

### 4.1 概述

本书 2.2 节介绍了可信密码服务机制。它能够依托可信根，提供具备可信特征的多种密码服务功能，包括可信度量、可信存储和可信身份验证/报告等功能。这些可信密码服务功能主要是在可信软件基的可信支撑机制中实现的。而可信根对这些可信密码服务的功能支持主要是通过可信根中的可信平台密码模块实现的。

我国可信计算标准为可信平台密码模块提出了 TCM 标准，而国际上提出的 TPM 标准仍旧是可信计算 2.0 概念的可信根，但它已具备比较完整的可信平台密码服务功能，因此，也可以作为可信 3.0 中的可信平台密码模块使用。TPM 和 TCM 均可作为密码平台支持可信计算，但在国内实际应用时，需注意遵循我国相关密码标准和可信计算标准的规定，使用符合国内相关要求的可信密码产品和可信密码算法。

本章介绍可信密码服务的工程实现方法，其整体框架具有通用性，在具体实现时，则遵循国内的两项 TCM 标准，分别是 GM/T 0012—2012《可信计算-可信密码模块接口规范》和 GM/T 0013—2012《可信计算-可信密码模块接口符合性测试规范》。目前遵循这些标准的 TCM 芯片并不是非常普及，并且在实际 TCM 芯片上进行开发，需要的资源支持较多，且芯片作为一个黑盒子，在开发过程中难以调试。为方便 TCM 上的开发工作，推动国内可信计算的发展，北京工业大学可信计算国家重点实验室在 cube 架构基础上开发了一款 TCM 模拟器 cube-tcm，并开发了配套的测试工具。本章的工程化内容主要是在这款模拟器以及配套测试工具上实现的。

目前，模拟器和测试工具已开源发布，它依赖于一个国密算法软件包的裁剪

版（源自国人的一个国密算法开源项目[https://github.com/stevenpsm/GM\\_SM2](https://github.com/stevenpsm/GM_SM2)，使用了国外的一个大数算法软件包 `libtommath`），以及笔者开发的开源可信计算框架 Cube-1.3 版。tcm 源码以及修改的密码库源码地址在可信计算实验室的 github 账号<https://github.com/TCLab-BJUT>上。读者到 github 上下载该软件以及它所依赖的国密算法软件包，并下载 cube-1.3 版本，即可自行搭建开发环境，验证本章所讲述内容，并可在此基础上自行开发。

## 4.2 可信根访问模式

可信密码服务，是指系统通过调用可信根中的密码学功能，实现的可信存储、可信度量、可信报告等功能。执行可信密码服务功能时，系统中应当有一个可信密码服务功能的调用者，它通过与可信根的交互来实现具体的可信密码服务功能。调用者可以是可信硬件、BIOS、引导程序、内核程序或者是应用程序。

### 4.2.1 可信根操作分类

可信 2.0 中，可以通过远程调用方式访问可信根，因为访问可信根的应用有可能在远端执行。在可信 3.0 中，计算节点对远端节点可信根的访问行为应当通过两节点间可信软件基之间的交互来完成，因此，访问模式应当是远程节点的可信软件基对本地节点可信软件基提出请求，本地节点可信软件基根据请求，主动访问可信根，通过对可信根的本地调用获得远程节点所请求的数据并返回。这里并不需要远程调用行为。因此，本章中我们只考虑本地调用者访问可信根的情况。

本章主要关注的是可信密码机制相关部分，为简化讨论内容，本章中除特殊说明外，均只考虑调用者是应用程序的情况。读者可自行将情况推广到其它调用者。

在讨论可信密码服务之前，我们首先对可信密码服务执行环境的安全和可信状况提出一些假设。只有这些假设成立时，可信密码服务的执行才是相对可信的。而为了保证可信密码服务的可信性，我们应当在系统实现时，通过软硬件的安全可信机制尽量逼近这些假设。

可信密码服务的安全性假设如下所列：

- 1. 可信根（TCM）内部的数据受到物理保护，可在较长时间内保证安全，而可信根内部的流程也可以不受外界干扰而运行。
- 2. 可信根初始化的过程应当是在干净的系统 and 相对隔离的环境中进行，这一过程可以保证其安全性。
- 3. 调用者在其生命周期中，可以保证其内部私有数据(如应用的内存数据)的相对保密性。这主要是通过系统内部的进程隔离等保护机制保障的。
- 4. 调用者在输入口令时，口令的明文形式在输入后立刻计算出摘要值，并随即清理掉。执行程序的保护措施和口令明文存在的短暂性使得这一过程也可以认为是相对安全的。

除此之外，磁盘上的数据，网络、总线上传输的数据，其保密性和完整性均不能保证。

调用者访问可信根时，一般以可信根内部的实体作为访问对象。可信根内部实体包括 PCR 寄存器、hash 对象等非授权实体，以及密钥、TCM 所有者、数据、计数器、NV 索引等授权实体。表 4-1 列出了 TCM 中常用的实体类型。

表 4-1 TCM 常用实体类型

实体名称	授权与否	实体说明
密钥	是	包括对称密钥（SM4 算法）和非对称密钥（SM2 算法），密钥的敏感部分（对称密钥，非对称密钥私钥，授权数据）仅在 TCM 中以明文方式出现，导出时使用封装密钥加密。
NV 索引	是	非易失性空间 NVRAM 存储器索引，用来存放少量敏感信息。
PCR	否	用于存放哈希值的寄存器，被保留用于保存系统的引导序列度量结果的完整性，可以被用来建立这个系统的信任等级
Hash	否	用来辅助执行外部的 HASH 操作，使用 SM3Hash 算法。
TCM 所有者	是	用来标识 TCM 本身的管理权限。
数据	是	通过 TCM 密封或绑定的数据
计数器	是	TCM 中的一个单调计数器，可用于防范重放攻击

授权实体在可信根内部存储了授权数据，调用者访问授权实体时，需首先在可信根和调用者间建立起会话，通过验证调用者输入的授权数据，确认调用者获得了授权，并利用会话来保障调用者和可信根之间通信的安全性。

可信根中有几个关键密钥实体，包括背书密钥（EK）、存储根密钥（SMK）和用户身份密钥（PIK）。

背书密钥 EK 同时也是可信报告根，它可以自行生成，也可以在可信的环境中由外部注入。一般背书密钥由可信根的生产方和管理方生成，本书假设可信根中已经生成了背书密钥。

存储根密钥 SMK 是节点的可信存储根，它一般是由可信根的属主在未授权的可信根上通过 `takeownership` 命令生成的。`Takeownership` 会同时生成可信根属主授权数据和 SMK 授权数据。在可信根中，同时只能有一个 SMK。

用户身份密钥 PIK 是可信根的使用用户用来证明自己身份和生成可信报告的密钥，它的生成需要可信根属主的授权和可信第三方的介入，可信第三方将为其生成一个可信证书，证书由可信第三方用自己的私钥签发，其中包含用户身份信息和 PIK 的摘要值，其它信任可信第三方的用户可以来通过该可信证书来确认 PIK 与用户身份的绑定关系。一个可信根可以同时支持多个 PIK 密钥。

图 4-1 表示可信根中 EK、SMK 和 PIK 之间的关系。

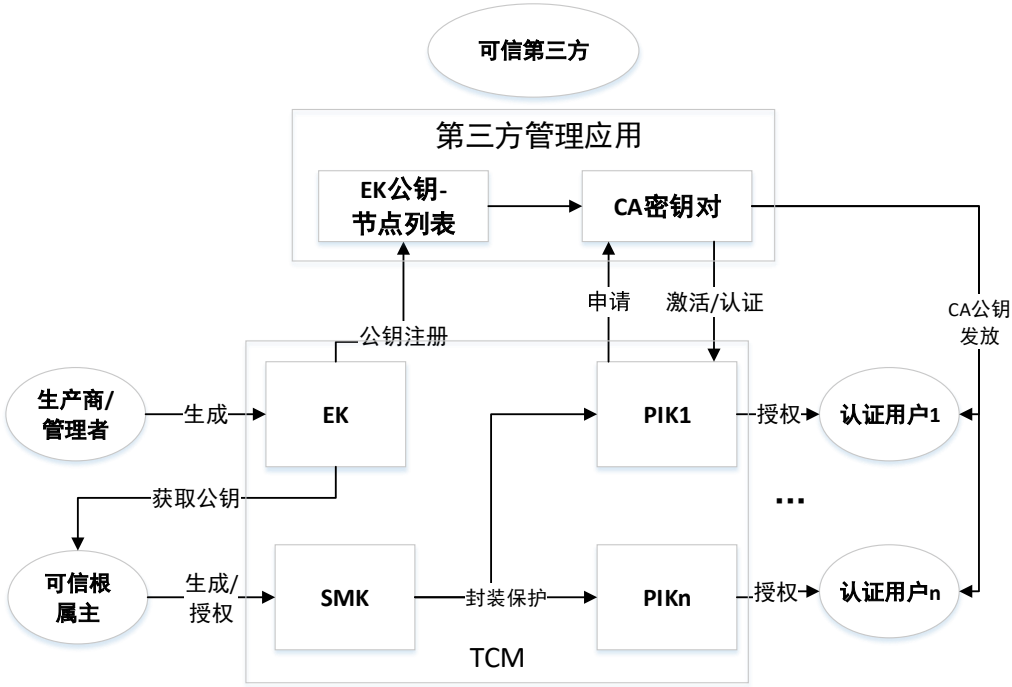


图 4-1 TCM 中 EK、SMK 和 PIK 之间的关联关系图

可信根的操作可以分为以下几类：非授权操作、可信管理操作、可信存储操作和可信报告操作。

非授权操作是指不需要授权的可信根访问操作，包括对可信根中 PCR 寄存器的读取和扩展、可信根状态查询、摘要值计算以及背书密钥公钥读取等。大部分非授权操作可以在任意时刻进行，而背书密钥公钥读取操作则需在系统初始化生成背书密钥对后才可操作。

可信管理操作则是用来在可信根中生成和管理 EK、SMK、PIK 等关键实体和构建调用者与可信根间会话的操作。常用的可信管理操作包括 `takeownership`、`apcreate` 和 `apterminate`、`makeidentity` 和 `activeidentity`。这些命令及其功能如表 4-2 所示：

表 4-2 TCM 常用管理命令介绍

可信管理命令	作用	前置要求	操作结果	备注
Takeownership	获取所有权	调用者需先通过 <code>readpubek</code> 读取背书密钥公钥，需通过 <code>apcreate</code> 命令建立与 <code>ET_NONE</code> 的连接	生成存储根密钥 SMK，为可信根的属主和存储根密钥设置口令	必须清除所有权才能第二次执行
Apcreate	建立调用者和可信根中实体会话	需输入实体的授权数据（与 <code>ET_NONE</code> 建立会话时不需要授权数据）	可信根完成对调用者授权的验证，可信根与调用者获得共享秘密，并返回一个会话句柄	
Aptermminate	终止一个可信会话	需获取已有会话的句柄	终止一个会话，释放会话句柄。	
Makeidentity	创建平台身份验证	需已通过 <code>takeownership</code> 获得了所有权，并获得了可信第三方的可信公钥	创建一个平台身份验证密钥。并生成一个包含验证密钥公钥、可信第三方可信公钥和用户附加的身份信息的身 份激活包	

Activeidentity	激活平台身份	可信第三方已经根据身份激活包		
----------------	--------	----------------	--	--

可信存储操作通过可信根中生成的密钥来实现数据保护功能，这类命令需要在 SMK 生成后才能执行。因为可信根中生成的密钥需要另一个密钥作为封装密钥，而 SMK 则是第一个封装密钥。

可信报告操作则向外界提供可信根内部状态的可信报告，这类命令需要依托于一个特定的 PIK 进行。可信根内部使用该 PIK 的私钥签发关于可信根内部状态的报告，报告的接收者则可以使用 PIK 公钥来验证报告的可信性，并通过 PIK 证书来验证 PIK 的可信性。

### 4.2.2 可信会话原理介绍

调用者对 TCM 中授权实体的访问必须通过可信会话来进行。可信会话通过 apcreate 命令来建立，通过 apterminate 命令结束。可信会话完成对象的授权验证，并为可信根内部和调用者间创建与该实体关联的共享秘密信息。每个可信会话有自己的会话句柄值、序号和对话创建时产生的随机数。这些信息以及共享秘密信息在调用者执行与授权实体相关的访问命令时，用来计算这些访问命令的授权验证码。调用者和 TCM 内部通过授权验证码的计算和对比来对这些访问行为进行验证，以确保访问行为的可信性。

下面我们分别通过对 apcreate 命令 TCM 端操作、调用者端操作以及一般命令中会话验证的过程来介绍 TCM 可信会话的原理。

#### 4.2.2.1 apcreate 命令介绍

本小节介绍 TCM 可信会话建立时，TCM 端的操作。

在我国可信计算标准中，如图 4-2 所示，规定了 apcreate 命令的输入格式和输出格式：

输入数据格式

标识	数据长度	命令码	实体类型	实体值	调用者 nonce	实体授权数据 验证码
2B	4B	4B	2B	4B	32B	32B

输出数据格式

标识	数据长度	返回码	授权会话 句柄	TCM nonce	序列号	实体授权数据 验证码
2B	4B	4B	2B	32B	4B	32B

图 4-2 apcreate 命令输入输出数据格式

下面我们分输入部分、执行部分和返回部分来介绍该命令。

1. 输入部分

输入数据格式中，前三项为每一条命令都需要的公共头部，给出了描述命令授权验证数的标识、命令的长度，以及每个命令唯一对应的命令码。第四项则是 apcreate 命令的会话对象类型。Tcm 中可供选择的会话对象类型如表 4-3 所示：

表 4-3 apcreate 命令可选用的实体类型

类 型值	类型内容	类型代号	句柄值
0x01	密钥句柄	TCM_ET_KEYHANDLE	
0x02	TCM 所有者	TCM_ET_OWNER	0x40000001
0x03	数据	TCM_ET_DATA	
0x04	存 储 根 主 密 钥 SMK	TCM_ET_SMK	0x40000000
0x05	密钥	TCM_ET_KEY	
0x06	可撤销 EK	TCM_ET_REVOKE	0x40000002
0x0A	计数器	TCM_ET_NONE	
0x0B	NV 索引	TCM_ET_NV	
0x11	对称密钥	TCM_ET_SM4	
0x12	无对象(用于获取 用户授权之前)	TCM_ET_NONE	
0x13	授权数据标识	TCM_ET_AUTHDATA_ID	

0x40	保留	TCM_ET_RESERVED	
------	----	-----------------	--

实体值为实体对应的句柄。TCM 中为每个实体分配一个 4 字节的密钥句柄，其中部分特殊实体（EK,SMK 等）有固定的实体值，其它实体的句柄则是在创建或载入过程中分配的一个 4 字节随机数，且与 TCM 已有句柄值不同。本章中使用的实体值包括 0x01、0x04 和 0x12 三种。

调用者 nonce 为应用程序生成的随机数，用来保证会话的唯一性。

最后一个输入变量是实体授权数据验证码，访问授权实体的命令一般都需要生成授权数据验证码。授权数据验证码的计算过程在下一节中详细说明。调用者首先完成授权数据验证码的计算，并将授权数据验证码填入输入命令的最后，TCM 收到数据后，根据命令数据重新计算授权数据验证码，并与命令最后附带的授权数据验证码比较，如一致则通过该命令，否则认为命令的数据遭到了破坏。

在 apcreate 命令中，当为 TCM\_ET\_NONE 实体创建会话时，授权数据验证码仅仅起到对命令进行校验值计算的作用。访问其它实体时，用户授权数据也会参与授权数据验证码的计算，此时授权数据验证码同时也起到了用户授权数据验证的作用。

## 2. 执行部分

根据中华人民共和国密码行业标准 GM/T 0012-2012《可信计算——可信密码模块接口规范》的规定，TCM\_APCreate 命令应针对实体类型为 TCM\_NONE 和非 TCM\_NONE 两种不同情况，以不同的方式建立授权会话。

实体类型为 TCM\_NONE 的会话建立行为一般用在 takeownership 命令之前，此时 TCM 尚未进行用户授权行为，TCM 中仅有 EK 这一授权实体，并且尚未确定授权数据。此时 Apcreate 命令直接创建会话和相应的会话授权句柄，生成防重放攻击的会话初始序列号，，并生成授权验证码。

否则，apcreate 就是与已创建并生成授权数据的 TCM 中实体建立授权数据。此时 apcreate 命令按照以下步骤执行：

1. TCM 验证调用者对实体的授权数据，并通过校验码验证输入参数的完整性。
2. TCM 内部生成随机数 TCM\_Nonce，并基于共享的授权数据 authData、调用者生成的随机数 callerNonce 和 TCM 生成的随机数 TCMnonce 来创建共享密码数据  $sharedSecret=HMAC(authData,callerNonce||TCMNonce)$  和共享会话密钥  $sessionKey=KDF(sharedSecret)$ 。
3. 创建会话和相应的授权会话句柄。这就是在 TCM 内部建立一个会话数据结构，并为这一结构分配一个不与当前其它句柄值重合的 4 字节随机数



作为会话的句柄。会话数据结构在标准中并未给出，可由生产厂商自行定义，但一般应包括会话句柄、重放序列号，实体类型，共享秘密信息以及会话密钥等内容。在 cube-TCM 模拟器中，会话数据结构如下定义：

/\* NOTE: Vendor specific \*/

```
typedef struct tdTCM_SESSION_DATA {
    /* vendor specific */
    TCM_AUTHHANDLE SERIAL;
    TCM_AUTHHANDLE handle;          /* Handle for a session */
    TCM_ENT_TYPE entityTypeByte;
    TCM_NONCE nonceEven;
    TCM_SECRET sharedSecret;        /* OSAP */
    TCM_DIGEST entityDigest;
    TCM_BOOL valid;                  /* added kgold: array entry is valid
    */
}__attribute__((packed)) TCM_SESSION_DATA ;
```

4. 生成一个防重放攻击的初始序列号，该序列号可以按序生成，也可以随机产生。
5. 根据输出内容，生成输出的授权数据验证码。

授权数据验证码的生成分为两步，第一步是选取输入/输出命令中需校验的一些参数，计算其 SM3 Hash 值，第二步则是使用授权数据作为 HMAC 算法的密钥输入，以第一步生成的 Hash 值再补充需校验的其它参数作为数据输入，进行 HMAC 计算，计算结果即为授权数据验证码，这一计算过程既保证了只有和 TCM 有共享秘密的调用者才能正确进行校验码计算，也验证了命令/返回数据中重要参数的完整性。TCM 中，授权数据是和实体一起存放在 TCM 内部。Acreate 命令中，命令码和返回码的计算过程如图 4-3 和图 4-4 所示：

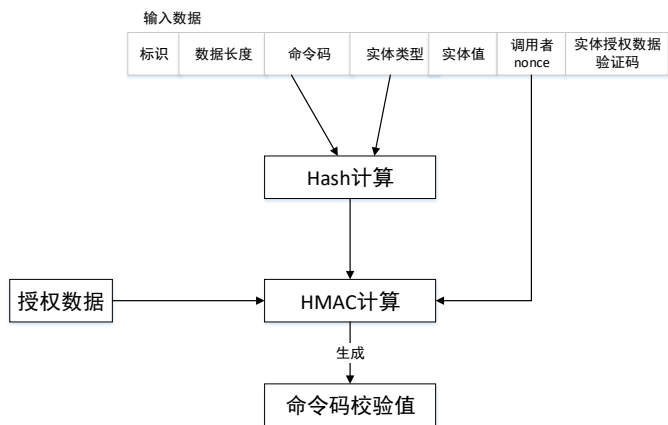


图 4-3 apcreate 命令输入的实体授权验证码计算过程

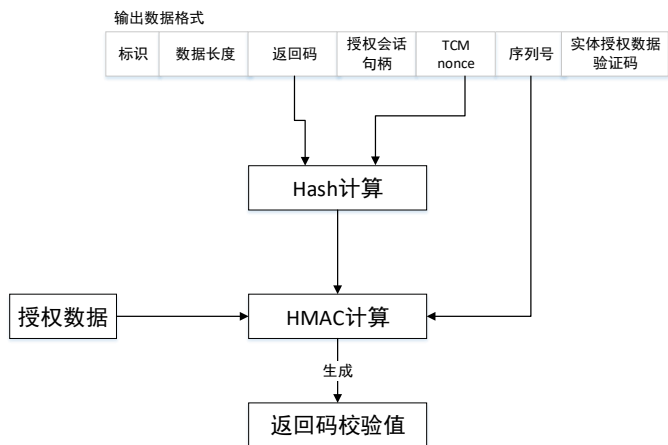


图 4-4 apcreate 返回数据的实体授权验证码计算过程

### 3. 输出部分

输出数据格式中，前三项为每一条返回数据都需要的公共头部，给出了描述命令授权验证数的标识(`apcreate` 命令该标识为 `TCM_TAG_RSP_AUTH1_COMMAND`,表示有一个校验数的返回值)、返回数据的长度，以及本命令的返回码（一般 0 表示正常返回，非 0 表示返回错误信息）。第四项开始为不同命令特有的返回数据。.`apcreate` 命令中，第四项为所创建会话的授权会话句柄，第五项为会话过程中 TCM 内部生成的随机数，第六项为会话的序列号，最后一项则为实体授权数据验证码。

#### 4.2.2.2 apcreate 调用者端操作介绍

与普通的函数调用不同，TCM 的调用者在调用前后需要进行很多具体的操作。调用者的工作可以分成两大部分：命令生成和返回值处理。在命令生成时，调用者需生成调用的相关参数，并将参数组装成符合 TCM 要求的数据包，再将数据包发送给 TCM。在返回值处理时，调用者将收到的 TCM 数据包解包并提取参数值，而后进行校验和后续处理工作。

调用端为了维持与 TCM 的安全通信，也需要建立和维持会话列表。可以仿照 TCM 内部机制，建立起一个 TCM 会话数据结构的列表，并在其中存放会话相关的信息。如在 cube-TCM 中，我们在调用端建立了一个 TCM\_SESSION\_DATA 结构的链表。而调用端的 apcreate 命令主要的任务就是为实体建立一个会话数据结构并添加到列表中。

下面分别介绍调用端执行 apcreate 时的命令生成过程和返回值处理过程。

调用端在执行 apcreate 命令时，应执行如下一些操作：

1. 创建一个会话数据结构，并将这一结构添加到调用端的会话列表中。
2. 生成调用者随机数 CallerNonce。
3. 根据用户输入的实体口令计算实体的授权数据 AuthData。
4. 根据用户输入的实体类型、取值等信息，填充会话数据结构的内容和构建 TCM 的命令输入数据结构，并根据图 4-3 所示过程计算输入命令的实体授权数据验证码。
5. 序列化命令输入数据结构，生成 apcreate 的输入数据流，并写入可信根中。

TCM 收到调用端生成数据后，执行 TCM 内部操作并返回 apcreate 执行结果，调用端收到返回值后，则需按照如下顺序执行操作：

1. 反序列化收到的返回值数据流，生成返回值数据结构。
2. 根据图 4-4 所示过程，计算返回命令的实体授权数据验证码，并与命令中的验证码比较，如一致，则确认返回命令的完整性。否则认为返回数据被破坏并退出。
3. 获取输入时生成的会话数据结构，根据标准中 apcreate 命令的同样操作流程，计算会话的共享密码数据和共享会话密钥。
4. 将共享密码数据、共享会话密钥以及返回数据结构所提供的会话句柄、序列号等数据填充到会话数据结构中。
5. 返回命令返回值和会话的句柄。

此时，调用者的授权信息已得到验证，会话已经建立，并且调用者和 TCM 内部已有针对访问实体生成的会话句柄，并有与会话句柄关联的共享秘密信息，可以用来维持双方之间的后续通信。而 apcreate 命令返回的句柄则需记录下来，

调用者在后续对授权实体的访问过程中需要多次使用这一句柄。

4.2.2.3 实体授权访问时的可信会话应用

调用者在和 TCM 之间建立起可信会话之后，会在对实体的授权访问过程使用这一会话。TCM 命令包括无授权的命令、需要一个实体授权的命令和需要两个实体授权的命令，后两种命令都需要预先建立会话。TCM 命令码和返回码最开始的命令标识就是用来标识命令的授权情况。表 4-4 给出了 TCM 命令标志和授权状况之间的对应关系。

表 4-4 TCM 命令标志列表

标志位取值	标记名称	标记描述
0x00C1	TCM_TAG_RQU_COMMAND	没有授权的命令
0x00C2	TCM_TAG_RQU_AUTH1_COMMAND	需要一个授权的命令
0x00C3	TCM_TAG_RQU_AUTH2_COMMAND	需要两个授权的命令
0x00C4	TCM_TAG_RSP_COMMAND	没有授权的命令返回
0x00C5	TCM_TAG_RSP_AUTH1_COMMAND	需要一个授权的命令返回
0x00C6	TCM_TAG_RSP_AUTH2_COMMAND	需要两个授权的命令返回

大多数命令只有一种标志位，setcapability, nv\_definespace,nv\_readvalue 和 nv\_writevalue 等命令则有零授权和一个授权两种情况，也就是说它们的命令有两种取值方式。而密钥证明命令 certifykey 则有零授权、一个授权和两个授权三种情况。

零授权时，命令不使用可信会话中的信息。

一个授权时，该命令应关联一个已授权实体，并且需在执行该命令之前，完成与该实体间的可信会话建立。此时，命令和返回数据最后一般应添加 36 字节内容，前 4 字节为可信会话的授权会话句柄，后 32 字节则是使用会话中的共享秘密信息作为 HMAC 算法的密钥，采用与图 4-3 和图 4-4 类似的计算过程算出的授权数据验证码，Hash 算法和 HMAC 算法的输入数据选择则根据命令不同而有所不同。

两个授权时，该命令应关联两个已授权实体，并且需在执行该命令之前，分别完成与这两个实体间的可信会话建立。此时，命令和返回数据最后一般应添加 72 字节内容，分为两个 36 字节数据段，分别与两个实体的可信会话关联。36 字节的分割方式和算法九三两个前 4 字节为可信会话的授权会话句柄，后 32 字节则是使用会话中的共享秘密信息作为 HMAC 算法的密钥，采用与图 4-3 和图 4-4 类似的计算过程算出的授权数据验证码。

调用者和 TCM 在为实体建立了可信会话之后，通过在调用端和 TCM 中分别

进行的授权数据验证码的计算,来防范攻击者通过干扰双方的通信而进行的攻击,保障调用者和 TCM 之间的安全数据交换。

### 4.3 模拟可信根环境部署

一个平台要实现可信密码服务,首先必须具备可信根。目前,可信根并未普及所有平台上。并且,即使平台具备可信根,在开发过程中直接改动可信根也很容易导致系统被锁死。因此,开发可信计算功能时,依托一个模拟环境进行开发,开发产品在模拟环境中可以稳定运行后,再移植到真实环境下进行开发,是一个比较适当的选择。软件模拟可信根难以模拟与硬件相关的可信计算功能,但系统启动之后,对硬件可信根和模拟可信根的访问接口在驱动程序层面并无区别。

本书采用北工大可信计算实验室在 cube 架构下开发的 cube-tcm 模拟可信根提供可信密码服务功能的示范。因此,本书的主要工程实践内容均可以移植到具备硬件可信环境的可信平台上。

#### 4.3.1 cube-tcm 设计思想

Cube-tcm 是北工大可信计算实验室开发的可重构可信根模拟器的 tcm 版。此前有 tpm-emulator, swtpm 等针对 tpm 的开源模拟器,微软则实现了 tpm2.0 的开源模拟器,国内也有厂家在做 tcm 的模拟器。这些模拟器都是针对特定的可信根规范进行模拟。

我们认为,可信根模拟器除了模拟可信根的行为之外,还应有两个重要的功能:内部跟踪分析功能和可信根扩展开发功能。

可信根模拟器的内部跟踪分析功能主要针对基于可信根开发上层应用的开发者,要求对可信根内部运算结果进行跟踪分析。因为物理可信根为保证其安全性,对外呈现一个黑盒状态,因此在开发过程中调试起来很困难,可信根模拟器可以为开发者提供可信根内部计算的过程和结果,并为可信根内部运算提供数据重定向输出等功能,以便开发者在出现问题时获取信息,迅速定位错误,为开发者提供有效的调试跟踪手段。

可信根模拟器的可信根扩展开发功能主要针对可信根的设计制造者,要求可信根模拟器能够方便地修改数据结构、添加和删除功能甚至对内部进行重构。因为可信根标准并未限定可信根的所有功能,不同场景下,根据需求不同,可信根可能会需要重新设计以增加特殊功能或者对已有功能有所裁剪,如云平台下的可

信根需要对虚拟化可信根的支持，物联网环境下感知端为满足低功耗要求可能需要裁剪的可信根。而便于修改的可信根模拟器则可以预先模拟可信根制造者的设计方案，并在其基础上进行开发，以验证设计方案的可行性，并可以在所设计的可信根尚未生产时就开始其上应用的开发工作，以提高开发效率。

考虑到上述两项要求，我们提出了可重构可信根模拟器的设想，对其提出了三项要求：

1. 可重构可信根模拟器可以方便地改造其结构，使其可以支持对不同可信根的模拟，也可以在已有可信根架构基础上方便地进行修改和扩展。
2. 可重构可信根模拟器应便于移植，可以在不同的平台（包括嵌入式系统和内核环境下）便捷地移植。
3. 可重构可信根模拟器应便于跟踪调试，可以为外界提供内部信息流的监控接口，可以监控模拟器内部的数据变化，也可以对可信根的互操作进行追踪。

我们的实现思路则是依托 cube 架构来实现可重构可信根模拟器，利用 cube 架构中软件定义数据结构能力和模块化组装能力，提供可重构可信根模拟器的重构和改造能力；依托 cube 架构的可移植性，并提供不依赖于外部库的国密算法，实现可重构可信根模拟器的可移植性；利用 cube 架构消息传输的软件定义能力，为可重构可信根模拟器的跟踪调试提供方便。

基于上述思路，我们实现了 tcm 的模拟器 cube-tcm，以及与模拟器配套的调用测试工具，并实现了它的虚拟化可信根版本。代码已上传到 github 网站上。本书只介绍 tcm 模拟器及其调用测试工具。

## 4.3.2 cube-tcm 组成

### 4.3.2.1 cube-tcm 源码目录结构

cube-tcm 模拟器及调用测试工具其源码均在 cube-tcm 的工作目录下。表 4-5 列出了 cube-tcm 工作目录下的各项内容。

表 4-5 cube-tcm-1.12 工作目录

路径名称	对象类型	对象内容
set_env.sh	脚本	环境变量设置脚本
env_build.sh	脚本	初始化环境设定脚本
include	头文件目	模拟器源码头文件

		录	
plugin		二进制目录	子模块二进制库文件目录
module		父目录	子模块源码目录
	auth	源码目录	模拟器认证子模块源码
	key	源码目录	模拟器密钥子模块源码
	nv	源码目录	模拟器非易失存储子模块源码
	state	源码目录	模拟器状态处理子模块源码
	pcr	源码目录	Tcm 模拟器的 PCR 寄存器计算部分子模块源码
	vtcm_store	源码目录	模拟器本地数据存储部分子模块源码
	vtcm_port	源码目录	模拟器数据端口部分子模块源码
	vtcm_utils	源码目录	tcm 命令生成部分子模块源码
	vtcm_input	源码目录	Tcm 互动输入子模块源码
	vtcm_auto	源码目录	Tcm 自动化输入子模块源码
	vtcm_switch	源码目录	
	vtcm_hub	源码目录	
	vtcm_manage	源码目录	
	Tpmd_dev	源码目录	
locallib		父目录	Tcm 模拟器本地库文件
	bin	二进制目录	本地库二进制文件存放位置
	src	父目录	本地库源码存放位置
	auth	源码目录	认证库函数源码
	io	源码目录	io 库函数源码
Init_module		父目录	实例的初始化模块。
	vtcm_init	源码目录	Tcm 模拟器初始化模块的源码
vtcm_emulator		实例目录	tcm 模拟器执行环境目录
vtcm_utils		实例目录	访问 tcm 模拟器的应用示例执行环境
vtcm_hub		实例目录	
vtcm_manager		实例目录	

这里的四个实例目录是四个 cube 实例，vtcm\_emulator 用来启动 tcm 可信根模拟器，vtcm\_utils 用来启动可信根调用测试程序，它是一个可信根调用者程序。本章中只使用这两个实例，而 vtcn\_hub 和 vtcn\_manager 两个实例则是用于虚拟

可信根的实现，本章不做讨论。module、loclib 和 init\_module 则为模拟器和调用测试程序提供模块和函数库。

下面分别介绍模拟器 vtcem\_emulator 和调用测试程序 vtcem\_utils 的原理。

4.3.2.2 cube-tcm 模拟器原理

cube-tcm 模拟器其内部结构如图 4-5 所示。

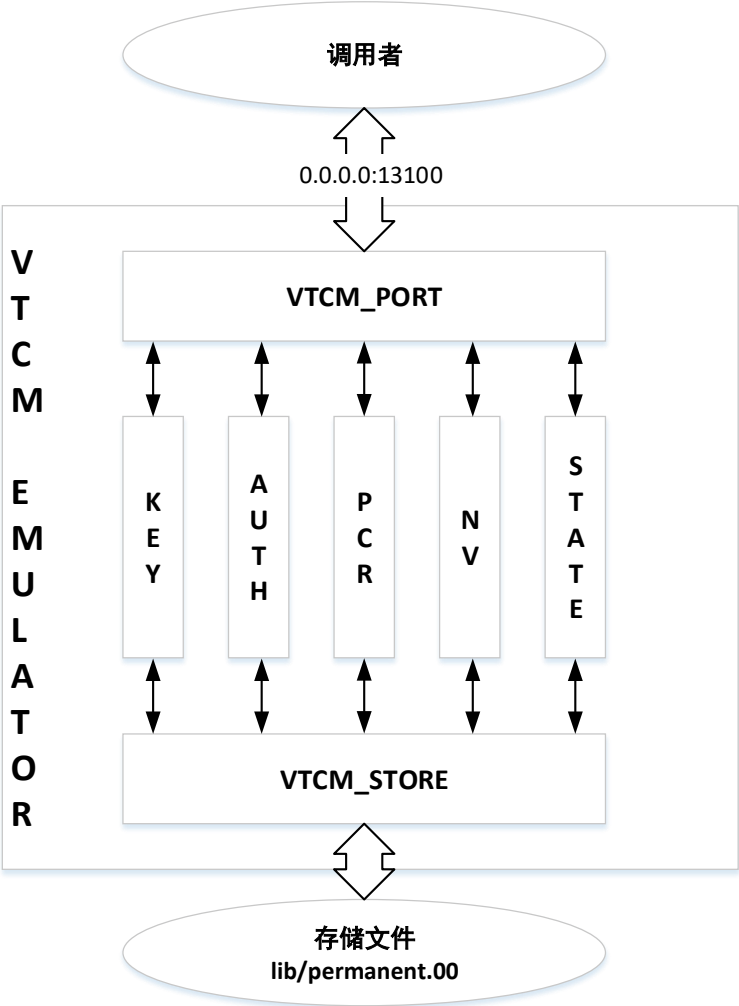


图 4-5 tcm 模拟器结构图



tcm 模拟器内部包含 7 个 cube 子模块，五个 tcm 功能模块 key、auth、pcr、nv 和 state，对外接口模块 vtc\_m\_port 和 vtc\_m 永久数据存储模块 vtc\_m\_store。

五个 tcm 功能模块中，key 模块模拟 tcm 的密钥操作，auth 模块模拟 tcm 的认证操作，pcr 模块模拟 tcm 的 pcr 操作，nv 模块模拟 tcm 的 nv 定义和读写操作，而 state 模块则模拟 tcm 的 state 状态操作。

vtc\_m\_port 为 tcm 模块对外的接口，它收到外界输入的 tcm 命令后，将其进行反序列化操作，再根据命令所属操作类别，将其分别派送到五个功能模块中。

vtc\_m\_store 实现 tcm 内部永久存储数据结构和存储文件之间的导入导出操作，vtc\_m\_emulator 启动时执行永久存储数据结构的导入，在功能模块对系统永久存储数据结构进行修改后，需向 vtc\_m\_store 模块发送消息，通知其进行永久存储数据结构的导出。

tcm 命令的序列化和反序列化直接使用了 cube 架构的数据结构处理能力而实现。在源码 include/app\_struct.h 头文件中定义了 tcm 各种命令的输入输出数据结构，在 define/tcm\_input.json 文件中则给出了 tcm 命令输入输出格式的 json 描述。模拟器利用 cube 架构的 struct\_2\_blob 和 blob\_2\_struct 命令实现命令的序列化/反序列化，命令/返回数据校验码的计算则利用了 cube 架构中部分结构数据导出的功能。因此，只要同步更改头文件定义和 json 描述，模拟器中 tcm 输入/输出数据结构的调整就只影响处理模块本身，而不会导致全局影响。

#### 4.3.2.3 cube-tcm 调用测试程序原理

Cube-tcm 调用测试程序的结构如图 6 所示：

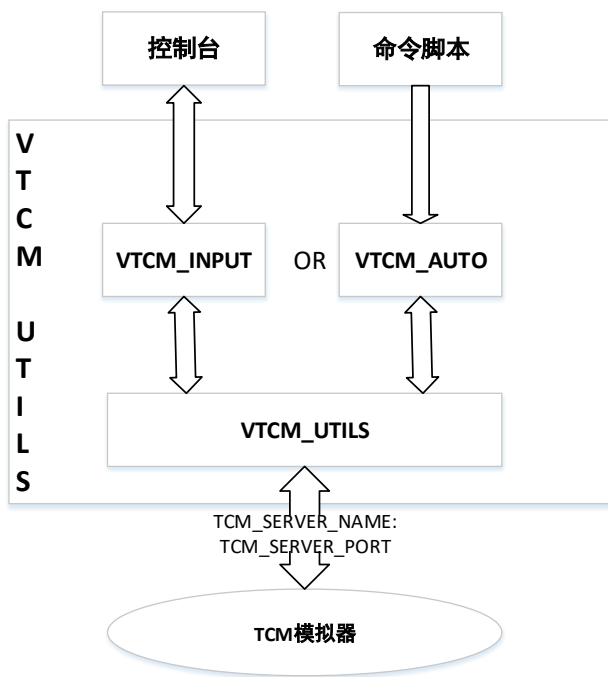


图 4-6 tcm 调用测试程序 vtc\_m\_utils 结构图

Vtcm 调用测试程序中只包含两个模块，模块 vtc\_m\_utils 为主体，接收字符串数组格式的命令和参数输入，实现 tcm 命令的生成和 tcm 返回数据的接收与解析。另一个模块则可以选择 vtc\_m\_input 模块和 vtc\_m\_auto 模块之一。选择 vtc\_m\_input 模块为交互输入模式，用户需在控制台执行命令输入，观察输出结果，得到输出结果后再次进行输入。选择 vtc\_m\_auto 模块则为脚本输入模式，调用测试程序在命令行需添加一个命令脚本参数，命令脚本中包含多条 tcm 指令。vtc\_m\_auto 模块读入脚本后，顺序执行脚本中命令，得到返回值后再执行下一条命令，直到完成脚本中所有命令或者执行过程中错误退出。

交互命令方式适合于初学者学习可信根的操作，而用调用测试程序完成具体任务时，则一般选择脚本输入方式。本章 4.3.1 小节将用交互方式介绍 TCM 的初始化过程，后面则都用脚本操作方式说明。而命令格式则在附录中列出。

### 4.3.3 cube-tcm 安装部署过程

本模拟器以源码编译方式进行安装。选定一个工作目录后，将三个源码包在

工作目录下解压后，工作目录下会出现 gm\_sm2、cube-1.3 和 cube-tcm-1.12 三个目录，分别对应国密算法，cube-1.3 框架以及 cube-tcm 模拟器。其中国密算法包不需要环境设置，cube-1.3 框架和 vtcn 模拟器则需要进行环境设置工作。Cube-1.3 的环境设置方法上一章已有说明，本章不再重复，本章介绍国密算法包和 cube-tcm 模拟器的安装方法。

cube-tcm 的环境变量设置方法必须在 cube-1.3 的环境变量设置完成后，进行，设置方法类似，进入 cube-tcm-1.12 目录后使用 source set\_env.sh 命令来执行环境变量脚本 set\_env.sh，脚本执行后，会修改 CUBEAPPPATH 和 CUBE\_APP\_PLUGIN 到 vtcn-1.12 工作目录。

由于模拟器运行过程中也需要环境变量设置，而系统新开终端并没有 CUBE 相关环境变量的配置，因此建议在用户目录的.bashrc 文件最后添加自动执行该脚本的命令，以使其每次都自动完成设置。

#### 4.3.3.1 国密算法包编译

进入 gm\_sm2 目录后，执行 make 命令，如果无错误提示信息，观察 gm\_sm2/bin 目录，应可看到编译完成的动态库 libsm2So.so。此时国密算法包即编译完成。

```
root@ubuntu:~/gm_sm2/gm_sm2# ls bin/  
libsm2So.so  
root@ubuntu:~/gm_sm2/gm_sm2#
```

下面将该库复制到 cube-tcm 的本地库目录 cube-tcm/locallib/bin 目录下。cube-tcm 内部模块将会调用该库执行国密算法计算。

#### 4.3.3.2 cube-tcm-1.12 编译

cube-tcm-1.12 包在 cube-1.3 编译完成后进行编译。注意编译前需完成国密算法库文件的复制和环境变量的设置。完成后，在 cube-tcm-1.12 目录下执行脚本 env\_build.sh，如果系统进行一系列的编译过程后，输出信息没有 error，则说明模拟器编译成功。否则请检查环境变量设置情况和算法库的位置。

在 cube-tcm-1.12 目录下执行 ls，可以看到系统下的多个目录，不同目录下的内容见前面的表 4-5。下表显示了不同目录下的内容。

```

root@ubuntu:~/vtcm-1.12# ls
define      include    locallib   plugin     set_env.sh  vtc Hub    vtc_utils
env_build.sh init_module module     README     vtc_emulator vtc_manager
root@ubuntu:~/vtcm-1.12#

```

此时 vtc\_emulator 和 vtc\_utils 两个目录下都有有效的可执行程序连接 main\_proc。我们在系统中启动两个终端 term1 和 term2, 确认两个终端均已完成环境变量配置。而后, 在 term1 终端进入 vtc\_emulator 实例目录下, 在 term2 终端进入 vtc\_utils 实例下。

Term1 终端下用 ls 观察, 可以看到多个以.cfg 结尾的配置文件。

```

root@ubuntu:~/vtcm-1.12/vtc_emulator# ls
aspect_policy.cfg      libvtcm_init.so  plugin_config.cfg  vtc Hub_config.cfg
connector_config.cfg  main_config.cfg  router_policy.cfg  vtc_init.cfg
lib                   main_proc        sys_config.cfg     vtc_port_config.cfg
root@ubuntu:~/vtcm-1.12/vtc_emulator#

```

观察 vtc\_port\_config.cfg 的内容:

```

root@ubuntu:~/vtcm-1.12/vtc_emulator# cat vtc_port_config.cfg
#connector config policy
{
    "name": "vtc_emu_port",
    "type": "SERVER",
    "family": "AF_INET",
    "address": "0.0.0.0",
    "port": 13100,
}
root@ubuntu:~/vtcm-1.12/vtc_emulator#

```

这一配置文件设置的是模拟器对外的模拟接口。模拟器模拟的是可信根芯片, 真正的芯片从它所接入的总线接收外部的输入数据包, 处理后返回输出数据包。模拟器则用网络端口来模拟总线。外部应用和模拟器的交互方式, 除端口有所区别外, 数据格式等都一致。这里我们设置模拟器端口为本机任意网络地址下, 端口为 13100。我们本地测试时, 常用 127.0.0.1:13100 作为模拟器的输入端口。

在 vtc\_emulator 目录下执行 ./main\_proc, 即可启动模拟器。运行情况如下图所示:

```

root@ubuntu:~/vtcm-1.12/vtcm_emulator# ./main_proc
mkdir: cannot create directory 'lib': File exists
vtcm init !
conn 1 is {      "name":"vtcm_hub",      "type":"SERVER",
ess":"0.0.0.0", "port":13100}
read 1 connector!
conn server vtcm_hub begin to listen!
vtcm_key_init :
tcm_permanent_data->revMajor  abc
vtcm_store_init :
vtcm_auth_init :
vtcm_switch_init :
vtcm_store module start!
vtcm_auth module start!
vtcm_switch module start!
vtcm_nv module start!
vtcm_state module start!
vtcm_key module start!
vtcm_pcr module start!

```

系统列出了所有相关的启动模块，并开始等待输入。

现在再转到 term2。在 term2 终端下进入 vtcm\_utils 目录。该目录下是可信根应用实例的运行环境 vtcm\_utils。一个可信根需要通过与应用层的一个实例间的交互过程，来执行可信根的功能。

在执行该目录下的 main\_proc 程序之前，我们需要通知 vtcm\_utils 模拟器的入口地址。这里是通过设置环境变量来完成通知的。执行 source tcm\_env.sh 127.0.0.1 13100，再执行 env | grep TCM，我们可以看到 TCM\_SERVER\_NAME 和 TCM\_SERVER\_PORT 两个环境变量已被设置。vtcm\_utils 将根据这两个环境变量来访问 vtcm\_emulator。

```

root@ubuntu:~/vtcm-1.12/vtcm_utils# source tcm_env.sh 127.0.0.1 13100
root@ubuntu:~/vtcm-1.12/vtcm_utils# env | grep TCM
TCM_SERVER_NAME=127.0.0.1
TCM_SERVER_PORT=13100
root@ubuntu:~/vtcm-1.12/vtcm_utils#

```

此时在 vtcm\_utils 目录下执行 main\_proc，即可与 vtcm\_emulator 互联。

## 4.4 TCM 服务功能与操作实例



### 4.4.1 初始化过程

TCM 在正式执行之前，需完成初始化过程。TCM 的初始化过程包括背书密钥（EK）生成和所有者授权两步，其中所有者授权操作同时完成了用户授权口令设置、SMK 生成和 SMK 授权设置三步。

为进一步熟悉可信根的运行模式，初始化过程我们用交互模式完成，而后介绍脚本模式原理并给出这一交互模式可信密码服务的实现命令脚本。

vtcm\_utils 缺省设为脚本操作，目录下有多个自动化脚本。脚本操作是由 vtcmm\_auto 子模块实现的，若想转为互动操作，则需用 vtcmm\_input 子模块替换 vtcmm\_auto 子模块。Grep vtcmm\_auto, 将查找到的所有地方替换为 vtcmm\_input, 即可转入互动操作模式，如需换回脚本操作模式，则只要再用 vtcmm\_auto 替换掉 vtcmm\_input 即可。

如图，只要把搜索到的 plugin\_config.cfg 和 router\_policy.cfg 中的 vtcmm\_auto 或 vtcmm\_input 替换，即可实现操作模式的转换。

```
root@ubuntu:~/vtcm-1.12/vtcm_utils# grep vtcmm_auto *
grep: lib: Is a directory
plugin_config.cfg:      "name": "vtcmm_auto",
plugin_config.cfg:      "libname": "vtcmm_auto"
router_policy.cfg:      "sender": "vtcmm_auto",
router_policy.cfg:      {"target_type": "LOCAL", "target_name": "vtcmm_auto"},
root@ubuntu:~/vtcm-1.12/vtcm_utils# vi plugin_config.cfg
root@ubuntu:~/vtcm-1.12/vtcm_utils# vi router_policy.cfg
root@ubuntu:~/vtcm-1.12/vtcm_utils#
```

互动操作模式下，在 vtcmm\_utils 目录下直接执行 ./main\_proc, 程序将进入等待命令状态。

```
root@ubuntu:~/vtcm-1.12/vtcm_utils# ./main_proc
mkdir: cannot create directory 'lib': File exists
begin vtcmm_input start!
Wait for the tcm command input!
```

下面我们就可以根据附录中参数列表的内容，输入命令进行操作了。

初始化的第一步是生成背书密钥 EK，EK 可以由生产商或管理方从外界注入到 TCM 中，也可以由 TCM 自己生成。我国可信根标准也未为 TCM 规定背书密

钥生成命令。为方便测试，cube-TCM 里设置了一条背书密钥命令 `createek`，命令格式为：

```
createek -wf ekey.key
```

这里可信根命令的每个参数都由一个-打头的前缀来确定其类型。`createek` 命令创建密钥后，会将密钥导出到 `ekey.key` 文件中，在 `term2` 则可看到芯片输入输出的二进制信息：

输入信息是 `vcm_utils` 根据芯片接口要求生成的数据包，如下所示：

```
createek -wf ekey.key
Begin Create EK Pair:
00 c1 00 00 00 46 00 00
80 78 00 00 00 cb 95 1e
1f 0b 85 97 03 e5 0b 26
e0 a8 ea d4 cb 22 4e eb
2d a7 5c 63 f8 25 b8 04
9a 7b 00 00 00 0b 00 02
00 05 00 00 00 04 00 00
01 00
Wait for the tcm command input!
```

输出信息则为 `vcm_emulator` 收到数据包后，按照 `tcm` 标准运行后，返回的数据，如下所示：

```
CreateEKPair is:
00 c4 00 00 00 7e 00 00
00 00 00 00 00 0b 00 02
00 04 00 00 00 04 00 00
01 00 00 00 00 40 e2 79
46 1c e6 47 f8 be cf 56
90 28 8c a2 99 1e 68 d0
be 70 df 1b 8d 19 39 67
14 01 91 7e a5 83 85 1b
66 b1 fb 8a bf 4b f4 2d
8d c9 9d 07 7f 82 e3 16
dd 9e 7f 10 6c 5a 21 e9
0f f7 c1 d9 63 bb fc c5
fc e3 99 3f 41 e9 10 b4
48 83 da 87 c0 72 0a e4
13 6c 37 ea 43 d6 7f d5
2d 6e 30 de 3e e6
```

其具体格式参考可信计算标准中的相关说明。

下面执行 readpubek 指令，该指令无输入参数，其格式为  
readpubek

该命令从 vtcem\_emulator 中获取可信根公钥，并在 vtcem\_utils 内存中存放。可信根公钥在后面的授权操作时需要用来建立起应用和可信根间的安全数据通道。

```
Begin readpubek:
00 c1 00 00 00 2a 00 00
80 7c 03 fa 81 a8 f3 6f
72 85 c0 db 30 35 ea a8
ce fe 2f d2 5a c7 71 11
d7 db 02 b4 f9 36 1e e7
0b 52
Wait for the tcm command input!
00 c4 00 00 00 7e 00 00
00 00 00 00 00 0b 00 02
00 04 00 00 00 04 00 00
01 00 00 00 00 40 e2 79
46 1c e6 47 f8 be cf 56
90 28 8c a2 99 1e 68 d0
be 70 df 1b 8d 19 39 67
14 01 91 7e a5 83 85 1b
66 b1 fb 8a bf 4b f4 2d
8d c9 9d 07 7f 82 e3 16
dd 9e 7f 10 6c 5a 21 e9
0f f7 c1 d9 63 bb e4 e3
e9 16 23 13 cb 32 32 f0
f0 7b 9a cc e3 ec 78 12
b2 2a 6c 1b 16 48 78 d4
94 ed b0 e5 d6 0a
readPubek Checksum succeed!
```

下一条指令是 apcreate，这条指令用来创建应用和模拟器中对象的会话。初始化过程中，模拟器内部并没有合适的对象，因此 apcreate 命令将创建一个叫做 ET\_NONE 类型(编号为 0x12)的会话，这一会话专用于初始化过程。此时 apcreate 命令格式为 apcreate -it 12，vtcem\_utils 根据这一命令生成的输入命令格式如下



```
apcreate -it 12
00 c2 00 00 00 50 00 00
80 bf 00 12 00 00 00 00
90 91 ab 3c 1e ab 52 89
7c 6a dd be 96 0f ab 43
61 69 3d e8 d1 57 5f c7
44 f0 03 17 dc bc ed cd
f8 a2 47 15 26 1b 8a 70
9b 91 11 20 ab 1a 77 c2
7e 4a 01 27 51 e4 99 09
f2 04 d5 3c aa b9 a1 58
```

输出命令格式如下：

```
Wait for the tcm command input!
00 c5 00 00 00 52 00 00
00 00 2c 18 27 01 6c 63
2a 11 92 04 34 0b 20 ea
78 86 9b 63 7e 68 31 58
f4 d7 e2 18 9e f1 5b 67
ab 55 4f 26 d7 93 00 00
00 00 5b e8 71 4d 61 bf
85 86 1d 99 31 f3 cb c8
e5 b8 44 14 64 0e 21 2b
59 30 33 3b 39 4b bc 5a
ab 2d
```

此时 vtcu\_utils 和 vtcu\_emulator 之间已经建立了一个会话。这个会话有一个句柄记录，但是由于 ET\_NONE 类型的会话是唯一的，因此我们在这个状态下省略了这个句柄的显示。

下面是初始化过程中最关键的一个命令 takeownership，该命令完成可信根属主口令的生成、可信根存储密钥以及根存储密钥口令的生成。属主口令用来获取可信根中属主的权限，可以进行用户鉴别密钥生成等操作。根存储密钥为一个随机数，作为 SM4 对称加密算法的私钥，只用于保护可信根生成的密钥，存放在可信根中，禁止导出，只有用根存储密钥口令建立会话后，才能使用。

Takeownership 命令的输入参数应包括属主口令、根存储密钥口令和 ET\_NONE 会话句柄，由于初始化过程中 ET\_NONE 会话句柄应当唯一，因此我们在 vtcu\_utils 里省略了这一句柄参数，而让程序在其生成的会话表中自行查找。

Takeownership 命令的输入格式为：

takeownership -pudo 用户口令 - pwds smk 口令

该命令的输入和返回数据过长，在这里略去，读者可试运行后自行观察。

Takeownership 命令执行成功后，将在 TCM 中创建存储根密钥 smk,其实体类型值为 04。我们可以通过用 apcreate 命令创建 vtcu\_utils 和 smk 的会话来验证 takeownership 是否成功。测试命令为

apcreate -it 04 -pwd smk 口令，

下面为测试命令的输出内容，我们可以看到两个返回值，第一个是命令返回码，其取值为 0，表示执行正确，第二个则是 apcreate 命令与 smk 之间建立可信会话连接的会话句柄。这表示 takeownership 命令执行正确。

```
00 c2 00 00 00 50 00 00
80 bf 00 04 00 00 00 00
c2 fe 40 32 43 2e 25 9d
a2 cc 8c 77 94 1e 72 dd
f6 25 9a 84 c1 e2 cf 01
e3 3c 6a 24 f6 f3 95 dd
38 d8 8f a2 85 d0 90 bc
c7 a6 13 14 2c 05 06 ec
fc a3 51 f6 fa 87 87 e0
f5 89 05 92 94 ff 46 7e

00 c5 00 00 00 52 00 00
00 00 87 17 b9 0c 18 6a
ab e0 65 b3 19 13 9e 8a
fc fc 43 33 9b 7c cc df
63 84 3d ff 3b 56 6f 11
c4 fc 05 ef bc 13 00 00
00 00 a4 0f 02 6e 2e 42
77 b7 15 04 1a e1 d3 dc
4d 4a 0d 0e da 75 85 6b
62 fc 2b 11 89 c5 3a e1
65 ef
Output para: 0 8717b90c
```

而如果我们输入错误的 smk 口令或者省略了 -pwd 选项，则我们应该在命令最后得到下面的结果：

```
APCreate check output authCode failed!
```

下面，我们写一个脚本来完成这一初始化过程。

脚本操作记录多个命令的执行格式，每个命令用 in:和 out:开头的两行来表示，为了支持连续操作，脚本操作中还引入了变量的概念，变量为以\$开头的字符串，

用来代表操作过程中生成的值。变量一般在命令输出时用来记录特定位置的输出信息，已记录信息的变量可以用在后面命令的输入信息中。

脚本操作中，**in:**表示命令的输入格式，后面跟与互动操作相同的输入命令，**out:**则表示命令的输出，后面用序号加冒号再加变量名的方式（如 **1:**）表示将对应序号的标识对应的输出参数。其中 **out** 命令在不需要记录输出信息时可忽略。

如下为一个名为 **takeowner.cmd** 的脚本示例，位于实例 **vcm\_utils** 目录下，用来让用户对一个已经生成背书密钥，但尚未创建 **SMK** 的可信根执行获取权限操作。该脚本包含了四条命令：**readpubek** 获取背书密钥公钥，**apcreate** 命令创建一个 **TCM\_ET\_NONE** 会话，**takeownership** 创建 **SMK** 并完成授权操作，而 **apterminate** 则终止 **apcreate** 所创建的会话。

```
# this cmdlist is for tcm_emulator init
```

```
in: readpubek
```

```
in: apcreate -it 12
```

```
out: 1:$ownerHandle
```

```
in: takeownership -pwd 000 -pws sss
```

```
in: apterminate -ih $ownerHandle
```

```
in apcreate -it 04 -pwd sss
```

```
out: 1:$smkHandle
```

```
in: apterminate -ih $smkHandle
```

脚本第一行为注释行。注释行下面分为六个小段落，分别对应六条命令。其中第 1,3,4,6 段落只有 **in:**开头的 1 行，表示我们并不关注命令的输出，第 2,5 段落则包括了输入和输出。所有命令的第一个输出变量（标识为 0:）均为命令的返回码，而 **apcreate** 命令的第二个输出变量为 **apcreate** 命令创建会话的句柄值。第二个命令是创建一个对 **ET\_NONE** 实体的会话，返回的会话句柄值被记录到变量

\$ownerHandle 中。在第四个命令中, \$ownerHandle 又被作为 apterminate 的输入参数, 以终止已经成功功能的会话, 释放会话所占用的资源。第五、六个命令与其类似, 只是实体对象变为 SMK。需要注意, 可信根中会话资源空间有限, 最少情况下可信根可以只支持三个会话同时进行, 其它会话则需要等运行中会话中断后才能继续进行。因此, 在会话完成后, 及时中断会话是非常必要的。

脚本执行时, 只需以脚本文件作为命令行参数, 在 vtcn\_utils 目录下执行 ./main\_proc 脚本文件名

这一格式命令即可。参照该脚本操作的格式, 我们可以将各种本地可信操作都用脚本方式书写。

### 4.4.2 加密、解密、签名、验证

#### 4.4.2.1 密钥创建

TCM 中的任意密钥 K 导出时, 其内部的敏感信息需要有另一个密钥  $K_w$  加密保护, 这一加密密钥  $K_w$  称为 K 的封装密钥。封装密钥可选择 SMK 或其它的存储密钥, 而第一个创建的非 SMK 密钥则只能选择 SMK 作为自己的封装密钥。因此, 在执行 takeownership 命令生成了 SMK 之后, TCM 才可以进行密钥相关操作。

下面我们首先执行创建密钥操作。TCM 中, 密钥创建可以选择基于 SM2 算法的非对称钥和基于 SM4 算法的对称钥。密钥创建命令为 createwrapkey。tcn\_utils 接受的密钥创建命令其格式如下:

createwrapkey -ikh 封装密钥句柄 -ish 封装密钥会话句柄 -is (sm2|sm4) -kf 密钥文件名称 -pwdk 密钥授权口令 [-pwm 密钥移植口令]

这里封装密钥句柄是所创建密钥的封装密钥, 当选择 SMK 为封装密钥时, 封装密钥句柄为 0x40000000, 否则密钥为用 loadkey 命令载入密钥文件到 tcm 中时, 返回的密钥句柄值。

封装密钥会话句柄是以封装密钥为对象使用 acreate 命令, 获得的会话句柄值。

-is 要求调用者选择密钥类型, 非对称钥选择 sm2 类型, 对称钥选择 sm4 类型。

-kf 则选择输出的密钥文件名称, 创建密钥时, 密钥并不在 tcm 中保存, 而是以封装后的密钥文件方式输出。因为 tcm 内部密钥空间有限, 一般只载入即将使用的密钥。输出的密钥文件需要用 loadkey 命令载入到 tcm 中才可以正式使用。

-pwdk 后面跟随的参数是密钥授权口令参数, 载入的密钥必须经过授权验证和会话建立才可以使用, 授权验证验证的就是授权口令用 sm3 算法哈希后的结果,

这一结果被封装密钥加密后存放于密钥文件的敏感数据结构中，在 tcm 载入密钥后用来验证。

-pwdm 后面跟随的参数是密钥移植时的授权口令，这一参数仅用于密钥移植的情况，我们暂时不讨论这一情况。

createwrapkey 成功执行后，我们可以获得一个密钥文件。但必须通过 loadkey 命令将其载入到 tcm 中，才可以正式使用这一密钥文件。

loadkey 命令的格式如下：

loadkey -ih 封装密钥会话句柄 -kf 密钥文件名称

下面我们写一个密钥创建脚本程序 createkey.cmd，该程序以 SMK 为封装密钥创建一个非对称密钥和一个对称密钥，密钥口令分别为 sm2 和 sm4，并分别存储在文件 sm2.key 和文件 sm4.key 中，脚本如下：

```
in: apcreate -it 04 -pwd sss
```

```
out: 1:$smkHandle
```

```
in: createwrapkey -ikh 40000000 -ish $smkHandle -is sm2 -kf sm2.key -pwdk  
sm2
```

```
in: createwrapkey -ikh 40000000 -ish $smkHandle -is sm4 -kf sm4.key -pwdk  
sm4
```

```
in: apterminate -ih $smkHandle
```

可以看到，我们首先需要通过 apcreate 命令创建与 SMK 的会话，并记录会话句柄，在创建密钥时，以 SMK 的句柄 0x40000000 为封装密钥句柄输入，以 apcreate 命令返回的句柄 \$smkHandle 为封装密钥会话句柄输入，即可执行 createwrapkey 命令，生成非对称密钥文件 sm2.key 和对称密钥文件 sm4.key。这两个密钥文件将用于本小节后面的加解密与签名验证操作中。

#### 4.4.2.2 加解密操作

密钥创建完成后，即可进行加解密操作。TCM 提供了三条加解密命令，分别是 SM2 解密命令 TCM\_SM2Decrypt，SM4 加密命令 TCM\_SM4Encrypt 和 SM4 解密命令 TCM\_SM4Decrypt。为什么 TCM 不提供 SM2 加密命令呢？因为 SM2 公钥不需要保密，调用者可以直接从密钥文件中读出公钥并用软算法执行加密操作，如果调用者用远端 TCM 创建的非对称密钥公钥来加密数据，那么加密的数据只

有远端 TCM 才能解开。这一特点可用于公共场合的敏感数据传输。

下面我们分别编写一个非对称钥加解密脚本和一个对称钥加解密脚本，使用的密钥是上一小节产生的非对称密钥和对称密钥。

非对称加解密脚本如下所示：

```
in: sm2encrypt -kf sm2.key -rf sm2test.dat -wf sm2crypt.dat
```

```
in:  apcreate -it 04 -pwd sss
```

```
out: 1:$smkHandle
```

```
in: loadkey -ih $smkHandle -kf sm2.key
```

```
out: 1:$keyHandle
```

```
in: apterminate -ih $smkHandle
```

```
in: apcreate -it 01 -iv $keyHandle -pwd sm2
```

```
out: 1:$keyAuthHandle
```

```
in: sm2decrypt -ik $keyHandle -is $keyAuthHandle -rf sm2crypt.dat -wf  
sm2decrypt.dat
```

```
in: apterminate -ih $keyAuthHandle
```

sm2encrypt 是一个 tcm 外部命令，它是在 vtcml\_utils 实例中独立实现，不需要和 TCM 进行交互，因此，这一命令也可以在其它 TCM 所在机器或者无 TCM 的机器上运行。本脚本用 sm2encrypt 命令加密了 sm2test.dat 文件，加密数据写入到 sm2crypt.dat 文件中。

解密过程比较复杂，首先要建立与封装密钥（SMK）的会话，使用会话句柄执行 loadkey 命令，载入密钥，并获得返回的密钥句柄，再使用密钥句柄，输入授权口令 sm2，用 apcreate 命令建立与载入密钥的会话。最后以密钥句柄和密钥会话句柄为输入，执行 sm2decrypt 命令，解密 sm2crypt.dat 文件并将解密数据写入 sm2decrypt.dat 中。

对称密钥加解密脚本如下所示：

```
in:  apcreate -it 04 -pwd sss
out: 1:$smkHandle
```

```
in: loadkey -ih $smkHandle -kf sm4.key
out: 1:$keyHandle
```

```
in: apterminate -ih $smkHandle
```

```
in: apcreate -it 01 -iv $keyHandle -pwd sm4
out: 1:$keyAuthHandle
```

```
in: sm4encrypt -ikh $keyHandle -idh $keyAuthHandle -rf sm4test.dat -wf
sm4crypt.dat
```

```
in: sm4decrypt -ikh $keyHandle -idh $keyAuthHandle -rf sm4crypt.dat -wf
sm4decrypt.dat
```

```
in:  apterminate -ih $keyAuthHandle
```

与 sm2 加解密命令不同的是, sm4encrypt 也是需要 tcm 执行的命令, 因此 sm4 加密时需要首先载入密钥, 再建立调用者和密钥的会话, 才能够进行 sm4 加解密操作。另外 sm4 算法加解密时, 采用的是 TCM\_ES\_SM4\_CBC 模式, 该模式会填充加密数据以保证其长度为 16 的整数倍, 因此加密后文件会略长于明文文件, 解密后文件长度将恢复。

#### 4.4.2.3 签名-验证操作

tcm 创建 sm2 密钥时, 如果密钥的 KeyUSAGE 属性为 TCM\_SM2KEY\_SIGNING, 则密钥可用来执行签名和验证操作。

tcm 的密钥签名操作必须在 tcm 内部执行, 需要首先建立与签名密钥封装密钥的会话, 再载入签名密钥, 建立与签名密钥的会话, 然后通过 sign 命令执行签名操作。签名操作使用签名密钥计算一个文件中数据的 SM2 签名, 并将签名数据输出到签名文件中。SM2 签名需要一个 UserID, cube-tcm 模拟器使用的 UserID 为 32 个'A'字符。

签名-验证操作的测试脚本如下所示:

```
in: apcreate -it 04 -pwd sss
```

```
out: 1:$smkHandle
```

```
in: loadkey -ih $smkHandle -kf sm2.key
```

```
out: 1:$keyHandle
```

```
in: apterminate -ih $smkHandle
```

```
in: apcreate -it 01 -iv $keyHandle -pwd sm2
```

```
out: 1:$keyAuthHandle
```

```
in: sign -ik $keyHandle -is $keyAuthHandle -rf signtest.dat -wf signtest.sig
```

```
in: apterminate -ih $keyAuthHandle
```

```
in: verify -kf sm2.key -rf signtest.dat -sf signtest.sig
```

```
~
```

该脚本使用密钥创建步骤生成的 sm2key,载入密钥后,执行签名命令,而后退出所有 TCM 连接,由调用者从 sm2 密钥文件中读出公钥文件后,执行验签操作。

#### 4.4.3 PIK 申请过程

PIK 是 TCM 提供可信报告功能时必须创建的 TCM 内部实体,它实际上是一个 KEYUSAGE 为 TCM\_SM2KEY\_IDENTITY (身份密钥)的 SM2 密钥对,但是其生成过程与上一小节介绍的 SM2KEY 生成过程有较大区别。

在申请 PIK 时,需要有一个 CA 方参与。最简单的 CA 只拥有一个 SM2 公私钥对作为 CA 密钥,CA 方秘密保存私钥,其它用户获得公钥并认可该公钥为 CA 公钥。同时 CA 应获取所有申请 PIK 用户的 EK 公钥以验证 PIK 申请并确认其签发的证书只有合法 PIK 用户才能使用。

实际开发中,可能会使用不同格式的公钥证书,但这些证书的使用不改变其原理,一般只是在这一基础上添加一些 CA 认证信息。作为测试程序,本书只构建一个最简单的 CA 来支持 PIK 申请操作。



在 `vcm_utils` 实例中，提供了 CA 密钥对创建和加载的命令，分别为 `createsm2key` 和 `loadcakey`，其中 `createsm2key` 命令创建一个 sm2 公私钥对做为 CA 密钥，并分别存放在公钥文件和私钥文件中，`loadcakey` 则将 CA 钥载入（可以选择载入公钥、载入私钥或同时载入）。下面是 CA 端密钥创建命令示例。

```
in: createsm2key -pubkey capub.key -prikey capri.key
```

```
#创建一个 CA 公私钥对，非可信根操作
```

```
out:
```

```
in: loadcakey -pubkey capub.key -prikey capri.key
```

```
#载入 CA 公钥 与 私钥
```

CA 方创建 CA 密钥后，将公钥发送给所有 PIK 申请者和使用者，而 PIK 申请者则将自己的 ek 公钥提交给 CA 方。设 ek 公钥名称为 `ekpub.key`，PIK 申请方可用 `readpubek -wf ekpub.key` 命令获取，并将获取到的公钥文件复制给 CA 方。当 CA 方确认 ek 公钥合法性，参与者也确认 CA 公钥可信性后，即可进行 PIK 生成流程。

PIK 生成流程分为三步：

1. PIK 申请方生成 PIK 并结合用户身份信息，产生 PIK 认证请求包。
2. CA 方验证认证请求，用 CA 私钥签名 PIK 证书，并将 PIK 证书封装到 PIK 激活包中
3. PIK 申请方解封 PIK 激活包，获得 CA 签名的 PIK 证书。

第一步由 PIK 申请方载入 CA 公钥后执行，PIK 申请方建立与 owner 的会话和 smk 的会话，然后执行 `makeidentity` 命令，输入用户信息文件，产生 pik 密钥文件以及 pik 请求包。cube-tcm 相关功能的自动化脚本如下：

```
in: loadcakey -pubkey capub.key
```

```
#载入 CA 公钥
```

```
in: apcreate -it 02 -pwd ooo
```

```
#建立 owner 会话 记录会话句柄
```

```
out: 1:$ownerHandle
```

```
in: apcreate -it 04 -pwd sss
#建立 smk 会话, 记录会话句柄
out: 1:$smkHandle
```

```
in: makeidentity -ioh $ownerHandle -ish $smkHandle -if user_info.list -of
request.req -kf pik.key
```

#生成鉴别密钥和密钥认证申请包, 密钥文件导出

```
in: apterminate -ih $ownerHandle
```

```
in: apterminate -ih $smkHandle
```

第二步由 CA 方执行 pik 请求包验证后, 签发 pik 证书。

CA 方需首先载入 CA 公钥和私钥, 再用 casign 命令来执行验证和签发证书操作。自动化脚本内容如下:

```
in: loadcakey -pubkey capub.key -prikey capri.key
#载入 CA 公钥 与私钥
```

```
in: casign -user user_info.list -pik pik.key -ek ekey.key -req request.req -cert
pik.cert -symm symm.key
out:
~
```

CA 方载入公钥和私钥后, 用 casign 命令完成 pik 证书生成, 并加密 pik 证书。这一命令在 TCM 之外完成。casign 命令参数较多, 每个参数对应一个文件, 其中 user\_info.list, pik.key, ekey.key, request.req 为输入文件, pik.cert 和 symm.key 为输出文件, 下面特别进行说明:

user\_info.list : 用户信息, 需要和 PIK 申请方执行 makeidentity 时的用户信息一致。

pik: PIK 申请方的 PIK 钥文件, 只需公钥部分内容。

ekey.key: 预先获取的 PIK 申请方 ek 公钥。

request.req: PIK 申请方调用 request.req 产生的申请包, 其实际为 user\_info.list 和 pik 公钥的摘要值。

pik.cert: ca 方签发的 pik 证书。cube-tcm 中, 我们用下面所示结构做了一个简

化的证书:

```
typedef struct tcm_pik_cert
{
    TCM_PAYLOAD_TYPE payLoad;    // should be 0x19
    BYTE userDigest[DIGEST_SIZE]; //用户信息的摘要
    BYTE pubDigest[DIGEST_SIZE]; //pik 公钥信息的摘要
    int signLen;                  //签名长度
    BYTE * signData;              //签名数据
}__attribute__((packed)) TCM_PIK_CERT;
```

symm.key: 证书加密对称钥。ca 端在生成证书后, 产生一个随机 sm4 密钥, 用该密钥通过 sm4 算法加密证书数据, 然后再用 PIK 申请方的 EK 公钥加密证书, 再将证书和非对称钥加密的对称钥提供给 PIK 申请方。

casign 命令读取 user\_info.list 和 pik.key 后, 验证用户信息摘要和 pik 公钥摘要, 并与 request.req 对比, 确认其一致性, 而后用用户信息摘要和 pik 公钥摘要填充 pik 证书, 用 CA 私钥签名, 再生成 sm4 对称密钥, 加密 pik 证书, 输出加密的证书文件, 并用 PIK 申请者的 EK 公钥加密对称密钥, 生成加密对称密钥文件输出。

CA 端完成签名之后, 将加密的证书文件 pik.cert 和加密对称密钥文件 symm.key 提供给 PIK 申请方, 即完成了 CA 端的 PIK 证书签发工作。

第三步由 PIK 申请方收到加密的 PIK 证书和对称密钥后执行, 主要通过 activateidentity 命令完成。自动化脚本如下所示:

```
in: readpubek

in: loadcakey -pubkey capub.key
#载入 CA 公钥 与私钥

in: apcreate -it 02 -pwd ooo
#建立 owner 会话 记录会话句柄
out: 1:$ownerHandle

in: apcreate -it 04 -pwd sss
#建立 smk 会话, 记录会话句柄
out: 1:$smkHandle
```

```
in: loadkey -ih $smkHandle -kf pik.key
```

```
out: 1:$keyHandle
```

```
in: apcreate -it 01 -iv $keyHandle -pwd kkk
```

```
#创建 pik 会话，返回 pik 会话句柄
```

```
out: 1:$authHandle
```

```
in: activateidentity -ish $authHandle -ioh $ownerHandle -ikh $keyHandle -symm
symm.key
```

```
in: apterminate -ih $authHandle
```

```
out:
```

```
in: apterminate -ih $ownerHandle
```

```
out:
```

```
in: apterminate -ih $smkHandle
```

```
out:
```

```
in: decryptpikcert -kf symm.key -cf pik.cert
```

完成 PIK 激活操作时，需载入 ek 公钥和 CA 公钥，建立起 owner 会话和 smk 会话，载入 pik，建立 pik 会话，然后执行 activateidentity 命令，该命令用 EK 私钥解密对称密钥。在获得对称密钥后，我们用 decryptpikcert 命令解密 pik 证书，这一命令为外部命令，可以不通过 tcm 执行。解密后的 pik 证书文件中包含用户信息的摘要值和 pik 公钥的摘要值。

PIK 申请方可以向其它认可 CA 公钥的用户提供这一证书、pik 公钥以及个人信息，这些用户可用 CA 公钥验证 pik 证书的可信性，再用 sm3 算法计算 pik 公钥和用户信息的完整性，并与 pik.cert 中存放的摘要值比对，以确认 pik 公钥和用户信息的可信性。验证 pik.cert 时不需要使用 tcm，其自动化脚本如下所示：

```
# this cmdlist is for tcm_emulator sign test
```

```
in: loadcakey -pubkey capub.key
```

```
#载入 CA 公钥
```

```
in: caverify -cf pik.cert
```

```
#用 CA 公钥验证 pik.cert 完整性
```

#### 4.4.4 完整性报告

完整性报告机制选择 TCM 中的 PCR 值生成关于 PCR 值的报告，并在 TCM 中用 PIK 私钥签名。签名后的报告可以发给远端，由远端用 PIK 公钥进行验证。注意远端的验证行为只是验证了 PCR 值确实是 TCM 当前的 PCR 值，PCR 值是否是合法值还需远端通过其它途径获取 PCR 的预期值并加以比较。

cube-tcm 中用 pcrread 命令读取 PCR 寄存器值，用 extend 命令扩展 pcr 寄存器。这两个命令对应 TCM 芯片的 pcrread 和 extend 命令。为了方便生成多个 pcr 寄存器的预期值，pcrread 命令可添加一个写文件参数 -wf pcrfile，这里 pcrfile 文件存储 TCM\_PCR\_COMPOSITE 格式数据，其格式如下所示：

```
typedef struct tdTCM_PCR_COMPOSITE {
    TCM_PCR_SELECTION select;    //PCR 寄存器选择位图
    uint32_t valueSize;          // PCR 值记录长度
    BYTE *pcrValue;              //PCR 值数组，为位图所选寄存器
                                //PCR 值按次序排列
}__attribute__((packed)) TCM_PCR_COMPOSITE;
```

如 pcrread 命令带 -wf 参数，命令将打开 pcrfile，读出数据结构，再将从 tcm 中读出的 PCR 值复制到结构中。如该寄存器已被选中，则替换当前值，否则在结构中添加新的 PCR 值。

在生成完整性报告之前，首先应产生一个预期值。预期值可以根据预期的 PCR 写入过程计算生成，也可以从一个达到预期状态的可信系统中直接采集。下面是一个预期值采集的自动化脚本：

```
in:  extend -ix 1 -ic aaaa
```

```
out: 1:$pcrValue
```

```
in:  pcrread -ix 0 -wf pcrfile
```

```
out: 1:$pcrValue
```

```
in: pcrread -ix 1 -wf pcrfile
out: 1:$pcrValue
```

该程序向 PCR1 中输入字符串 aaaa，执行 pcr 扩展操作，然后将 pcr0 和 pcr1 的值导出到 pcrfile 文件中。

下面的自动化脚本则生成了 pcr0 和 pcr1 的完整性报告。

```
in: apcreate -it 04 -pwd sss
out: 1:$smkHandle
```

```
in: loadkey -ih $smkHandle -kf pik.key
#载入鉴别密钥,返回密钥句柄
out: 1:$keyHandle
```

```
in: apterminate -ih $smkHandle
```

```
in: apcreate -it 01 -iv $keyHandle -pwd kkk
#创建 pik 会话, 返回 pik 会话句柄
out: 1:$keyAuthHandle
```

```
in: quote -ikh $keyHandle -ish $keyAuthHandle -ix 3 -of quote.rpt
#生成寄存器的完整性报告
```

```
in: apterminate -ih $keyAuthHandle
```

脚本开始时载入了 pik 认证密钥并建立起认证密钥的会话，然后调用 quote 命令生成了完整性报告。quote 命令的输入参数中，-ikh 后面是密钥的句柄，-ish 后面是密钥会话句柄，-ix 后面是位图形式选择的 pcr 寄存器，3 的二进制表示为 00000011，第 0 位和第 1 位为 1，表示选择 PCR0 和 PCR1 生成完整性报告，quote.rpt 则是搜集了命令的输出信息而形成的 PCR 报告，报告结构如下：

```
Struct Pcrreport
{
    TCM_QUOTE_INFO quoteinfo; //PCR 报告信息
```

```

    int sigSize;                // PCR 报告签名尺寸
    BYTE * sig;                 //PCR 签名大小
}__attribute__((packed))

```

PCR 签名信息中包含的数据结构如下:

```

typedef struct tdTCM_QUOTE_INFO {
    TCM_STRUCTURE_TAG tag;      //标识值 TCM_TAG_QUOTE_INFO2
    BYTE fixed[4];              //固定信息"QUOT"
    TCM_NONCE externalData;     //命令中生成的防重放攻击随机数
    TCM_PCR_INFO info;          //度量 PCR 信息, 包含 PCR 摘要值
}__attribute__((packed)) TCM_QUOTE_INFO;

```

这一信息被转化为二进制包后, 由 `pik` 私钥签名, 签名长度为 `sigSize`, 签名数据为 `Sig`。

这一 PCR 报告生成后, 可以发送给持有 `pik` 公钥的用户, 由该用户验证 PCR 报告的可信性, 并从中提取度量的 PCR 值, 与预期值比较。

下面是验证 `quote` 命令生成报告的自动化脚本:


```
in: verifyquote -kf pik.key -rf quote.rpt
```

```
in: checkquotePCR -pf pcrfile -rf quote.rpt
```

该脚本可以在无 `tcm` 的环境下运行。脚本中只有两条命令, 第一条命令是从 `pik.key` 中读取 `pik` 公钥, 用 `pik` 公钥验证 `quote.rpt` 报告的可信性。第二条命令则计算 `pcrfile` 中 `pci` 信息 `TCM_PCR_COMPOSITE` 的完整性校验值, 并与 `quote.rpt` 中记录的 `pcr` 完整性校验值比较。如果两条命令返回值均为 0, 则表示执行 `quote` 时所度量的 TCM 中 `pcr` 寄存器的值与 `pcrfile` 中的值一致。

#### 4.4.5 NV 操作

## 4.5 可信密码服务框架设计

 可信密码服务会涉及多个密钥的可信管理。密钥可信管理涉及密钥的检索、存储和证明等内容。本节介绍了利用 Cube 的内存数据库实现可信密钥存储和管理的方法，以及利用可信根功能实现密钥可信性证明的方法，并用一个密钥自动化交换的实例作为可信密钥管理的示例。

### 4.5.1 可信密钥管理基本架构

## 4.6 结论