DA \rightarrow T_D$:*E*(K_{s1},[ID_S||Y_s|| N_{T_D} ||N_{DA1}|| N_{T_S} ||*Sign*(*pri_DA*,[ID_S||Y_s|| N_{T_D} ||N_{DA1}|| N_{T_S}])]).

(4) $T_D \rightarrow DA$:*E*(K_{s1},[ID_D||Y_D|| N_{T_D} ||N_{DA1}|| N_{T_S}]).

(5) $DA \rightarrow T_S$:$E(K_{S2},[ID_D||Y_D||\ N_{T_D}\ ||N_{DA1}||\ N_{T_S}\ ||Sign(pri\_DA,[ID_D||Y_D||\ N_{T_D}\ ||N_{DA1}||\ N_{T_S}\ ])])$.

(6) $T_S \rightarrow T_D$:$E(K_0,object.sensitiveArea)$,$E(K_0,[\ N_{T_D}\ ||N_{DA1}||\ N_{T_S}\ ])$.

(7) $T_D \rightarrow T_S$:$E(K_0,E(K_0,[\ N_{T_D}\ ||N_{DA1}||\ N_{T_S}\ ])$.

下面,我们对上述每一步进行详细解释.具体如下.

- 步骤 1:DA 向 T_S 发出 *Pro_DH_KEY* 标志,要求 T_D 和 T_S 交换共享密钥.
- 步骤 2~步骤 5:用防止中间人攻击的 DH 算法使得 T_S 和 T_D 共享密钥 K_0.
- 步骤 6:T_S 用 K_0 加密复制密钥和现时,然后传递给 T_D.
- 步骤 7:T_D 验证现时并回复 T_S,表明复制成功.

情况 12: 当 *fixedTPM*=0,*fixedParent*=0,*encryptedDuplication*=0,*newParentHandle=TPM_RH_NULL*,*objecthandle*→*S*=0,*newParentHandle*→*S*=−1 时,本情况与情况 11 一致,略.

从以上分析可以看出,由于情况 2 与情况 1 一致、情况 5 与情况 3 一致、情况 9 与情况 7 一致、情况 12 与情况 11 也一致,因此,原本 12 种迁移流程实际为 8 种.

4 协议分析

4.1 特点分析

(1) 一致性

在《TPM-Rev-2.0-Part-1-Architecture-01.38》中规定了密钥复制的过程的基本要求,即如何根据复制密钥的复制属性(*fixedTPM*,*fixedParent*,*encryptedDuplication*)以及新父密钥的密钥句柄类型对复制密钥进行 *innerwrap* 和 *outerwrap*,也建议采用 Duplication Authority 对复制过程进行控制、认证等,但并没有给出具体的方案.本文提出的基于 Duplication Authority 的 TPM2.0 密钥迁移协议方案不仅给出了具体的方案,而且和《TPM-Rev-2.0-Part-1-Architecture-01.38》中的要求完全一致.具有一致性.

(2) 完备性

本文考虑了(*fixedTPM*,*fixedParent*,*encryptedDuplication*,*newParentHandle*,*objecthandle*→*S*,*newParentHandle*→*S*)各种合法的组合,并设计了这些组合的密钥迁移方案.在这些方案中,我们不仅考虑了 Duplication Authority 的认证和控制方法,而且还考虑了 *innerwarp* 密钥和 *outerwrap* 密钥种子的保护交换方法,特别是当被复制密钥和新父密钥都是对称密钥时 *innerwarp* 密钥和 *outerwrap* 密钥种子的保护交换方法.另外,我们还考虑了密钥复制过程中既不 *innerwarp* 又不 *outerwrap* 时密钥迁移的安全问题,以及 DA 根据 *objecthandle*→*S* 和 *newParentHandle*→*S* 的类型控制由源 TPM 方或目的 TPM 方产生 *innerwarp* 的加密密钥和 *outerwrap* 所需要的密钥种子.因此,本文的 TPM2.0 密钥迁移协议方案是完备的.

4.2 安全分析

本文提出的基于 Duplication Authority 的 TPM2.0 密钥迁移协议方案具有认证性、机密性、抗中间人攻击和抗重放攻击,因而具有较高的安全性.下面我们对本文提出的方法进行安全性分析,具有以下 4 个特点.

(1) 具有认证性

一方面,所有合法的 TPM 均需用其 EK 和名字 ID 在 Duplication Authority 注册,这就确保了密钥一定是在两个具有合法身份的 TPM 之间进行迁移,注册过程保证了源 TPM 与目标 TPM 之间的认证性;另一方面,当被复制密钥和新父密钥都是对称密钥时,*innerwarp* 密钥和 *outerwrap* 密钥种子的保护交换方法是具有认证功能、防止中间人攻击的 DH 算法.当新父密钥是非层次结构的非密钥对象,如密钥种子、空授权对象等,我们也采用的是具有认证功能、防止中间人攻击的 DH 算法交换保护复制密钥的共享密钥.最后,*outerwrap* 的消息认证码 *outerHAMC* 也具有认证功能.因此,该协议通过各层次保证信息交互的各方均具有合法的身份,故具有认证性.

(2) 具有机密性

在初始化阶段,各 TPM 与 DA 之间的信息交互都是用对称密钥 k_s 加密,因此,如果信息被截取,攻击者会由

于没有对称秘密密钥 k_s 而无法获得信息的具体内容,因此确保了各 TPM 与 DA 之间的机密性;在认证和属性获取阶段,DA,T_S 和 T_D 之间的信息交换也用对称密钥 k_{s1} 和 k_{s2} 加密,因此,如果信息被截取,攻击者会由于没有对称秘密密钥 k_{s1} 和 k_{s2} 而无法获得信息的具体内容,因此确保了 DA,T_S 和 T_D 之间的机密性;而在控制和执行阶段,DA,T_S 和 T_D 之间的信息交换也用对称密钥 k_{s1} 和 k_{s2} 加密,因此,如果信息被截取,攻击者会由于没有对称秘密密钥 k_{s1} 和 k_{s2} 而无法获得信息的具体内容,因此确保了 DA,T_S 和 T_D 之间的机密性.由此可见,无论是在初始化阶段、认证和属性控制阶段还是控制和执行阶段,都保证了交互信息的密文传送,因而具有机密性.

(3) 具有抗中间人攻击

中间人攻击是一种通过修改或者伪装发送消息而达到攻击目的的手段.首先,在初始化阶段,各 TPM 与 DA 之间的信息交互采用的是 DA 证书对会话密钥进行加密,只有拥有证书的私钥才能解密会话密钥,即只有 DA 才能解密会话密钥,因此,即使敌手获得了该消息,由于它无法解密该消息,因此无法伪装和修改该消息进行中间人攻击;在认证和属性获取阶段,DA 和 T_S,T_D 之间的信息交换采用的是 DA 的证书对会话密钥进行加密,只有拥有证书的私钥才能解密会话密钥,即只有 DA 才能解密会话密钥,因此,即使敌手获得了该消息,由于它无法解密该消息,因此无法伪装和修改该消息进行中间人攻击;在控制与执行阶段,DA 和 T_S,T_D 之间的信息交换仍采用的是 DA 证书对会话密钥进行加密,只有拥有证书的私钥才能解密会话密钥,即只有 DA 才能解密会话密钥,因此,即使敌手获得了该消息,由于它无法解密该消息,因此无法伪装和修改该消息进行中间人攻击;而 T_S 和 T_D 之间的密钥交换,我们采用的是抗中间人攻击的 DH 算法,即采用 DA 的证书对交互的信息进行签名,证书具有身份认证功能,中间人无法伪造.因此,具有抗中间人攻击.

(4) 具有抗重放攻击

首先,在初始化阶段,各 TPM 与 DA 之间的信息交互采用了现时 N_{tpm} 和 N_{DA},所以各 TPM 与 DA 可以通过 N_{tpm} 和 N_{DA0} 来确保不是重放消息;在认证和属性获取阶段,DA 和 T_S,T_D 之间的信息交换采用了现时 N_{T_D}, N_{DA1} 和 N_{T_S}, DA 和 T_S,T_D 可以通过 N_{T_D}, N_{DA1} 和 N_{T_S} 来确保不是重放消息;在控制与执行阶段,DA 和 T_S,T_D 之间的信息交换仍然采用了同一现时 N_{T_D}, N_{DA1} 和 N_{T_S}, DA 和 T_S,T_D 之间以及 T_S 和 T_D 均可以通过 N_{T_D}, N_{DA1} 和 N_{T_S} 来确保不是重放消息.因此,具有抗重放攻击.

值得注意的是,本协议中的 DA 具有重要作用.在初始化阶段,源和目的 TPM 的身份信息均注册到 DA;在认证和属性获取阶段,DA 会获得复制密钥和新父密钥的相关属性;在控制和执行阶段,DA 需要根据复制密钥和新父密钥的相关属性控制复制流程.因此,DA 是整个协议的控制中心,不仅需要保护好所有注册 TPM 的身份信息,而且还要保证自身不受攻击.一旦 DA 遭受攻击,攻击者不仅可以窃取注册到 DA 中的 TPM 身份信息,而且还能控制复制流程;攻击者不仅可以获得复制密钥,而且可以将自己控制的密钥复制到目标 TPM 中.这不仅严重威胁到用户的隐私,而且还会造成 TPM 密钥的泄露和加密数据的失窃.因此,本文的 DA 必须是一个可信第三方,对其安全要求必须和 PKI 中的 CA 一样.

4.3 对比分析

实际上,TPM1.2 规范在《TPM-main-1.2-Rev94-part-3》中有关于 Migration 部分,并已定义了与密钥迁移相关的接口,包括 *TPM_CMK_ApproveMA*(),*TPM_CMK_CreateKey*(),*TPM_CMK_CreateTicket*(),*TPM_CMK_CreateBlob*()和 *TPM_CMK_ConvertMigration*()等.通过这些接口设计密钥迁移协议时,也需要 TPM 的所有者指定一个可信的第三方(migration authority,简称 MA)参与,在这一点上与本文的 Duplication Authority 有些相似.为了清楚说明它们之间的异同,我们用 TPM1.2 密钥迁移接口简要设计一个密钥迁移流程框架,忽略细节部分.

为了表述方便,首先假定一个场景:密钥 *Kmigration* 需要从源平台 *Psrc* 迁移到平台 *Pdes*,在 Psrc 端保护 *Kmigration* 的父密钥是 *Kparent*,而 *Pdes* 端选定的将要保护 *Kmigration* 的父密钥是 *Kstorage*.因此,*Kparent* 和 *Kstorage* 必须都是存储密钥.在此假定的场景下,首先需要 TPM 的所有者用接口 *TPM_CMK_ ApproveMA*()授权一个可信的第三方(migration authority,简称 MA),并由 MA 通过接口 *TPM_CMK_CreateTicket*()来指定迁移目标 *Pdes* 并对迁移目标 *Pdes* 进行认证.具体的协议流程如图 5 所示.

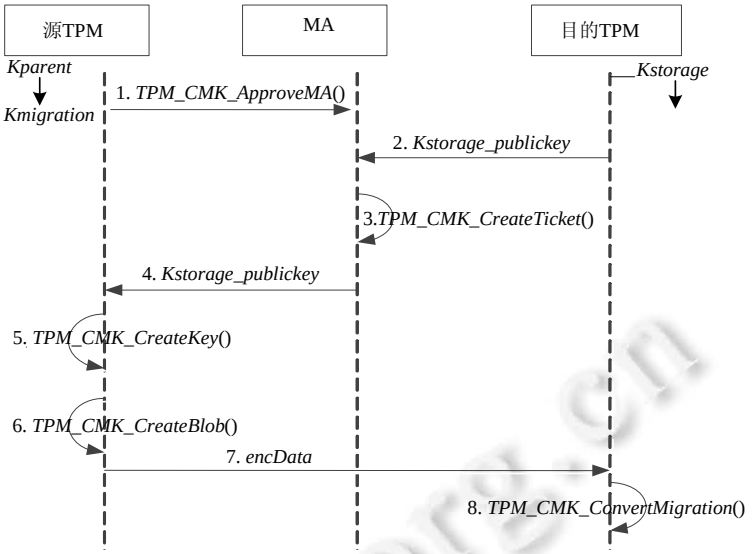

Fig.5 Key migration procedure for TPM1.2

图 5 TPM1.2 密钥迁移流程

在图 5 中,第 1 步是 *Psrc* 调用 *TPM_CMK_ApproveMA*()授权一个可信的第三方 MA;第 2 步和第 3 步是 MA 调用 *TPM_CMK_CreateTicket*()来指定迁移目标 *Pdes*,并对迁移目标 *Pdes* 进行认证;第 4 步和第 5 步是 *Psrc* 调用 *TPM_CMK_CreateKey*()产生待迁移的密钥 *Kmigration*;第 6 步是 *Psrc* 调用 *TPM_CMK_CreateBlob*()生成迁移数据 *encData*;第 7 步是 *Psrc* 将 *encData* 传递给 *Pdes*;第 8 步是 *Pdes* 调用 *TPM_CMK_ConvertMigration*()对迁移数据进行转换.至此,*Pdes* 就可以用 *Kstorage* 为父密钥加载 *Kmigration*.

显然,图 5 与本文第 3 节提出的协议是有区别的.下面,我们在安全性、复杂性、可信第三方的作用等方面对这两个协议进行比较.具体如下.

(1) 在安全性方面,无论是利用 TPM1.2 的迁移接口设计的迁移协议还是本文利用 TPM2.0 密钥复制接口设计的迁移协议,均能具有认证性、机密性、抗中间人攻击和抗重放攻击等特点.因此,只要通过精心设计,这两个协议均能达到同等的安全.

(2) 在复杂性方面,显然,本文的协议要比利用 TPM1.2 的迁移接口设计的迁移协议复杂得多.这是因为 TPM2.0 与密钥相关的功能均扩大了,密钥类型不仅支持非对称密钥,而且还支持对称密钥;密码算法不仅支持 RSA,而且还支持 ECC.在设计密钥迁移协议时,除了需要考虑迁移密钥和目标密钥的复制属性以外,还需要考虑迁移密钥、目标密钥是对称密钥还是非对称密钥的情况,因此,涉及的迁移情况多、控制迁移流程复杂.

(3) 在可信第三方方面,无论是 TPM1.2 中的密钥迁移还是本文设计的密钥复制,均需要可信第三方.利用 TPM1.2 的迁移接口设计的迁移协议需要可信第三方 MA,本文协议中涉及到可信第三方是 DA.但 DA 远比 MA 复杂:MA 只需指定迁移目标并对迁移目标进行认证;而 DA 不仅需要管理各 TPM 的身份信息,而且还要根据迁移密钥、目标密钥的复制属性确定和控制迁移流程.

5 实验验证与性能对比分析

本节将对第 3 节提出的密钥迁移协议进行验证,并与 TPM1.2 的密钥迁移协议进行性能对比分析.

5.1 实验验证

要将迁移密钥迁移到新父密钥下,首先,源 TPM 和目标 TPM 应在 DA 中注册;然后,目标 TPM 向 DA 发起迁移请求,DA 对源 TPM 和目标 TPM 进行认证,并分别获得新父密钥的类型和句柄类型以及迁移密钥的复制属

性(*fixedTPM*,*fixedParent*,*encryptedDuplication*);最后,DA 根据收集到的所有属性,判断此流程属于 12 种流程中的哪一种流程,并控制复制流程,即可完成迁移.

5.1.1 实验环境

目前,市面上已出现 TPM2.0 芯片,经过深入的咨询我们发现,在国内使用 TPM2.0 芯片受到一定的限制.因此,这里我们使用微软的 TPM2.0 模拟器以及 TSS.net 进行实验.本实验共使用 2 台计算机,其中一台既做 target TPM 又充当 DA,剩下一台充当 source TPM.这里我们用 OpenSSL 的 CA 机关模拟 DA,作为注册和控制中心.具体环境配置见表 5,其中,target TPM 与 DA 所在主机的配置完全一致.

Table 5 Experimental environment related configuration

表 5 实验环境相关配置

	source TPM	target TPM	DA
CPU	Intel(R) Core(TM) i5-3210M CPU @2.50GHZ	Intel(R) Core(TM) i7-7500U CPU @2.70GHZ	Intel(R) Core(TM) i7-7500U CPU @2.70GHZ
内存	4GB	8GB	8GB
OS	Windows 7 32bit	Windows 10 64bit	Windows 10 64bit
IP 地址	192.168.0.109/169.254.8.81	192.168.0.101/169.254.12.45	192.168.0.101/169.254.12.45
子网掩码	255.255.255.0/255.255.0.0	255.255.255.0/255.255.0.0	255.255.255.0/255.255.0.0
Openssl 版本	Openssl_1.0.0	Openssl_1.0.0	Openssl_1.0.0
TPM 模拟器	TSS.MSR v2.0 TPM2 simulator	TSS.MSR v2.0 TPM2 simulator	TSS.MSR v2.0 TPM2 simulator
TSS 版本	TSS.MSR-master	TSS.MSR-master	TSS.MSR-master

以情况 3 为例,具体实验流程原型如图 6 所示,其中,*S*,*DA*,*D* 分别表示源 TPM、复制权威和目的 TPM.主要分为初始化、认证和属性获取以及控制迁移这 3 个阶段.

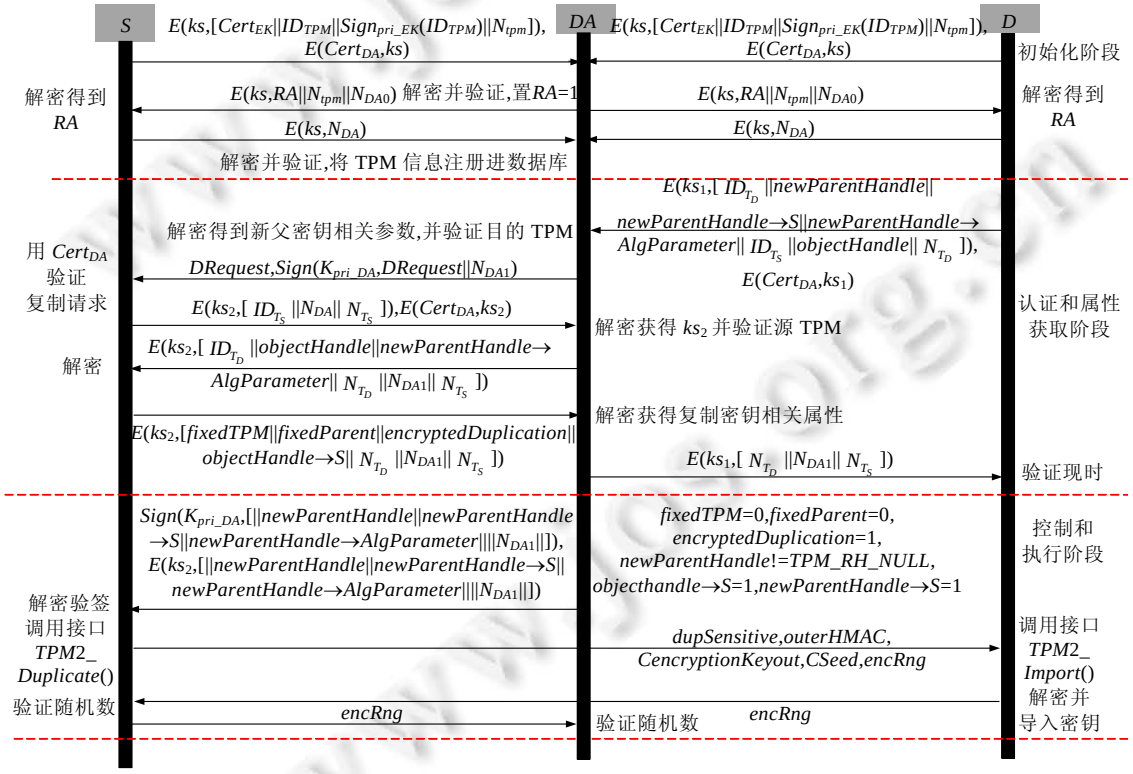

Fig.6 Experimental whole process prototype

图 6 实验流程原型

5.1.2 实验结果

实验主要基于 openssl 和 Microsoft 的 TPM2.0 simulator 实现.

1. 初始化阶段的执行过程.

初始化过程主要完成源 TPM 和目的 TPM 向 DA 注册自己的信息.由于目的 TPM 的初始化与源 TPM 初始化过程完全相同,这里以目的 TPM 的初始化为例.目的 TPM 与 DA 间的初始化阶段流程如图 7、图 8 所示.

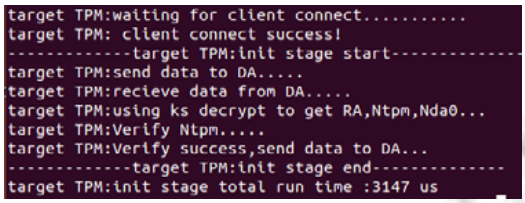

Fig.7 Targe TPM initialization process

图 7 目的 TPM 初始化阶段流程

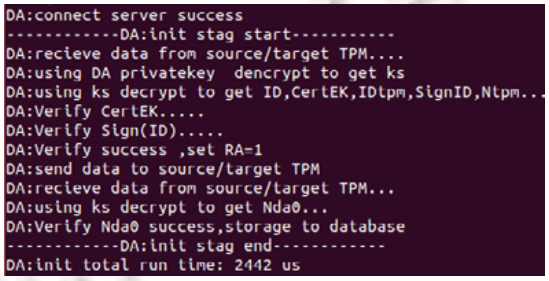

Fig.8 DA initialization process

图 8 DA 初始化阶段流程

2. 认证和属性获取阶段.

认证和属性获取过程主要完成源 TPM 和目的 TPM 间的认证以及向 DA 发送新父密钥和复制密钥的相关属性等.源 TPM、目的 TPM、DA 的认证和属性获取阶段流程如图 9~图 11 所示.

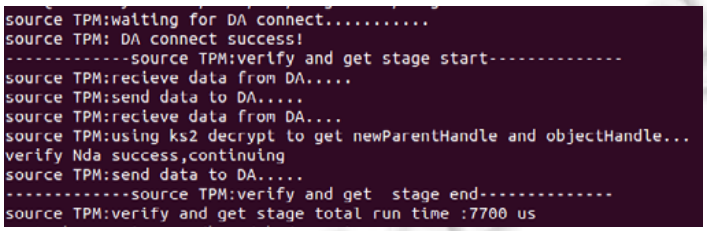

Fig.9 Source TPM authentication and attribute acquisition process

图 9 源 TPM 认证和属性获取阶段流程

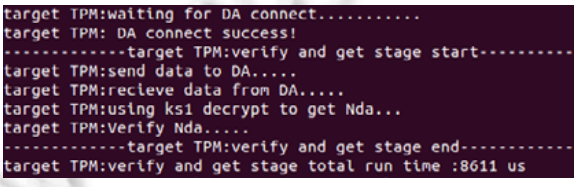

Fig.10 Targe TPM authentication and attribute acquisition process

图 10 目的 TPM 认证和属性获取阶段流程

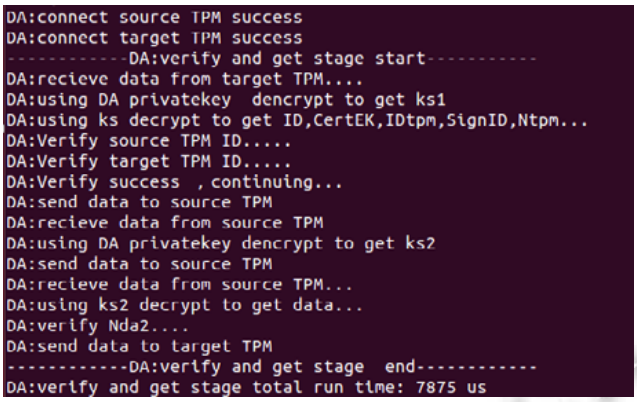

Fig.11 DA authentication and attribute acquisition process

图 11 DA 认证和属性获取阶段流程

3. 控制和执行阶段.

该阶段共有 12 种情况,由于情况 1 和情况 2 的实现过程基本相似,情况 3 和情况 4、情况 5、情况 7、情况 8 的实现过程基本相似,情况 6 和情况 9 的实现过程基本相似,情况 11 和情况 12 的实现过程基本相似,为了减少篇幅,我们仅列出情况 3 和情况 6 的实验结果.这里,我们基于微软 TPM2.0 模拟器提供的复制接口和导入接口来实现.

情况 3:When *fixedTPM*=0,*fixedParent*=0,*encryptedDuplication*=1,*newParentHandle*!=*TPM_RH_NULL*,*objecthandle*→*S*=1,*newParentHandle*→*S*=1.执行过程如图 12~图 14 所示.

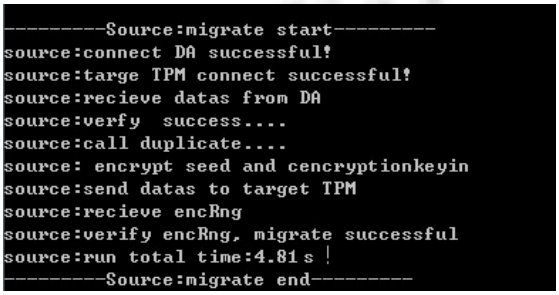

Fig.12 Case 3: Source TPM execution process

图 12 情况 3:源 TPM 执行流程

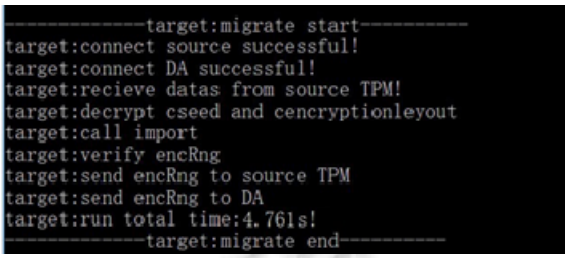

Fig.13 Case 3: Targe TPM execution process

图 13 情况 3:目的 TPM 执行流程

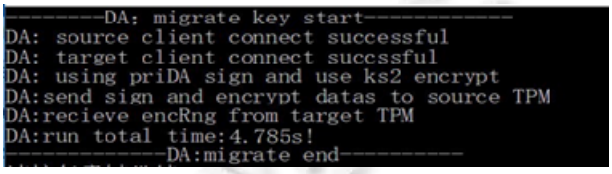

Fig.14 Case 3: DA execution process

图 14 情况 3:DA 执行流程

情况 6:*fixedTPM*=0,*fixedParent*=0,*encryptedDuplication*=1,*newParentHandle*!=*TPM_RH_NULL*, *objecthandle*→*S*=0,*newParentHandle*→*S*=0.执行过程如图 15~图 17 所示.

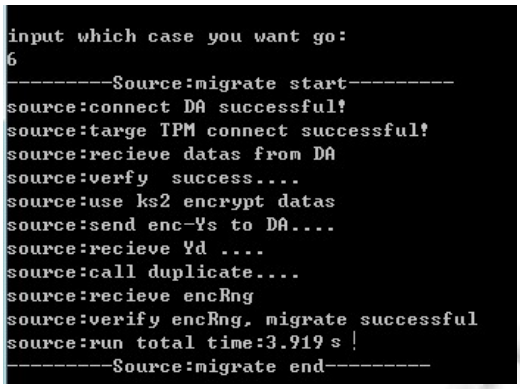

Fig.15 Case 6: Source TPM execution process

图 15 情况 6:源 TPM 执行流程

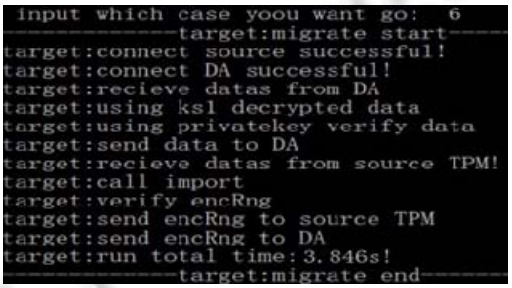

Fig.16 Case 6: Targe TPM execution process

图 16 情况 6:目的 TPM 执行流程

Fig.17 Case 6: DA execution process

图 17 情况 6:DA 执行流程

5.2 性能分析

5.2.1 本协议的性能分析

1. 初始化阶段的执行时间见表 6.

Table 6 Initialization execution time

表 6 初始化阶段执行时间

	source TPM	target TPM	DA
执行时间(s)	0.003	0.003	0.002

2. 认证和属性获取阶段的执行时间见表 7.

Table 7 Authentication and attribute acquisition phase execution time

表 7 认证和属性获取阶段执行时间

	Source TPM	Target TPM	DA
执行时间(s)	0.008	0.009	0.008

3. 控制和执行阶段的执行时间.

由于情况 1 和情况 2 的实验过程相同,情况 3 和情况 5 的实验过程相同,情况 7 和情况 9 的实验过程相同,情况 11 和情况 12 的实验过程相同,我们将实验过程相同的情况的执行时间视为一致.各情况的执行时间如下表 8 所示,其时间性能如图 18 所示.

Table 8 Initialization execution time (s)

表 8 初始化阶段执行时间 (s)

	Source TPM	Target TPM	DA
情况 1/2	0.017	0.011	0.006
情况 3/5	4.81	4.761	4.785
情况 4	4.958	4.834	4.834
情况 6	3.919	3.846	3.846
情况 7/9	3.47	3.464	3.464
情况 8	3.541	3.478	3.471
情况 10	2.317	2.321	2.319
情况 11/12	0.654	0.647	0.59

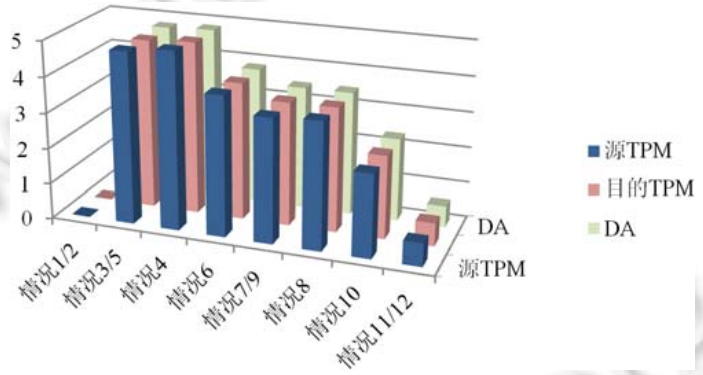

Fig.18 Execution time for each situation in the control and execution phases

图 18 控制和执行阶段各情况执行时间

从图 18 可以看出,情况 4 的执行时间最长.这是因为情况 4 既执行了 *innerwrap* 又执行了 *outerwrap*,同时,由于 *innerwrap* 和 *outerwrap* 的加密密钥和密钥种子在目的 TPM 方产生,在保护传输过程中增加了 T_S 与 DA,T_S 与 T_D 之间的交互.情况 3/5 与情况 4 唯一的不同就是 *innerwrap* 和 *outerwrap* 的加密密钥和密钥种子在源 TPM 方产生,因此执行时间也就稍少于情况 4.而情况 6 中的加密密钥和密钥种子是同一个,所以执行时间应少于情况 3/5.情况 7~情况 10 由于未进行 *outerwrap* 因此执行时间应少于情况 6.由于情况 11/12 既不进行 *innerwrap* 也不进行 *outerwrap* 因此执行时间少于情况 10.而情况 1/2 基本不能进行密钥复制迁移操作,因此时间是最短的.

5.2.2 与 TPM1.2 密钥迁移的性能对比分析

本节主要将本协议的性能与 TPM1.2 密钥迁移性能进行对比分析.由于 TPM1.2 不支持对称密钥,且本文协议的情况较多,为此,我们仅在如下场景下进行对比分析.

具体场景:非对称密钥 *Kmigration* 需要从源平台 *Psrc* 迁移到平台 *Pdes*,在 *Psrc* 端保护 *Kmigration* 的父密钥是 *Kparent*,而 *Pdes* 端选定的将要保护 *Kmigration* 的父密钥是非对称密钥是 *Kstorage*.

对于 TPM1.2,要求 *Kmigration* 是可认证迁移密钥 CMK(certifiable-migration key).本实验需要两台机器,这两台机器的硬件配置与表 5 相同,不同的是软件环境,这两台机器的 OS 是 Ubuntu 16.04,TPM 模拟器为 *TPM_emulator*_0.7.4 模拟器,TSS.net 的版本为 TSS.MSR-master.Source TPM 与 MA 在同一台机器,授权口令以默认的方式在内部产生.这里,我们选用耗时最长的 *TPM_MS_RESTRICT_APPROVE_DOUBLE* 迁移模式.对于 TPM2.0,此场景对应着第 3.3 节中的情况 3,具体的时间开销见表 9.

Table 9 TPM1.2 execution time (s)

表 9 TPM1.2 执行时间 (s)

TPM 版本	Source TPM	Target TPM	MA
TPM2.0	4.81	4.761	4.785
TPM1.2	2.94	0.83	1.23

可以看出,本文协议的时间开销要比 TPM1.2 的大一些.我们认为,主要原因包括两个方面.

(1) 在情况 3 中,源 TPM 既执行了 *innerwrap*,又执行了 *outerwrap*,其复杂性比 TPM1.2 大.

(2) DA 通过与源 TPM 与目的 TPM 交互,获得迁移密钥与新父密钥的属性并控制协议流程,其时间开销要比 MA 仅仅是对目标密钥(新父密钥)进行认证的时间开销大.

性能对比如图 19 所示.

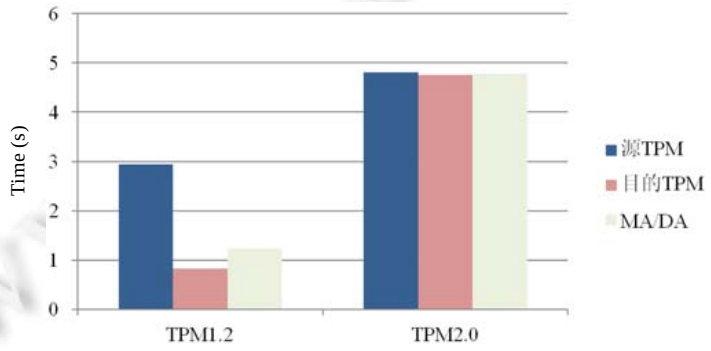

Fig.19 Performance comparison

图 19 性能对比

6 相关工作

在 TPM 密钥迁移方面,到现在为止取得了较多研究成果.TCG 组织一直是这一工作积极的主导者和推动者,TCG 最初在 TPM1.1 规范的密钥迁移模块中提供了 *TPM_AuthorizeMigratinKey*(),*TPM_CreateMigrationBlob*()和 *TPM_ConvertMigrationBlob*()等 3 个 API 接口用于迁移密钥.TPM1.1 的密钥迁移模块优点是简单,但弱点是迁移源和目标之间缺少认证,敌手很容易冒充目标 TPM 而获得迁移密钥的敏感信息.TPM1.2 对迁移模块作了重大改进,引进了认证迁移概念.在认证迁移下,用户可以指定可信第三方迁移权威(migration authority,简称 MA),并保证如果没有 MA 的允许,不可以把密钥迁移到特定的平台.TPM1.2 关于认证迁移密钥(certifiable-migration key,简称 CMK)的 API 包括 *TPM_AuthorizeMigratinKey*(),*TPM_CMK_ApproveMA*(),*TPM_CMK_CreateKey*(),*TPM_CMK_CreateTicket*(),*TPM_CMK_CreateBlob*()和 *TPM_CMK_ ConvertMigration*()等.TPM1.2 的密钥迁移模块优点是安全,缺点是复杂,每次迁移均需要可信第三方 MA 参与,效率低.TPM2.0 取消了 TPM1.2 的密钥迁移接口,并使用新的迁移接口,即 *TPM2_Duplicate*()和 *TPM2_Import*().由于 TPM2.0 改变了密钥属性的表示方式,一个密钥对象是否可以被复制,由 *fixedTPM* 和 *fixedParent* 两个属性组合来决定,当(*fixedTPM*=0,*fixedParent*=0)时,该密钥对象可以被复制实现迁移;当(*fixedTPM*=0,*fixedParent*=1),该密钥对象可以跟随父密钥一起被复制实现迁移.TPM2.0 使用 *TPM2_Duplicate*()进行密钥复制时,需要对被复制密钥进行

innerwrap 或 *outerwrap* 或既进行 *innerwrap* 又进行 *outerwrap*,来保证被复制密钥的机密性、完整性以及认证性.TPM2.0 的密钥迁移模块优点是简单、高效、灵活而且基本安全,可不需要可信第三方参与;但缺点是容易出现配置错误,导致不安全.

在 TPM1.1 和 TPM1.2 密钥迁移机制的分析方面,文献[9]用一阶逻辑语言建立 TPM API 的形式化模型,并对 TPMAPI 进行了全面的逻辑推理分析,其中,对密钥迁移 API 的分析指出,TPM1.1 的弱点在于迁移目标由源 TPM 的 owner 指定,目标 TPM 并不参与迁移,目标 TPM 在接收可迁移密钥时,可迁移密钥有可能已经泄漏,因此具有较大的安全隐患.文献[10]应用 π 演算对 TPM 进行形式化建模,并使用自动定理证明工具 ProVerif 验证其安全属性.作者分析了 TPM CMK(certifiable migratable key)的 *TPM_MS_RESTRICT_MIGRATE* 迁移模式,分析结果表明:若作为第三方的迁移权威(migration authority,简称 MA)用软件处理迁移数据,则敌手能获得被迁移密钥的私钥.作者建议 TPM 规范强制要求 MA 使用 TPM 代替软件处理迁移数据.文献[11]对 TPM 可迁移密钥的安全性进行了分析,指出,TPM 提供密钥迁移机制的同时,降低了可迁移密钥的安全保护强度,敌手能够利用 TPM 的密钥迁移类接口和密钥加载接口破坏 TPM 可迁移密钥的安全性.

在 TPM2.0 密钥迁移机制的分析方面,文献[12]建立了 TPM2.0 保护存储 API 的抽象模型,并利用类型系统证明了 TPM2.0 保护存储的安全性.证明结果表明,TPM2.0 保护存储中的密钥复制类接口是安全的.文献[13]对 TPM2.0 密钥管理 API 的安全性进行了形式化分析,证明了密钥存储和使用类接口能够保证 TPM 不可迁移密钥的安全性,并发现了针对密钥复制类接口的两种攻击.作者提出了该类接口的改进方案,并证明了利用改进的接口实施密钥复制能够保证被复制密钥的安全性.文献[14]分析了 TPM2.0 密钥复制相关流程,对于其中存在的密钥隐私泄露问题进行了改进.在用户不安全复制传输情形下,从 TPM 管理者的角度出发,提出了一套基于 TPM 自身的加密传输协议.通过利用 TPM 自身产生安全密钥,对未受保护的用户敏感数据进行加密,并通过签名的方式保障传输的可靠性.

另外,对于我国 TCM 芯片,文献[15]指出,由该芯片的密钥迁移模块实现的密钥迁移协议存在两个问题.

- 对称密钥不能作为被迁移密钥的新父密钥,违背了 TCM 的初始设计思想;
- 缺少交互双方 TCM 的相互认证,导致源 TCM 的被迁移密钥可以被外部敌手获得,并且敌手可以将其控制的密钥迁移到目标 TCM 中.

针对上述问题,作者提出两个新的密钥迁移协议.

- 协议 1 遵循 TCM 目前的接口规范,以目标 TCM 的 PEK(platform encryption key)作为迁移保护密钥,能够认证目标 TCM,并允许对称密钥作为新父密钥.
- 协议 2 简单改动了 TCM 接口,源 TCM 和目标 TCM 进行 SM2 密钥协商,得到的会话密钥作为迁移保护密钥,解决了上述两个问题,并且获得了前向安全属性.

最后,使用形式化分析方法对上述协议进行安全性分析,结果显示,协议满足正确性和预期的安全属性.

7 结束语

本文分析了《TPM-Rev-2.0-Part-1-Architecture-01.38》国际标准,并总结出使用该标准中的密钥复制接口来实施密钥迁移存在的 3 个问题.

- 其一是缺少交互双方 TPM 的相互认证,会导致密钥能够在敌手和 TPM 间迁移;
- 其二是当迁移密钥的属性 *encryptedDuplication*=0 且新父密钥的句柄 *newParentHandle=TPM_RH_NULL* 时,复制接口不能实施 *innerwrap* 和 *outerwrap*,迁移密钥将以明文传输而造成泄露;
- 其三是当新父密钥是对称密钥时,*innerwrap* 中的对称加密密钥以及 *outerwrap* 中的密钥种子如何在源 TPM 与目标 TPM 之间安全交换,《TPM-Rev-2.0-Part-1-Architecture-01.38》并没有给出具体的解决办法.

针对这些问题,提出了基于 Duplication Authority 的密钥迁移协议.该协议以 Duplication Authority 为认证和控制中心,将密钥迁移过程分为初始化阶段、认证和属性获取阶段以及控制和执行阶段.Duplication Authority

通过判定密钥的复制属性和类型、新父密钥的密钥类型和句柄类型来决定迁移流程.我们考虑了各种合规的属性组合,共设计了 12 种迁移流程.最后,对该协议进行了安全分析和模拟实验验证,结果显示,该协议不仅完全满足《TPM-Rev-2.0-Part-1-Architecture-01.38》规范,而且满足完整性、机密性和认证性.

References:

[1] Feng DG, Qin Y, Wang D, Chu XB. Research on trusted computing technology. Journal of Computer Research and Development, 2011,48(8):1332−1349 (in Chinese with English abstract).

[2] Yu FJ, Zhang HG, Zhao B, Wang J, Zhang LQ, Yan F, Chen ZL. A formal analysis of trusted platform module 2.0 Hash-based message authentication code authorization under digital rights management scenario. Security & Communication Networks, 2016, 9(15):2802−2815.

[3] Zhang HG, Han WB, Lai XJ, Lin DD, Ma JF, Li JH. Cyberspace security review. SCIENCE CHINA Information Sciences, 2016,46(2): 125−164 (in Chinese with English abstract).

[4] Arthur W, Challener D, Wrote; Wang J, *et al*., Trans. A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security. Beijing: Machinery Industry Press, 2017 (in Chinese).

[5] Tan L, Chen J. Remote attestation project of the running environment of the trusted terminal. Ruan Jian Xue Bao/Journal of Software, 2014,25(6):1273−1290 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4414.htm [doi: 10.13328/j.cnki.jos.004414]

[6] Hu LB, Tan L. Research on the trusted virtual platform remote attestation method in cloudcomputing. Ruan Jian Xue Bao/Journal of Software, 2018,29(9):2874−2895 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5264.htm [doi: 10.13328/j.cnki.jos.005264]

[7] Feng DG. Trusted Computing-Theory and Practice. Beijing: Tsinghua University Press, 2013. 5−20 (in Chinese).

[8] Song M, Tan L. Research of key migration protocol which is based on TPM 2. 0 key duplication interface. Journal of Chinese Computer Systems, 2018,39(9):1962−1969.

[9] Chen J. Security analysis of trusted platform module and application [Ph.D. Thesis]. Beijing: Institute of Computing Technology Chinese Academy of Sciences, 2006 (in Chinese with English abstract).

[10] Xu SW, Zhang HG. Formal security analysison trusted platform module based on applied π calculus. Journal of Computer Research and Development, 2011,48(8):1421−1429 (in Chinese with English abstract).

[11] Zhang QY, Zhao SJ, Feng DG. Security analysis and research on TPM migratable key. Journal of Chinese Mini-Micro Computer Systems, 2012,33(10):2188−2193 (in Chinese with English abstract).

[12] Shao JX, Feng DG, Qin Y. Type-based analysis of protected storage in the TPM. In: Proc. of the Information and Communications Security. Springer Int'l Publishing, 2013. 135−150.

[13] Zhang QY, Zhao SJ, Qin Y, Feng DG. Formal analysis of TPM2.0 key management APIs. Chinese Science Bulletin, 2014,59(32): 4210−4224.

[14] Xu Y, Zhao B,·Heinayati M, Yu FJ. Security enhancement of key duplication in TPM2.0. Journal of Wuhan University (Natural Science Edition), 2014,60(6):471−477 (in Chinese with English abstract).

[15] Zhang QY, Feng DG, Zhao SJ. Design and formal analysis of TCM key migration protocols. Ruan Jian Xue Bao/Journal of Software, 2015,26(9):2396−2417 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4719.htm [doi: 10.13328/j.cnki.jos.004719]

附中文参考文献:

[1] 冯登国,秦宇,汪丹,初晓博.可信计算技术研究.计算机研究与发展,2011,48(8):1332−1349.

[3] 张焕国,韩文报,来学嘉,林东岱,马建峰,李建华.网络空间安全综述.中国科学:信息科学,2016,46(2):125−164.

[4] Arthur W, Challener D,著;王鹃,等,译.TPM2.0 原理及应用指南:新安全时代的可信平台模块.北京:机械工业出版社,2017.

[5] 谭良,陈菊.一种可信终端运行环境远程证明方案.软件学报,2014(6):1273−1290. http://www.jos.org.cn/1000-9825/4414.htm [doi: 10.13328/j.cnki.jos.004414]

[6] 胡玲碧,谭良.云计算中可信虚拟平台的远程证明方案研究.软件学报,2018,29(9):2874−2895. http://www.jos.org.cn/1000-9825/5264.htm [doi: 10.13328/j.cnki.jos.005264]

[7] 冯登国.可信计算——理论与实践.北京:清华大学出版社,2013.

[8] 宋敏,谭良.TPM2.0 密钥迁移协议研究.小型微型计算机系统,2018,39(9):1962−1969.

[9] 陈军.可信平台模块安全性分析与应用[博士学位论文].北京:中国科学院计算技术研究所,2006.

[10] 徐士伟,张焕国.基于应用π演算的可信平台模块的安全性形式化分析.计算机研究与发展,2011,48(8):1421−1429.

[11] 张倩颖,赵世军,冯登国.TPM 可迁移密钥安全性分析与研究.小型微型计算机系统,2012,33(10):2188−2193.

[14] 徐扬,赵波,米兰·黑娜亚提,余发江.TPM2.0 密钥复制安全性增强方案.武汉大学学报(理学版),2014,60(6):471−477.

[15] 张倩颖,冯登国,赵世军.TCM 密钥迁移协议设计及形式化分析.软件学报,2015,26(9):2396−2417. http://www.jos.org.cn/1000-9825/4719.htm [doi: 10.13328/j.cnki.jos.004719]

谭良(1972−),男,四川泸州人,博士,教授,主要研究领域为可信计算,信息安全,云计算及大数据处理,区块链.

宋敏(1994−),女,硕士,主要研究领域为可信计算,网络安全.