

可信平台模块的形式化分析和测试

陈小峰

(中国科学院软件研究所信息安全国家重点实验室 北京 100190)
(信息安全共性技术国家工程研究中心 北京 100190)

摘 要 可信平台模块(Trusted Platform Module, TPM)是可信计算平台的核心和基础,可信平台模块的功能测试和验证是保证可信平台模块的实现正确性以及规范一致性的重要手段,但是目前尚不存在一种有效严格的可信平台模块测试和功能验证方法,同时可信计算组织给出的 TPM 规范是描述性的,不利于产品的开发和测试. 文中在分析可信平台模块目前存在的一些问题的基础上,以 TPM 密码子系统为例给出了该子系统的形式化规格说明,并且基于该规格说明,给出了扩展有限状态机模型,最后,将该有限状态机模型应用于测试用例的自动生成,并通过实验验证了形式化测试的有效性.

关键词 可信计算平台;可信平台模块;一致性测试;形式化分析

中图法分类号 TP309 **DOI号**: 10.3724/SP.J.1016.2009.00646

The Formal Analysis and Testing of Trusted Platform Module

CHEN Xiao-Feng

(State Key Laboratory of Information Security, Institute of Software, Chinese Academy of Sciences, Beijing 100190)
(National Engineering Research Center of Information Security, Beijing 100190)

Abstract Trusted platform module is the core component of trusted computing platform, the functional testing and validation is an important method to ensure the correctness and specification compliance of the trusted platform module, but until now, there is no valid formal testing and validation method, meanwhile the specification which is given by trusted computing group is descriptive, it is not convenient for product development and testing. The paper firstly analyzes the problem of the trusted platform module, and then gives the Z specification of TPM's cryptography system, based on this formal specification, gives the extended finite state machine model. Finally, the authors use the EFSM model for generating the test case and analyze the testing results.

Keywords trusted computing platform; trusted platform module; compliance testing; formal analysis

1 引 言

可信计算平台技术是一种通过硬件信任根解决安全问题的新技术,可信平台模块(简称 TPM)是可信计算平台的核心和基础,是可信计算平台推广和应用的关键. 可信计算平台的发展与应用是和相应

的技术规范分不开的,为了促进可信计算平台的发展,2003年,可信计算组织(Trusted Computing Group, TCG)^①给出了可信平台模块(Trusted Platform Module, 简称 TPM) 1.2 规范,详细界定了 TPM 的功能.

为了保证 TPM 实现的质量,必须对其进行详尽的测试和验证,然而目前针对可信平台模块的正

收稿日期:2008-12-08;最终修改稿收到日期:2009-02-09. 本课题得到国家自然科学基金(60673083, 60603017)、国家“九七三”重点基础研究发展规划项目基金(2007CB311202)资助. 陈小峰,男,1980年生,博士研究生,主要研究方向为网络安全、可信计算. E-mail: chenxiaof@is.iscas.ac.cn.

① Trusted Computing Group. www.trustedcomputinggroup.org

确性验证和测试的研究工作尚显不足,主要存在以下几方面问题:

(1)可信平台模块的形式化建模

目前可信计算组织给出的 TPM 规范是描述性的,容易造成歧义,不利于产品开发,因此给出一个准确的形式化模型是十分必要的,同时准确的形式化模型可以给 TPM 的功能验证、分析以及形式化测试提供基础。

(2)一致性测试的测试用例的数量和质量问题

目前针对可信平台模块的测试用例都是手工来完成的,而手工测试最大的问题在于测试的完整性和覆盖度问题无法解决,从而造成测试结果的可信度不高。此外,测试用例的有效性将直接影响 TPM 的测试效率和测试成本,如何实现测试用例的自动生成成为亟需解决的问题。

(3)可信平台模块的测试缺少自动化测试方案的支持

由于 TPM 规范涉及的内容广泛,纯手工的测试需要大量的人力和物力,迫切需要一种自动化的测试方案以支持测试的自动化,并提供相应的支撑工具。

针对上面的问题,本文在分析 TPM 规范的基础上给出了 TPM 的形式化模型,并且探讨了该形式化模型在自动化测试方面的应用,首次给出了一个完整的自动化测试方案,解决了测试用例只能手工生成的问题,保证了针对 TPM 的测试具有一定的覆盖范围,并且对于生成的测试用例的有效性进行了实验验证。

本文第 2 节介绍相关的工作;第 3 节给出 TPM 的形式化模型;第 4 节给出模型的应用实例;第 5 节重点介绍针对 TPM 的一致性测试和结果分析;第 6 节总结全文。

2 相关工作

在可信平台模块的形式化分析和建模方面,文献[1]对 TPM 的一些关键安全机制进行了形式化分析,其建立的模型主要是用于安全性分析。同时在文献[2]中,作者对 TPM 的授权协议进行了形式化的分析,并且通过模型检测技术分析该协议存在的攻击方法。

文献[3]首次针对 TPM 给出了一个详细的测试方案并且给出了测试结果,对测试的结果进行了详细的分析,但是文献[3]给出的测试方案的缺陷在于该方案基本上采用手工测试方法,其测试用例都是通过手工输入的,而且没有进行测试范围的分析,

无法保证其测试的合理性和有效性。

在测试用例的自动生成方面,已经有不少的研究成果^[4-7]。同时用例生成技术已经应用于多个领域,如在文献[8]中,作者探讨了智能卡的测试用例自动生成;在文献[9]中给出了一个测试用例自动生成技术在 Web 服务测试中的应用实例。由于可信平台模块尚没有一个形式化模型用以支持测试用例的自动生成,因此该部分的工作目前还是空白。

3 TPM 形式化模型

形式化方法能精确、无二义地描述规范,有助于提高安全系统的质量,以便于进行严格的测试。同时建模是一致性测试的基础,也是对其进行形式化验证的基础,对 TPM 的建模也有助于达到通用准则(common criteria)的第 5 级标准,因此对可信平台模块进行形式化建模是十分必要的。

TPM 是一个相当复杂的系统,各个功能模块之间相对独立,依存度不高,因此可以采取各个功能模块单独建模的方法。本文采用扩展有限状态机对可信平台模块进行建模,为了节省篇幅,在下面的讨论中,将主要关注可信平台模块密码子系统的建模,首先给出密码子系统的 Z 规格说明,然后根据该规格说明给出对应的 EFSM 图(扩展有限状态机)。TPM 的建模必须注意以下几个问题:

(1)模型是一个抽象的功能模型,刻划了被建模系统的外部可见行为,但是并不对具体的实现细节进行建模;

(2)模型必须能支持测试用例的生成,并且被建模系统的输入、输出在模型中体现出来;

(3)模型必须不能太庞大,只包括必须的部分。

3.1 TPM 密码子系统

在 TPM 1.2 规范^{①②}中,TPM 提供了基本的密码操作,主要的密码操作有 RSA 的密钥生成,加密、解密操作,RSA 的签名操作,同时 TPM 提供了封装存储的功能。其中主要有 3 类密钥:加密密钥、封装密钥和签名密钥。不同的密钥能执行不同的操作,如封装密钥能执行 Seal 和 UnSeal 的操作。

3.2 Z 语言描述

本节将以 TPM 的密码子系统作为建模的对象,关于该部分的非形式化的描述见脚注②。

在具体建模过程中,由于 TPM 规范中指定的具体数据结构比较复杂,需要进行一定程度的数据

② TCG. TPM Main Specification Version 1.2. www.trusted-computinggroup.org/specs/TPM/, 10/2003

抽象. 从状态上看, TPM 的初始状态是已经建立了属主身份(take owner), 并且 TPM 中将会保存有密钥的相关信息, 最重要的是密钥的句柄以及密钥的属性(如类型), 因此, 引入如下的抽象数据类型和全局变量:

[KeyHandle, KeyType]

maxcount: \mathbb{N} ;

KeyType := TPM_STORAGE | TPM_SIGN | TPM_BIND;

其中 KeyHandle 是 TPM 内部密钥的句柄类型; KeyType 是密钥的类型, 有 3 种不同的密钥类型.

——TPMState——
 $keys: \mathbb{P} \text{KeyHandle}$
 $keyHasType: \text{KeyHandle} \mapsto \text{KeyType}$
 $\text{dom } keyHasType := keys$
 $\#keys \leq \text{maxcount}$

——Init——
 $keys = \{srkKeyHandle\}$
 $keyHasType = \{srkKeyHandle \mapsto \text{TPM_STORAGE}\}$

图 1 系统抽象状态和初始化定义

——TPMCreateKey——
 $\exists \text{TPMState}$
 $\text{parentHandle?}: \text{KeyHandle}$
 $\text{type?}: \text{KeyType}$
 $\text{result!}: \mathbb{N}$

 $\exists h: \text{KeyHandle} | h = \text{parentHandle?} \wedge$
 $h \mapsto \text{TPM_STORAGE} \in \text{keyHasType}$
 $\text{result!} = 0$

——TPMEvictKey——
 $\Delta \text{TPMState}$
 $\text{evictHandle?}: \text{KeyHandle}$
 $\text{result!}: \mathbb{N}$

 $keys \neq \emptyset$
 $\text{evictHandle?} \in keys$
 $keys' = keys / \{\text{evictHandle?}\}$
 $keyHasType' = \{\text{evictHandle?}\} \triangleleft keyHasType$

——TPMLoadKey——
 $\Delta \text{TPMState}$
 $\text{parentHandle?}: \text{KeyHandle}$
 $\text{type?}: \text{KeyType}$
 $\text{result!}: \mathbb{N}$
 $\text{keyHandle!}: \text{KeyHandle}$

 $\exists h: \text{KeyHandle} | h = \text{parentHandle?} \wedge$
 $h \mapsto \text{TPM_STORAGE} \in \text{keyHasType}$
 $\text{result!} = 0$
 $\text{inKeyHandle!} \neq \forall x: x \in \text{KeyHandle} \wedge x \notin keys$
 $keys' = keys \cup \{\text{inKeyHandle?}\}$
 $keyHasType' = \text{keyHasType} \oplus \{\text{inKeyHandle} \mapsto \text{type?}\}$

图 2 密钥生成、载入、载出操作模式

——TPMUnSeal——
 $\exists \text{TPMState}$
 $\text{keyHandle?}: \text{KeyHandle}$
 $\text{result!}: \mathbb{N}$

 $\text{keyHandle?} \in keys \wedge \text{keyHasType}(\text{keyHandle?}) = \text{TPM_STORAGE}$
 $keys' = keys$
 $keyHasType' = \text{keyHasType}$

——TPMSeal——
 $\exists \text{TPMState}$
 $\text{keyHandle?}: \text{KeyHandle}$
 $\text{result!}: \mathbb{N}$

 $\text{keyHandle?} \in keys \wedge \text{keyHasType}(\text{keyHandle?}) = \text{TPM_STORAGE}$
 $keys' = keys$
 $keyHasType' = \text{keyHasType}$

图 3 封装和解封操作模式

3.3 EFSM 模型

定义 1. 扩展的有限状态机(Extended Finite State Machine, EFSM). M 定义为一个六元组 $\langle S, S_0, I, O, T, V \rangle$, 其中 S 是一个非空的状态集合, S_0 是初始状态, I 是一个非空的输入消息集合, O 是一个非空的输出消息集合, V 是变量集合, 对于任意的 $t \in T$, t 是一个六元组 (s, x, P, op, y, s') , 其中

从操作上看, 按照脚注②中的描述, 主要有创建密钥、销毁密钥、使用密钥等操作. 密码子系统抽象状态以及系统初始化定义的描述如图 1 所示, 其中函数 $keyHasType$ 将密钥句柄映射成不同的密钥类型, 初始时 TPM 内部有存储根密钥 SRK.

图 2 的操作模式给出了密钥创建、载入、载出操作.

图 3 给出了 TPM 的封装操作以及解封操作的操作模式, 其它关于 TPM 密钥的操作如解密、签名操作可用类似的形式给出③.

$s, s' \in S$ 分别是初始状态和终止状态; $x \in I$ 是状态迁移 t 的输入; $y \in O$ 是状态迁移 t 的输出; P 是状态迁移 t 的前置条件, 可能为空; op 是状态迁移中的操作, 其中由一系列的输出语句和变量赋值语句组成.

③ 由于加密、签名验证操作是通过公钥进行的, 因此这些操作并不由 TPM 完成, 是在 TPM 外部完成的.

基于 3.2 节给出的 Z 语言规格说明,下面通过状态提取和迁移提取给出对应的 EFSM 图. TPM 状态根据 Z 规格说明的抽象状态进行划分. 本文首先给出一种基于类型的状态划分方法.

3.3.1 状态提取的一般方法

设决定状态空间的状态变量为 $x_1, x_i, x_n, A_1..A_j, A_l$, 其中 x_i 表示的是单值变量, 类型分别为 T_1, \dots, T_n , 而 A_j 是一个集合变量, 集合中的元素类型为 TT_1, \dots, TT_ℓ (类型决定了变量的取值空间, 本文不区分类型和取值空间)

(1) 初始状态空间为

$$S_{\text{Initial}} \triangleq \{x_1, \dots, x_n, A_1, \dots, A_l \mid x_1 \in T_1, \dots, x_n \in T_n, A_1 \in \mathbb{P} TT_1, \dots, A_\ell \in \mathbb{P} TT_\ell\};$$

(2) 状态的细分

下面根据状态变量的不同对状态进行细分:

① 单值变量

基于一定的策略对单值变量 x_i 的取值进行划分. 通过这种方法可以将状态进一步的细分. 基于策略 *policy*, 对状态 S 关于状态变量 x_i 的划分定义为 $\text{Partition}(S, x_i, \text{policy}) = \{AS_1, \dots, AS_j, \dots, AS_{m_i} \mid$

$$AS_j \triangleq (P_j(x_i) = \text{true} \wedge \forall AS_{j_1}, AS_{j_2}: AS_{j_1}(x_i) \cap AS_{j_2}(x_i) = \emptyset \wedge \bigcup_{i=1}^{m_i} AS_i = T_i)\},$$

其中 *policy* 指的是采用的策略方法, 如边界值分析法、类别划分方法等; AS_j 表示状态 S 经过状态变量 x_i 细分之后的子状态, P_j 是对 x_i 变量进行约束的谓词逻辑, 如 $x_i \geq 0$; $AS_{j_1}(x_i)$ 表示状态 AS_{j_1} 中状态变量 x_i 的取值空间.

② 集合变量

对集合变量, 采用基于类型的划分方法进行状态空间的划分. 下面介绍一种基于函数的集合划分^[16]方法.

若存在由集合 T 到类型 V 的函数 $f: T \rightarrow V$, 且类型 V 是有限集, $\text{ran}(f) = \{v_1, v_2, \dots, v_n\}$. 则类型 T 的函数值划分

$$\pi = \{\{\forall t: T \mid f(t) = v_1\}, \{\forall t: T \mid f(t) = v_2\}, \dots, \{\forall t: T \mid f(t) = v_n\}\}.$$

因此函数 f 按照状态集合变量 A_j 可对状态 S 进行划分, 得到划分后的结果为

$$\text{Partition}(S, A_j, f) =$$

$$\{BS_1, \dots, BS_{k_j} \mid \{\forall t: t \in BS_1(A_j) \mid f(BS_1(A_j)) = v_1\} \dots \{\forall t: t \in BS_{k_j}(A_j) \mid f(BS_{k_j}(A_j)) = v_{k_j}\}\},$$

其中 $BS_1(A_j)$ 表示状态 BS_1 中集合变量 A_j 的取值空间中类型为 TT_j 的元素集合. 各个划分之间互相组合得到最后的状态空间. 划分之间进行状态组合得

到状态空间的个数为 $C_{k_j}^1 + C_{k_j}^2 + \dots + C_{k_j}^{k_j} = 2^{k_j} - 1$.

③ 组合状态变量

如果内部存在的状态变量既包含单值变量 x_1, \dots, x_n , 又包含集合变量 A_1, \dots, A_ℓ , 那么最后的状态空间是各个变量之间的完全组合. 得到的状态数为

$$\prod_{i=1, j=1}^{i=n, j=\ell} m_i \cdot (2^{k_j} - 1).$$

(3) 状态的缩减

从上面的分析可以看出, 如果状态变量的个数比较多, 那么其存在的状态空间将会急剧增加, 一种解决办法是在状态细分这一步中, 控制各个状态变量的划分粒度; 另一种解决办法是根据需求 (如测试需求) 对最后产生的状态空间进行限制, 下面通过 TPM 状态的提取来说明基于类型的状态划分方法.

3.3.2 TPM 状态的提取

在 3.2 节给出的 Z 语言规格说明中, 只有一个集合变量 *keys*, 并且存在一个约束函数 *keyHasType*.

(1) 第 1 步: 给出初始状态

$$S_{\text{Initial}} \triangleq \{keys \mid \#keys \leq \text{maxcount}\};$$

(2) 第 2 步: 状态的细分 (集合变量)

$$\text{Partition}(S_{\text{Initial}}, keys, \text{keyHasType}) = \{S_{\text{Sign}}, S_{\text{Storage}}, S_{\text{Bind}}\},$$

其中 S_{Sign} 内的所有元素密钥句柄都为签名密钥, 其它的定义类似.

(3) 第 3 步: 状态的组合

$$\text{状态组合数 } n = C_3^2 + C_3^1 + C_3^3 = 2^n - 1 = 7,$$

因此最后给出的状态为

$$\begin{aligned} s_1 &= \{S_{\text{Sign}}\}, s_2 = \{S_{\text{Storage}}\}, s_3 = \{S_{\text{Bind}}\}, \\ s_4 &= \{S_{\text{Sign}}, S_{\text{Storage}}\}, s_5 = \{S_{\text{Storage}}, S_{\text{Bind}}\}, \\ s_6 &= \{S_{\text{Bind}}, S_{\text{Sign}}\}, s_7 = \{S_{\text{Bind}}, S_{\text{Sign}}, S_{\text{Storage}}\}. \end{aligned}$$

(4) 第 4 步: 状态的缩减

如果加上约束条件 $P(S) \triangleq \{S \mid \forall key \in keys, \text{keyHasType}(key) = \text{TPM_SIGN}\}$, 状态空间可以缩减为 4 个, 分别为 s_1, s_4, s_6, s_7 .

3.3.3 迁移的提取

图 4 给出了状态迁移的提取算法.

其中 $\text{def}(s) = \{key \mid key \in s\}$, S 为内部状态的集合, OP 表示 TPM 操作的集合, 在密码子系统中, $OP = \{\text{TPMCreateKey}, \text{TPMLoadKey}, \text{TPMEvictKey}, \text{TPMSeal}, \text{TPMUnSeal}, \text{TPMSign}, \text{TPMVerify}, \text{TPMEncrypt}, \text{TPMDDecrypt}\}.$

最后得到的 EFSM 图如图 5 所示.

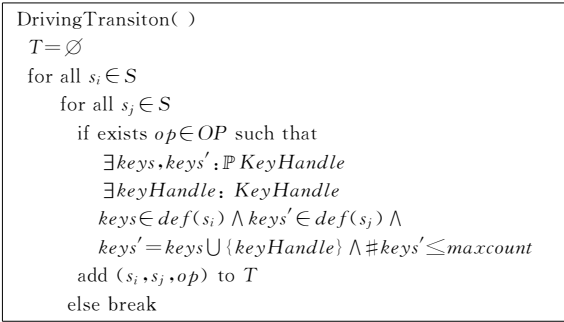


图 4 状态迁移的提取算法

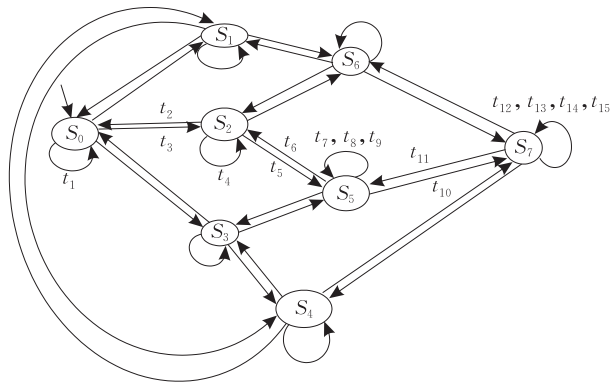


图 5 密码子系统的 EFSM 图

具体的状态含义以及状态迁移见附录,附录中的标记遵循文献[6]中提出的标记法,附录中只给出了其中的一条路径的迁移,其它路径的迁移是类似的.在迁移的标记($s-x, P/op, y \rightarrow s'$)中, s 表示当前状态, x' 表示迁移后的状态, x 表示输入, P 表示转移的条件, op 表示转移中的操作, y 表示输出.

4 模型的应用

TPM 的 EFSM 形式化模型可以应用于多种技术,如图 6 所示,可利用模型检测工具对 EFSM 形式化模型进行功能验证,同时 EFSM 模型是形式化测试的基础,本文着重对基于 EFSM 的测试方法进行深入的探讨.

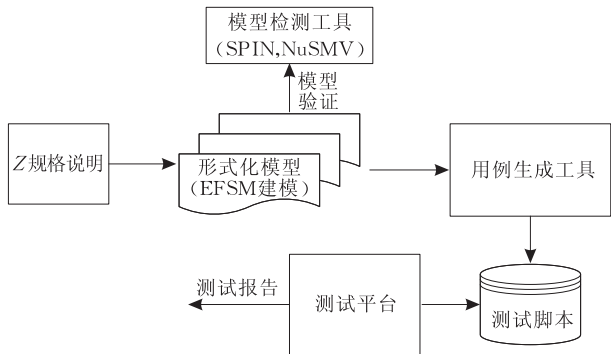


图 6 模型的应用

4.1 基于 EFSM 的形式化测试

4.1.1 测试用例的生成

本节的测试用例生成方法采用两阶段生成的方法,测试用例的生成分为两步:第 1 步通过算法自动生成抽象测试用例,抽象测试用例是不能执行的;第 2 步将抽象测试用例具体化为可执行的测试用例,在这一步中需要填入具体的测试数据.采用两阶段的方法更有利于测试方案的实施,更有利于模块化的部署.由于 TPM 命令的复杂性,第 2 步测试数据的生成目前还不能完全进行自动化的生成,需要人工的参与.

在 TPM 的符合性测试时需要注意到的问题是:在测试一些模块时,不需要用户显式地要求 TPM 产生授权会话,这是由用例工具自动生成的,也是一个基本的假设条件;通过 EFSM 模型生成的测试用例只能对 TPM 的抽象功能(主要是 TPM 规范的第一部分)进行符合性测试,并不能对具体的实现接口进行参数化的测试;各个不同的子系统的测试是有先后顺序的,如密码子系统依赖于授权协议管理子系统,只有先对授权协议管理子系统进行测试之后才能对密码子系统进行测试,图 7 表明了这种依赖关系.

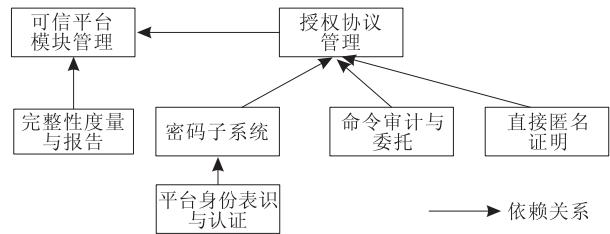


图 7 TPM 测试的模块之间的先后关系图

覆盖度是衡量测试用例完备性的一个重要手段.在一致性测试中,状态覆盖度和迁移覆盖度是最常见的覆盖标准.

定义 2(完全状态覆盖, all state coverage). 测试集完全状态覆盖状态变量 x 指的是对于 x 的任意取值,至少有一个测试用例覆盖到该值.测试集完全状态覆盖 EFSM 模型,指的是对于任意的状态变量 y ,测试集都完全状态覆盖变量 y .

EFSM 的可达性分析树,是一颗表达在所有的可能性输入的情况下,从初始节点出发扩展有限状态机的行为.对于每一个输入序列,该树包含一条从根出发的路径.可达性树是一个有向图,因此可以通过图论中的 DFS 或 BFS 方法对图进行遍历.

生成可达性分析树的算法如下:

1. 设置遍历搜索的深度 l ,从 EFSM 的指定初始节点出发对 EFSM 进行深度优先遍历,生成可达性分析树.

2. 在深度优先遍历过程中将遍历到的节点放入已遍历状态集合 $S_{\text{traversal}}$ 中.

3. 当遍历深度 $>l$ 时,停止可达性分析树的生成.

4. 先在可达性树中找到所有的可行路径,为每条可信路径指定具体的数据,主要指定的数据格式为〈命令号,随机产生的命令数据,预期值〉.每一条路径对应一个完整的测试用例.

图 5 和表 6 所表达的 EFSM 图有如下的示例路径. 通过该 EFSM 图生成的可达性图如图 8 所示, 在这个图中, 节点表示 EFSM 中的状态, 路径表示 EFSM 中的迁移, 黑色节点表示初始节点.

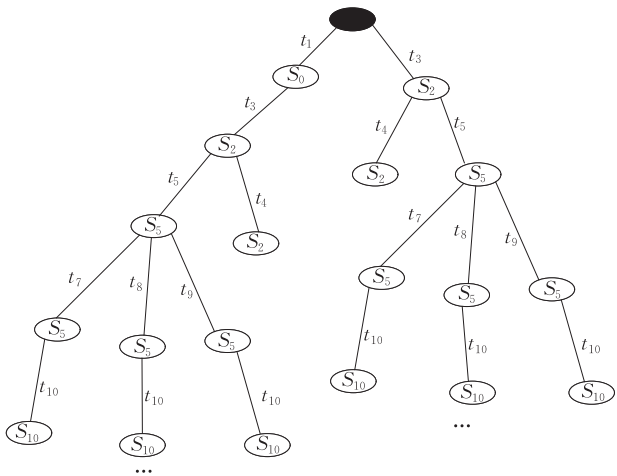


图 8 密码子系统的可达性图

覆盖率的计算公式为: $\#S_{\text{traversal}}/\#S_{\text{total}}$, 其中 S_{total} 表示总的状态空间.

4.1.2 实验结果

根据 4.1.1 节的测试用例自动生成实验,得到了状态覆盖标准下生成的不同子系统的用例数,如表 1 所示,该表表示的是覆盖度在 0.9 时生成的用例数,从表 1 中可以看出,子系统的复杂度越高,生

成的测试用例数越多,并且其生成的测试用例数尚在可接受的范围内.

表 1 覆盖度为 0.9 时生成的不同子系统的用例数

	状态覆盖标准
密码子系统	65
审计子系统	55
授权协议管理	63
TPM 管理	31
总数	214

图 9 则反映的是测试用例数与覆盖度之间的关系,从图 9 可以看出随着覆盖度的提高,生成的测试用例也不断地增加,但是在覆盖率为 0.8~0.9 时,测试用例数是比较合适的,如果再提高覆盖率测试用例数将会急剧增加。

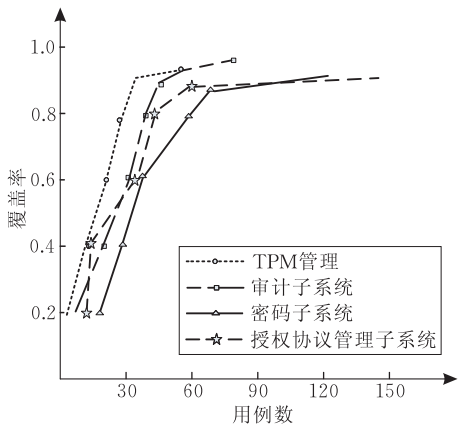


图 9 状态覆盖标准下的测试用例数与覆盖率的关系

表 2 表示的则是覆盖度为 0.9 时,采用状态覆盖标准时不同的子系统生成的测试用例的数目和性能.从表 2 中可以看出,各个子系统所建立的 EFSM 模型在可接受的范围内,并没有引起状态空间的爆炸.

表 2 状态覆盖标准下的用例生成结果

模型	转换数(transitions)	状态数(States)	CPU 执行时间/s④	测试用例数
密码子系统	32	8	40	65
审计子系统	24	4	36	55
授权协议管理	34	7	80	63
TPM 管理	25	4	25	31

5 测试结果分析

5.1 实验结果分析

最后,我们采用基于 EFSM 生成的测试用例对 TPM 进行符合性测试,表 3 显示的是在状态覆盖率为 0.9 的情况下,使用自动生成的测试用例进行测试得到的结果^⑤。从表 3 中可以看出,目前虽然各个厂商都声称推出了符合 TPM 1.2 的产品,但是实际

测试结果表明,还是存在与规范不一致的地方,这对于不同 TPM 之间的数据迁移和数据通信必然造成一定的影响,因此对 TPM 进行符合性和安全功能点的测试是十分必要的.从图 10 中可以看出,测试用例的增加与检测出的错误数成正比的关系.特别是测试用例数在 100~200 之间时,其检测率(错误

④ 运行环境为 CPU P4 3.2GHz, 内存 1GB.

⑤ 在表 3 中,Lenovo 的 TPM 是 1.1b 规范,其它的 TPM 是 1.2 规范.

数/用例数)比较高。

5.2 方案的比较

在文献[3]中 TPM 的测试采用的是测试包的形式进行的,针对 TPM 规范的每一条功能点,生成测试用例。这种方法缺少自动化工具的支持。文献[3]中的测试方案(简称 AMC 方案)与本文提出的方案的比较如表 4 所示。

表 3 3 种 TPM 检测出的错误数

	Atmel AT97SC3203	NSC TPM 1. 2	Lenovo TPM 1. 1b
密码子系统	13	17	11
审计子系统	0	1	0
授权协议管理	2	0	0
TPM 管理	0	2	0
总计	15	20	11

表 4 AMC 方案和本文的方案比较

方案	脚本支持	用例生成自动化	扩展性	覆盖度的分析	检测的错误数/用例数 (NSC TPM)
AMC 测试方案	不支持	×	需要修改源代码	不支持	16/400
本文的方案	支持	✓	修改 EFSM 模型	支持	20/214

6 结束语

本文对 TPM 的形式化分析和建模进行了深入的探讨,并且给出了 TPM 的扩展有限状态机模型,对该模型的应用特别是 TPM 的自动化测试进行了分析,并通过实验说明基于该模型的形式化测试的有效性。本文的贡献在于:(1)通过实验可以表明虽然基于 EFSM 的自动用例生成算法主要应用在协议测试领域,但是同样可以应用于 TPM 的符合性测试,生成的测试用例具有比较高的错误检测率;(2)对当前推出的符合 TPM 1.2 规范的可信平台模块进行了深入的测试,并且给出了测试结果的分析。在今后的工作中,将重点对 TPM 的命令接口测试做进一步的研究,以便更完整地支持 TPM 的符合性测试。

参 考 文 献

[1] Matthew Barrett. Towards an open trusted computing framework [M. S. dissertation]. University of Auckland, Auckland, 2005

[2] Bruschi Danilo, Cavallaro Lorenzo, Lanzi Andrea, Monga Mattia. Replay attack in TCG specification and solution// Proceedings of the 21th Annual Computer Security Application Conference (ACSAC). Tucson, AZ, USA, 2005: 127-137

[3] Sadeghi Ahmad-Reza, Selhorst Marcel, Stuble Christian, Wachsmann Christian, Winandy Marcel. TCG Inside? — A

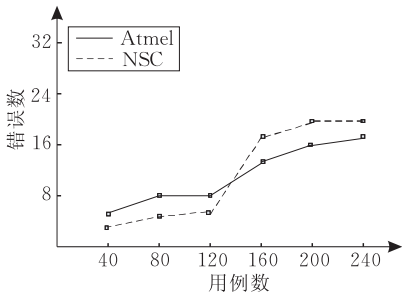


图 10 不同 TPM 的错误检测率(错误数/用例数)

从表 4 中可以看出,自动化的测试方案带来的好处是显而易见的,自动化测试方案不仅能提高错误的检测率,还具有较强的扩展性(规范变动以后,只需要修改 EFSM)。自动化测试方案主要的工作量在于 TPM 模型的分析与建立。

note on TPM specification compliance//Proceedings of the 1st ACM Workshop on Scalable Trusted Computing (STC'06). Virginia, USA, 2006: 1-10

[4] Fujiwara S, Bochmann G V, Khendek F, Amalou M, Ghedamsi A. Test selection based on finite state models. IEEE Transactions on Software Engineering, 1991, 17(6): 591-603

[5] Guerrouat Abdelaziz, Richter Harald. Reuse of test generation methods for embedded systems. Clausthal University of Technology, 2005

[6] Bourhfir C, Dssouli R, Aboulhamid E M. Automatic test generation for EFSM-based systems. The Journal of Real-Time Systems, 1989, 1(1): 27-60

[7] Petrenko A, Boroday S, Groz R. Confirming configurations in EFSM testing. IEEE Transactions on Software Engineering, 2004, 30(1): 29-42

[8] Celine Bigot, Alain Faivre, Christophe Gaston, Julien Simon. Automatic Test Generation on a (U)SIM Smart Card. LNCS 3928, 2006: 345-360

[9] Keum ChangSup, Kang Sungwon, Ko In-Young. Generating test cases for Web services using extended finite state machine. TestCom, New York, 2006: 103-117

[10] Blom Johan, Hessel Anders, Jonsson Bengt. Specifying and generating test cases using observer automata//Proceedings of the FATES 2004. Paul Pettersson, 2004: 125-139

[11] Bourhfir C, Aboulhamid E, Khendek F, Dssouli R. Test cases selection from SDL specifications. Computer Networks, 2001, 35(6): 693-708

[12] Bourhfir C, Dssouli R, Aboulhamid E, Rico N. Automatic executable test case generation for EFSM specified protocols//Proceedings of the IWTC'S'97. 1997: 75-90

[13] Zhang Yong, Qian Le-Qiu, Wang Yuan-Feng. Automatic testing data generation in the testing based on EFSM. Chinese Journal of Computers, 2003, 26(10): 1295-1303 (in Chinese)

(张涌, 钱乐秋, 王渊峰. 基于扩展有限状态机测试中测试输入数据自动选取的研究. 计算机学报, 2003, 26(10): 1295-1303)

[14] Huang Liang, Feng Deng-Guo, Zhang Min. A generation tool of test case based on the security model. Journal of the Graduate School of the Chinese Academy of Sciences, 2006, 24(3): 300-306(in Chinese)

(黄亮, 冯登国, 张敏. 一个基于安全模型的测试用例生成工具. 中国科学院研究生院学报, 2006, 24(3): 300-306)

附 录.

下面本文将根据密码学子系统的 Z 规格说明给出对应的 EFSM 图. 其中涉及到的符号有

- 输入变量集合: $x_{tag}, x_{ordinal}, x_{keyHandle}, x_{keyType}$;
- 输出变量集合: $y_{tag}, y_{retValue}, y_{ordinal}, y_{keyHandle}, y_{keyType}$.
- 输入命令有: TPM_CWK(产生密钥), TPM_EK, TPM_LK, TPM_UB, TPM_Seal, TPM_UnSeal, TPM_Sign, 符号 $y_{tag}(value)$ 表示对变量 y_{tag} 赋值 value.
- 环境变量和全局变量集合: $keys$ (TPM 中的句柄集合) 等.

附表 1 EFSM 状态描述

状态	描述
s_0	初始状态, 已经 TakeOwner, 拥有 SRK, 并且 TPM 是 enabled, 处于 TPM 的 s_1 操作状态
s_1	TPM 内部都是加密密钥
s_2	TPM 内部都是签名密钥
s_3	TPM 内部都是存储密钥
s_4	既有加密密钥, 也有存储密钥
s_5	内部既有签名密钥, 又有存储密钥
s_6	既有加密密钥, 又有签名密钥
s_7	同时有加密密钥、签名密钥、存储密钥

附表 2 状态转换表

状态迁移	描述
t_1	$\{s_0 - TPM_CWK, P_{t3} / \langle y_{retValue}(0), y_{ordinal}(TPM_CWK), y_{tag}(RspTag), y_{keyType}(x_{keyType}) \rangle, 0 \rightarrow s_0\}$ 其中 $P_{t3} = (x_{tag}(auth1Tag), x_{keyHandle} \in keys)$
t_2, t_6, t_{11}	$\{s_2(s_5, s_7) - TPM_EK, P_{t2} / \langle y_{retValue}(0), y_{ordinal}(TPM_EK), y_{tag}(RSPTag) \rangle, 0 \rightarrow s_0(s_2, s_5)\}$ 其中 $P_{t2} = \langle x_{tag}(reqAuth), x_{keyHandle} \in keys \rangle$
t_3, t_5, t_{10}	$\{s_0(s_2, s_5) - TPM_LK, P_{t3} / \langle y_{retValue}(0), y_{ordinal}(TPM_LK), y_{tag}(resAuth1) \rangle, 0 \rightarrow s_2(s_5, s_7)\}$ 其中 $P_{t3} = (x_{tag}(reqAuth1), x_{pkeyHandle} \in keys, x_{keyType}(SignKey))$
t_4, t_7, t_{12}	$\{s_2(s_5, s_7) - TPM_Sign, P_{t16} / \langle y_{tag}(rspAuth2), y_{retValue}(0), y_{ordinal}(TPM_Sign) \rangle, 0 \rightarrow s_2(s_5, s_7)\}$ 其中 $P_{t16} = (x_{tag}(reqAuth2))$
t_8, t_{13}	$\{s_5(s_7) - TPM_Seal, P_{t10} / \langle y_{tag}(rspAuth1), y_{retValue}(0) \rangle, 0 \rightarrow s_5(s_7)\}$ 其中 $P_{t10} = (x_{tag}(reqAuth1), x_{keyHandle} \in keys \wedge x_{keyType}(TPM_STORAGE))$
t_9, t_{14}	$\{s_5(s_7) - TPM_UnSeal, P_{t11} / \langle y_{tag}(rspAuth2), y_{retValue}(0), y_{ordinal}(TPM_UnSeal) \rangle, 0 \rightarrow s_5(s_7)\}$ 其中 $P_{t11} = (x_{tag}(reqAuth2))$
t_{15}	$\{s_7 - TPM_UB, P_{t9} / \langle y_{tag}(repAuth1), y_{retValue}(0), y_{ordinal}(TPM_UB) \rangle, 0 \rightarrow s_7\}$ 其中 $P_{t9} = (x_{tag}(reqAuth1), x_{keyHandle} \in keys)$



CHEN Xiao-Feng, born in 1980, Ph. D. candidate. His research interests include information system and security, trusted computing.

Background

This work is supported by National Natural Science Foundation of China under grant No. 60673083, No. 60603017 and National Basic Research Program of China (973 Plan) under grant No. 2007CB311202.

In trusted computing platform proposed by TCG group, the TPM is the core component the functional testing and validation is an important method to ensure the correctness and specification compliance of the trusted platform module, but until now, there is no valid formal testing and validation

method, meanwhile the specification which is given by trusted computing group is descriptive, it is not convenient for product development and testing.

This paper firstly analyzes the problem of the trusted platform module, and then gives the Z specification of TPM's cryptography system, based on this formal specification, gives the extended finite state machine model. Finally, the authors use the EFSM model for generating the test case and analyze the testing results.