国 际 电 信 联 盟

ITU-T J.1013

(04/2020)

国际电信联盟
电信标准化部门

系列J：有线网络和电视、声音节目及其他多媒体信号的传输

有条件访问和保护 – 可交换嵌入式有条件访问和数字版权管理解决方案

可交换CA/DRM解决方案的嵌入式通用接口；虚拟机

ITU-T J.1013 建议书

国际电信联盟

ITU-T J.1013 建议书

可交换CA/DRM解决方案的嵌入式通用接口；虚拟机

摘要

ITU-T J.1013建议书是由多个部分组成的可交付成果中的一部分，涵盖有关可交换有条件访问/数字版权管理（CA/DRM）解决方案规范的嵌入式通用接口（ECI）的虚拟机。

本ITU-T建议书是ETSI标准ETSI GS ECI 001-4的转换，是ITU-T SG9与ETSI ISG ECI之间合作的结果。在第7.3.7.1条中做了一些小的修改。

沿革

版本	建议书	批准日期	研究组	唯一标识（ID）*
1.0	ITU-T J.1013	2020-04-23	9	11.1002/1000/13574

关键词

有条件访问（CA）、数字版权管理（DRM）、交换

* 为获取本建议书，请在网页浏览器内键入URLhttp://handle.itu.int/，然后输入唯一ID。例如，http://handle.itu.int/11.1002/1000/11830-en。

前言

国际电信联盟（ITU）是从事电信领域工作的联合国专门机构。ITU-T（国际电信联盟电信标准化部门）是国际电信联盟的常设机构，负责研究技术、操作和资费问题，并且为在世界范围内实现电信标准化，发表有关上述研究项目的建议书。

每四年一届的世界电信标准化全会（WTSA）确定ITU-T各研究组的研究课题，再由各研究组制定有关这些课题的建议书。

WTSA 第 1号决议规定了批准建议书须遵循的程序。

属 ITU-T 研究范围的某些信息技术领域的必要标准，是与国际标准化组织（ISO）和国际电工技术委员会（IEC）合作制定的。

注

本建议书为简明扼要起见而使用的“主管部门”一词，既指电信主管部门，又指经认可的运营机构。

遵守本建议书的规定是以自愿为基础的，但建议书可能包含某些强制性条款（以确保例如互操作性或适用性等），只有满足所有强制性条款的规定，才能达到遵守建议书的目的。“应该”或“必须”等其它一些强制性用语及其否定形式被用于表达特定要求。使用此类用语不表示要求任何一方遵守本建议书。

知识产权

国际电联提请注意：本建议书的应用或实施可能涉及使用已申报的知识产权。国际电联对无论是其成员还是建议书制定程序之外的其它机构提出的有关已申报的知识产权的证据、有效性或适用性不表示意见。

至本建议书批准之日止，国际电联尚未收到实施本建议书可能需要的受专利保护的知识产权的通知。但需要提醒实施者注意的是，这可能并非最新信息，因此特大力提倡他们通过下列网址查询电信标准化局（TSB）的专利数据库：http://www.itu.int/ITU-T/ipr/。

目录

引言

本ITU-T建议书[1]是ETSI标准[b-ETSI GS ECI 001-4]的转换，是ITU-T SG9与ETSI ISG ECI之间合作的结果。在第7.3.7.1条中做了一些小的修改。

本建议书的目的是促进电子通信服务、尤其是广播和视听设备市场的互操作性和竞争。然而，根据成员国的情况，还可以使用其他技术，这些技术也可能是适当的和有益的。

本建议书描述了可在沙盒中执行并提供一系列指令和系统调用功能的虚拟机（VM）概念。虚拟机旨在在多种环境中工作，它利用一些定义明确的接口可与存在于同一机器上的其他应用程序进行互操作。它既能为其自身的指令集提供一系列支持，也能为执行**ECI主机**中央处理单元（CPU）的**本机代码**[2]中所写的要素以及与硬件和其他**ECI主机**环境中的要素进行交互提供模块化机制，从而为虚拟机随时执行可再生代码提供途径，可再生代码可提供包括CA/DRM客户端实施在内的、广泛的潜在安全应用。

1 附录I确定了一些有待进一步发展的领域。

2 在本建议书的文本中，使用黑体字来表示具有特定于嵌入式通用接口上下文的定义的术语，它们可能与通常用法有所不同。

ITU-T J.1013 建议书

可交换CA/DRM解决方案的嵌入式通用接口；虚拟机

1 范围

本建议书所描述的虚拟机旨在纳入数字电视接收器和机顶盒的实施方案中，它能为条件访问内核或数字版权管理客户端应用的执行提供一个安全环境。目的是提供一个统一的执行环境，在这种环境中，此类客户端可以在满足最低**ECI主机**性能要求的情况下进行操作，并提供一个标准API，以用于从内容（即用内容进行封装）中检索基本的安全数据，或者通过外部网络（例如，互联网）进行访问，并且可以以标准化的方式从**ECI主机**环境来访问资源。另请参阅[b-ITU-T J.1010]和[b-ITU-T J.1011]。

虚拟机的存在和使用可实现CA/DRM客户端的自由交换，并支持**ECI主机**中此类客户端的多个同步实例，从而使用户和运营商不必绑定于某个特定的内容保护（CP）提供商，而是可使用不同类型的安全解决方案来适应不同的内容类型。对于内容保护系统的提供商，它可确保可访问已知的执行平台，从而不需要与**ECI主机**设备的任何厂商进行特定的融合。

2 参考文献

下列ITU-T建议书和其他参考文献的条款，通过在本建议书中的引用而构成本建议书的条款。在出版时，所指出的版本是有效的。所有的建议书和其他参考文献都面临修订，使用本建议书的各方应探讨使用下列建议书或其他参考文献最新版本的可能性。当前有效的ITU-T建议书清单定期出版。本建议书引用的文件自成一体时不具备建议书的地位。

[ITU-T J.1012] ITU-T J.1012建议书（2020年），可交换CA/DRM解决方案的嵌入式通用接口；CA/DRM容器、载入器、接口和撤销。

[ETSI GS ECI 001-4] ETSI GS ECI 001-4 V1.1.1（2017年），可交换CA/DRM解决方案的嵌入式通用接口（ECI）；第4部分：虚拟机。

3 定义

3.1 他处定义的术语

无。

3.2 本建议书定义的术语

本建议书中定义了以下术语：

3.2.1 字节码Bytecode：虚拟机（VM）所执行的**ECI客户端**代码（一般包括有条件访问的内核或是数字版权管理的客户端）。

3.2.2 客户驻地设备customer premises equipment（CPE）：提供嵌入式通用接口（**ECI**）规定的解密和加密功能的客户设备。

3.2.3 ECI（嵌入式CI）ECI（Embedded CI）：ETSI ISG“嵌入式CI”中规定的架构和系统，它有助于在客户驻地设备（**CPE**）中开发和实施基于软件的可交换**ECI客户端**，因此，在**ECI**方面实现**CPE**的互操作性。

3.2.4 ECI客户端（嵌入式ECI客户端）ECI Client（Embedded CI Client）：实施符合嵌入式通用接口（ECI）规范的有条件访问/数字版权管理（CA/DRM）客户端。

3.2.5 ECI主机ECI Host：CPE的硬件和软件系统，它包含**ECI**相关功能性并拥有面向**ECI客户端**的接口。

3.2.6 本机代码Native Code：在**ECI主机**处理器的本地可执行指令集中所写下的编程代码

3.2.7 虚拟机实例VM Instance：ECI主机建立的虚拟机（VM）的实例化，在ECI客户端看来是在其中进行操作的执行环境。

4 缩略语和首字母缩写词

本建议书采用下列缩略语和首字母缩写词：

API 应用编程界面

CA 条件接收

CAS 有条件访问系统

CI 通用接口

CP 内容保护

CPE 客户驻地设备

CPU 中央处理单元

DRM 数字版权管理

ECI 嵌入式通用接口

ECP 增强型内容保护

ELF 可执行和可连接的格式

EPG 电子程序指南

ID 识别/身份/标识符

OS 操作系统

OTT 机顶盒

PC 程序计数器

POSIX 可移植操作系统接口

RISC 精简指令集计算机

VM 虚拟机器

5 惯例

在本建议书中，使用加黑且首字母为大写的术语来表示那些以ECI特定的含义定义的术语，它们可能与这些术语的通常使用惯例相偏离。

6 概念性原理

6.1 作为CPU的虚拟机

本质上，虚拟机（VM）包括一个含有自身代码和数据内存的中央处理单元（CPU）以及一套可接入**ECI主机**机器硬件性能的系统接口。这一仿真的CPU以虚拟的32位CPU的方式来执行代码，在本建议书中，其所执行的代码称之为**字节码**。由于虚拟机模拟的是通用型的精简指令集计算机（RISC）处理器，因此能够执行不同的应用程序。

6.2 虚拟机的特点

虚拟机应提供单工序、单线程环境。

ECI主机硬件的接口和其他功能是通过标准的调用存储库形式而提供的，称之为SYSCALL。SYSCALL指令是虚拟机定制指令中的一个，通常在准备好库存程序所需的参数（如，在虚拟机的“寄存器”中通过）之后执行。

ECI客户端和**ECI主机**间的所有交互是通过这一操作而实现的。这里并无定义任何**中断**架构，因**ECI客户端**一旦启动，即运行到结束。因此，并无实现调用到虚拟机内的机会。尽管灵活性受到某种程度的限制，但对虚拟机执行的控制得到了提升（确保了运行的稳健性），同时避免了竞态条件，也改善了与限时操作等的交互。

因此，将数据或信息传递给在虚拟机中执行的**ECI客户端**的唯一途径是基于**ECI客户端**所发出的请求，通过调用相应的SYSCALL来实现的。

6.3 各个ECI客户端间的隔离

ECI客户端在虚拟机中作为**ECI主机**固件中运行的一个应用而存在和执行。可能会调用虚拟机多个实例，每个可能运行不同的**ECI客户端**。这对**ECI主机**的操作环境提出3个基本要求：

1) 隔离单个ECI客户端 – 操作系统（OS）须为每个虚拟机实例分配足够的资源，以便性能要求可由同时运行的所有实例满足；建议的性能要求值如[b-ITU-T J-Suppl.7]中所示。

2) 第8条和附件C中所定义的库须是完全重入的或每个虚拟机实例之间是分开实施的。

3) 操作系统和虚拟机须确保运行中的**ECI客户端**与外部世界之间无信息交换，包括为此除明确详述方式以外且作为SYSCALL接口一部分的其他**ECI客户端**。这其中意味着所有映射到**虚拟机实例**数据空间内的内存都要提前抹去其之前的内容，应避免任何试图使用虚拟机中异常条件来激活未规定描述行为的行为。这还意味着**ECI客户端**不能改变其**字节码**。具体意味着**ECI主机**和虚拟机须进行所有要求的检查来避免**ECI客户端**在**ECI主机**或虚拟机实施中导致意外行为，例如，可能直接或间接造成**ECI客户端**能够操纵（黑客进入）**ECI主机**。

6.4 虚拟机详细说明

本建议书的以下条款是有关虚拟机本身的清晰详述：

1) 虚拟机的技术架构。

2) 虚拟机的指令集。

3) **ECI主机**的接口。

6.5 ECI客户端载入器

为执行**ECI客户端**，首先须在虚拟机内存的代码空间内载入**字节码**，并将数据空间初始化。第9条阐述了**ECI客户端**容器格式和虚拟机初始化等的一些具体问题。

7 虚拟机

7.1 执行环境

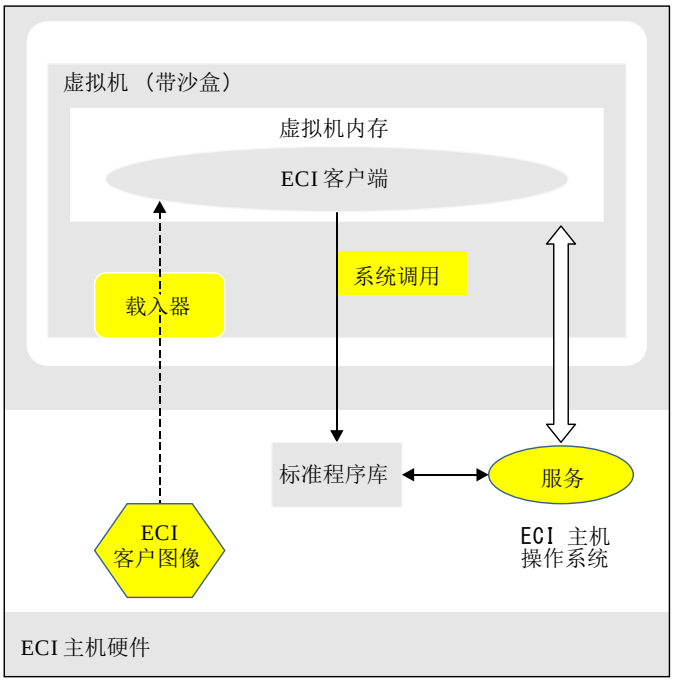

图1 – 虚拟机主机环境

如图1中所描述的，虚拟机须在沙盒环境中执行，以确保与**ECI主机**的操作系统、虚拟机的其他实例和**ECI主机**中执行的任何其他应用隔离。

虚拟机包括**ECI主机**一个具有相关内存的本机应用、接口库和用于安装组成**ECI客户端字节码**的载入器。接口库为**ECI客户端**提供接入**ECI主机**操作系统的功能和硬件，以及提供接入可在**ECI主机**内执行的其他应用和需与**ECI客户端**进行交互的其他应用。典型的例子是与电子程序指南（EPG）应用进行的交互需要展示给用户的具体内容处于授权状态。

7.2 虚拟机架构

7.2.1 CPU架构

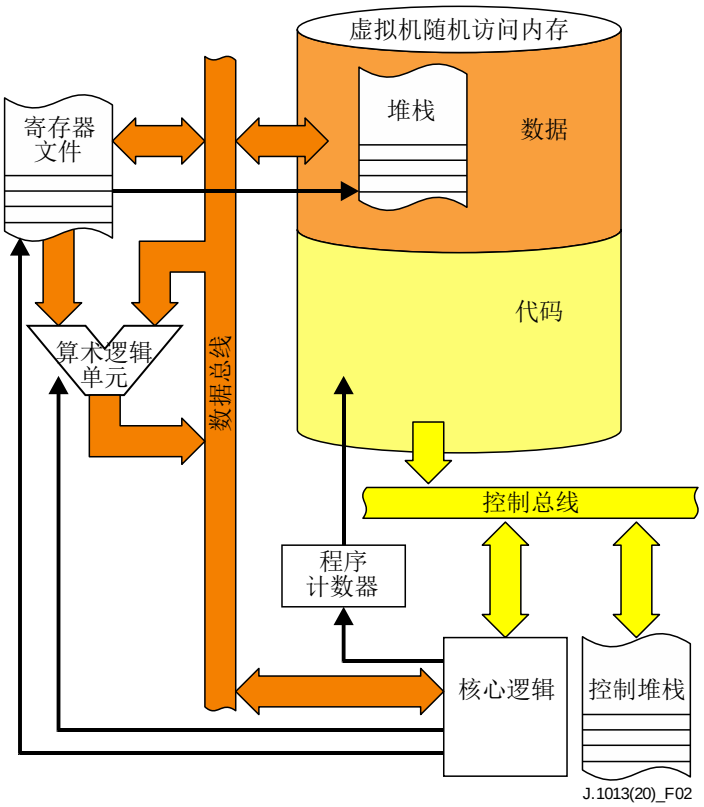

图2 – 虚拟处理器架构

图2显示的是虚拟机CPU的架构。虚拟机作为一种寄存机器，具有以下特点。

– 带有32位通用寄存器的寄存器文件。在寄存器窗口中组织寄存器。每个寄存器窗口包含32个寄存器。每个窗口的后16个寄存器与下一个窗口的前16个寄存器重叠。这些窗口寄存器中的2个用作栈指针和帧指针。寄存器文件中的寄存器总数为REGISTER_FILE_SIZE，详见附件A。

– 哈佛CPU架构。数据存储在32位平面内存空间里。代码存储在只读、非可编址的内存空间里。

– 分开的一个控制栈记录返回地址。该栈的内容是**字节码**或外部应用不可接入的。栈可储存至CONTROL_STACK_SIZE返回地址（见附件A）。

– 加载和存储符号字节和无符号字节、半字和字数据类型，即分别为8位、16位和32位。

– 由许多根据应用域定制的数据处理指令所组成的指令集。

– 不受字节序影响、可高效加载和储存的本机字节排序。自然对齐（对齐=大小）用于基本类型，可实现**字节码**最大化的便捷性。换句话说，半字的内存地址总是偶数，整字的地址总是4的倍数。

– 可用于实施系统服务的系统调用指令（SYSCALL）。这可使虚拟机的内置功能得到扩展，如本地执行那些经常出现的数据处理。
– 支持碎片化内存空间的分页内存。可使本机内存映射到虚拟机内存空间内。

7.2.2 寄存器

每个寄存器窗口中，可见32个寄存器，从R0到R31。预留2个寄存器供特别处理。R0为帧指针，R16为栈指针。关于这些寄存器的使用，以下会进一步详细说明。

在功能入口处，ENTER指令可使寄存器窗口上移16个寄存器。旧的栈指针变成新的帧指针，并形成新的栈指针和增加了15个可用的寄存器。从帧指针发出的ENTER指令所提供新的栈指针减去来完成对新栈指针的初始化。

“返回”指令可逆转该流程。可使窗口下移16个寄存器，从而恢复旧的帧指针和栈指针。

既然通过被调用例程无法达到原始的R0到R15，则作为被调用者进行自动保存。既然返回地址保存在一个分开的控制栈上，则作为被调用者所保存的寄存器和返回地址无可用的数据栈。

寄存器的真实数量是有限的，因此围绕**ECI客户端**的调用深度有最大值（CONTROL_STACK_SIZE）。超过这个深度就终止虚拟机的程序。在创建虚拟机流程时，可细化寄存器的数量和控制栈的对应深度。

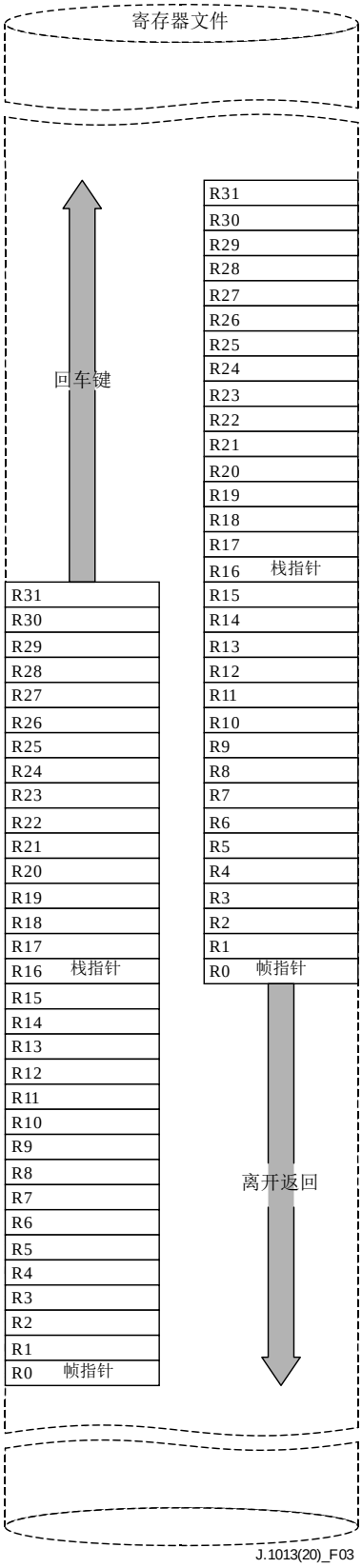
寄存器文件
回车键
R31
R30
R29
R28
R27
R26
R25
R24
R23
R22
R21
R20
R19
R18
R17
R16 栈指针
R15
R14
R13
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0 帧指针
R31
R30
R29
R28
R27
R26
R25
R24
R23
R22
R21
R20
R19
R18
R17
R16 栈指针
R15
R14
R13
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0 帧指针
离开返回
J.1013(20)_F03

图3 – 寄存器文档架构

7.2.3 数据空间

定义为DATA_BASE_ADDRESS（见附件A）的虚拟机数据基地址须是0x1000000 （16兆字节）。在可寻址存储器以上且不可寻址的最小地址是DATA_BASE_ADDRESS + ADDRESSABLE_DATA_SIZE（见附件A）。栈的基地址根据虚拟机实施而定义，但须朝向该地址空间的高端。虚拟机可在位于栈底部（更高的一个地址）“以下”的**ECI客户端**的地址空间内因保密性而保留VM_RESEVED_SIZE的最大值（见附件A）。虚拟机初始化时，栈指针须指向首个自由栈的位置。**ECI客户端**可确保初始化时的（空）堆的顶部等于初始化数据的大小+非初始化数据段的大小，四舍五入到4的倍数。

图4中所展示的是数据内存的布局。

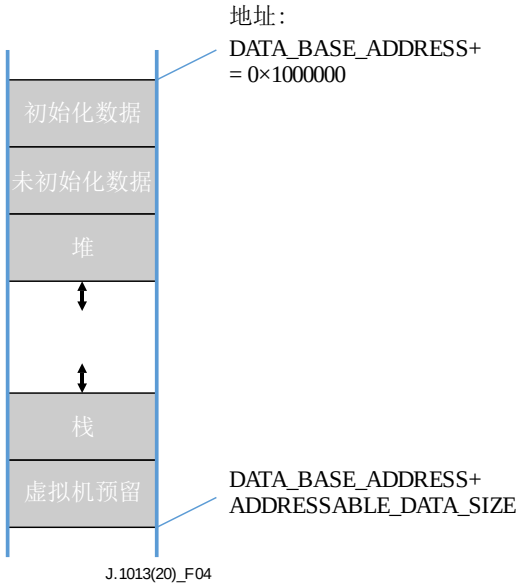

图4 - 虚拟机数据内存的布局

ECI客户端初始化时，**ECI客户端**载入器须将初始化和非初始化数据段加载到地址DATA_BASE_ADDRESS的开头。所有非初始化数据段的字节须设置为0。初始化的数据段没有进行写保护。

注1 – 堆栈大小最初是受限制的。由于c函数中所定义的本地数据结构一般分配给堆栈，则**ECI客户端**应设置合适的堆栈段大小，以便在c代码中使用大的本地变量。

注2 – **ECI主机**可将虚拟机预留的存储器中的信息缓冲映射到基堆栈地址的下面。

未来的虚拟机版本可保留更多的**ECI客户端**可寻址存储，即它们可具有更大的ADDRESSABLE_DATA_SIZE。为了实现反向兼容性，**ECI客户端**不能由这里定义的堆栈指针的具体值或值的范围所决定，但仅将堆栈指针用作初始化传递上。

ECI客户端载入器不能加载初始化和未初始化数据段中不符合以上存储器布局惯例的任何图片文档。

7.2.4 代码空间

程序不能直接获得代码。程序可获得静态代码对象的32位不透明引用（如例程入口点、跳转目标），称之为*代码引用*（见MOVF指令）。*代码引用*仅可与非直接控制流指令（JMPR

和CALLR）一起使用。*代码引用*不是代码内存空间的指针，指针运算不能代替它。

代码地址空间内代码段的起始地址须是0x00000000。代码段大小的最大值定义为CODE_SIZE（见附件A）。

7.2.5 堆栈

堆栈的传统定义位置是在数据存储器，仅包含字，逐渐向较低地址发展，且使其当前寄存器窗口的R16寄存器（*堆栈指针*）的尖端（最后推送的字）始终这一朝向。

R0是帧指针，用作进入被调用者堆栈的指针，以便通过被调用的例程即可获得参数或推送到堆栈上的其他数据（见第7.2.8条）。

7.2.6 字节序

多字节数据（半字和字）在系统内存的表现格式为小端模式。**ECI客户端**软件须使用小端格式。

7.2.7 异常处理

虚拟机的CPU在执行时不发布任何异常处理。若一指令在除本建议书所描述的以外的条件下运行（如非对齐访问内存中的半字或整字，访问无对应内存的任意内存地址，访问未知代码提示的分支），则行为是未定义的。虚拟机可选择终止内核。虚拟机应确保本建议书范围之外运行的**ECI客户端**任何条件都无法获得未授权数据或影响其他应用。

7.2.8 调用惯例

调用惯例通过了R17到R23的前7个标量参数（指针和整数）。被调用者将视其为R1到R7。

7个以外的标量参数按从左到右的顺序被调用者在堆栈上传递。因寄存器窗口的机制，被调用者将总是找到R0指向的第8个参数（若存在的话）。因此，R0是*帧指针*。结构参数总是在堆栈上传递，或按引用传递。指针总是指虚拟机的内存空间。

注 – 所有的SYSCALL仅通过引用传递任何结构和阵列。这一方法也用在其他调用上。

被调用者将返回值留在R1中，调用者视其为R17。小于32位的类型按32位值传递（和返回）。

结构返回的实施是传递第一个隐式参数，也就是可储存返回类型的内存区域的一个指针（按引用传递）。被调用者将其结果写到该参数所指的位置。返回指针被视为像标准自变数（R17→R1传递），表示函数的常规自变数，它返回一个结构，并转到其他调用惯例寄存器处（R18..R23→R2..R7）或通过堆栈。

7.3 虚拟机指令集

7.3.1 符号

以下符号用于：

```
rx       Register x.
uimm5    5 bit unsigned immediate.
uimms9   9 bit unsigned immediate. Always a multiple of two.
uimms10  10 bit unsigned immediate. Always a multiple of four.
simm11   11 bit signed immediate.
simm16   16 bit signed immediate.
uimm16   16 bit unsigned immediate.
pcr16    16 bit signed PC-relative
pcr24    24 bit signed PC-relative
```

```
imm32    32 bit immediate.
low8(x)  The least significant 8 bits of x.
low16(x) The least significant 16 bits of x.
```

函数描述使用的是32位整数的C语义。支持签名或未签名数据类型的操作能力被认为是注解。MEM1()、MEM2()和MEM4()提供存储访问，分别可访问1、2或4字节。其操作对象为进入数据段的偏移。当具有相关性时，MEM前缀为U时，是指未签名操作，为S时，指的是签名的扩展操作。

7.3.2 算术指令

7.3.2.1 寄存器寻址

```
ADD     r1,r2,rd        ; rd = r1 + r2;
SUB     r1,r2,rd        ; rd = r1 - r2;
OR      r1,r2,rd        ; rd = r1 | r2;
AND     r1,r2,rd        ; rd = r1 & r2;
XOR     r1,r2,rd        ; rd = r1 ^ r2;
SRA     r1,r2,rd        ; rd = r1 >> r2;     signed shift right
SRL     r1,r2,rd        ; rd = r1 >> r2;     logic shift right
SLL     r1,r2,rd        ; rd = r1 << r2;
MUL     r1,r2,rd        ; rd = r1 * r2;
SDIV    r1,r2,rd        ; rd = r1 / r2;      signed divide
SMOD    r1,r2,rd        ; rd = r1 % r2;      signed remainder
UDIV    r1,r2,rd        ; rd = r1 / r2;      unsigned divide
UMOD    r1,r2,rd        ; rd = r1 % r2;      unsigned remainder
EQ      r1,r2,rd        ; rd = r1 == r2;
NE      r1,r2,rd        ; rd = r1 != r2;
LT      r1,r2,rd        ; rd = r1 < r2;      signed less than
GE      r1,r2,rd        ; rd = r1 >= r2;     signed greater or equal
LTU     r1,r2,rd        ; rd = r1 < r2;      unsigned less than
GEU     r1,r2,rd        ; rd = r1 >= r2;     unsigned greater or equal
NOT     r1,rd           ; rd = ~r1;
NEG     r1,rd           ; rd = -r1;
ABS     r1,rd           ; rd = abs(r1);
MOV     r1,rd           ; rd = r1;
EXTB    r1,rd           ; rd = (int8_t) r1;   sign-extend from 8 bits
EXTH    r1,rd           ; rd = (int16_t) r1;  sign-extend from 16 bits
ZEXTB   r1,rd           ; rd = (uint8_t) r1;  zero-extend from 8 bits
ZEXTH   r1,rd           ; rd = (uint16_t) r1; zero-extend from 16 bits
MASKHI  r1,rd           ; rd = ~(-1) >> r1;   logic shift right
```

7.3.2.2 立即寄存器

```
ADDI    r1,imm32,rd       ; rd = r1 + imm32;
RSUBI   r1,imm32,rd       ; rd = imm32 - r1;
ORI     r1,imm32,rd       ; rd = r1 | imm32;
NORI    r1,imm32,rd       ; rd = ~(r1 | imm32);
ANDI    r1,imm32,rd       ; rd = r1 & imm32;
NANDI   r1,imm32,rd       ; rd = ~(r1 & imm32);
XORI    r1,imm32,rd       ; rd = r1 ^ imm32;
XNORI   r1,imm32,rd       ; rd = ~(r1 ^ imm32);
SRAI    r1,uimm5,rd       ; rd = r1 >> uimm5;     signed
SRLI    r1,uimm5,rd       ; rd = r1 >> uimm5;     logic
SLLI    r1,uimm5,rd       ; rd = r1 << uimm5;
MULI    r1,imm32,rd       ; rd = r1 * imm32;
MACI    r1,imm32,rd       ; rd += r1 * imm32;
SMODI   r1,imm32,rd       ; rd = r1 % imm32;      signed
SDIVI   r1,imm32,rd       ; rd = r1 / imm32;      signed
UMODI   r1,imm32,rd       ; rd = r1 % imm32;      unsigned
UDIVI   r1,imm32,rd       ; rd = r1 / imm32;      unsigned
EQI     r1,imm32,rd       ; rd = r1 == imm32;
NEI     r1,imm32,rd       ; rd = r1 != imm32;
LTI     r1,imm32,rd       ; rd = r1 < imm32;      signed
GTI     r1,imm32,rd       ; rd = r1 > imm32;      signed
GEI     r1,imm32,rd       ; rd = r1 >= imm32;     signed
LEI     r1,imm32,rd       ; rd = r1 <= imm32;     signed
LTUI    r1,imm32,rd       ; rd = r1 < imm32;      unsigned
GTUI    r1,imm32,rd       ; rd = r1 > imm32;      unsigned
GEUI    r1,imm32,rd       ; rd = r1 >= imm32;     unsigned
LEUI    r1,imm32,rd       ; rd = r1 <= imm32;     unsigned
ADDMXI  r1,imm32,rd       ; rd = (r1 + imm32) % 0x7fffffff;
MOVC    simm16,rd         ; rd = simm16;
MOVI    imm32,rd          ; rd = imm32;
```

```
MOVF    caddr,rd          ; rd = caddr;           load code reference
CLR     rd                ; rd = 0;
INC     rd                ; rd = rd + 1;
DEC     rd                ; rd = rd - 1;
```

C99定义后面是符号数除法和余数操作：除法将数学结果降至趋于0；余数尊重序关系：

$$\frac{a}{b} \times b + a\%b = a$$

其中%表示余项函数，或系数。

移位指令的正确操作对象的范围是[0, 31]，否则行为是未定义的。带符号的右移将原始最重要的位复制到空白位置处。在算术上，这对应的是除以2的幂数，将数学结果四舍五入到负无穷大（取整四舍五入）。

7.3.3 简易格式

出现三种操作对象指令许多时候使用其中一个操作对象作为结果。既然可以提供更为简洁的代码，则可为它们提供特殊的操作码：

```
ADD2    r1,rd                       ; rd += r1;
SUB2    r1,rd                       ; rd -= r1;
MUL2    r1,rd                       ; rd *= r1;
AND2    r1,rd                       ; rd &= r1;
OR2     r1,rd                       ; rd |= r1;
XOR2    r1,rd                       ; rd ^= r1;
XNOR2   r1,rd                       ; rd = ~(rd ^ r1);
NE2     r1,rd                       ; rd = r1 != rd;
EQ2     r1,rd                       ; rd = r1 == rd;
SLL2    r1,rd                       ; rd <<= r1;
SRA2    r1,rd                       ; rd >>= r1;          signed
SRL2    r1,rd                       ; rd >>= r1;          logical
```

按位的立即操作对一位进行测试或更改。考虑数位位置，立即操作可利用5位来进行编码。

```
ANDB   r1,uimm5,rd                  ; rd = r1 & (1 << uimm5);
ORB    r1,uimm5,rd                  ; rd = r1 | (1 << uimm5);
XORB   r1,uimm5,rd                  ; rd = r1 ^ (1 << uimm5);
TESTB  r1,uimm5,rd                  ; rd = (r1 >> uimm5) & 1;
TESTBC r1,uimm5,rd                  ; rd = ! ((r1 >> uimm5) & 1);
```

许多都是跟0对比。这可节省立即操作数，且模仿起来更容易。

```
EQZ     r1,rd                       ; rd = r1 == 0;
NEZ     r1,rd                       ; rd = r1 != 0;
LTZ     r1,rd                       ; rd = r1 < 0;
GTZ     r1,rd                       ; rd = r1 > 0;
LEZ     r1,rd                       ; rd = r1 <= 0;
GEZ     r1,rd                       ; rd = r1 >= 0;
```

它们的未签名版本无任何意义。或真或假或使用EQZ或NEZ来表达。

7.3.4 控制流

7.3.4.1 通用规则

带有直接操作对象的控制流指令对与指令的结束地址相关的目标进行编码。在以寄存器为基础的控制流中，寄存器持有一个函数指针索引。

7.3.4.2 无条件分支与功能调用

```
JMP     pcr24             ; goto PC+pcr24;
JMPR    rd                ; goto rd (shall be code reference);
CALL    pcr24             ; push PC; goto PC+pcr24;
CALLR   rd                ; push program counter; goto rd (code reference);
ENTER   uimm16            ; shift register file by 16 (new r0 is old r16);
                          ; r16 = r0 - 4 * uimm16;
ENTER0                    ; equivalent to ENTER 0
ENTERC  uimms10           ; equivalent to ENTER uimms10
LEAVE                     ; unshift register file
RETURN                    ; unshift register file;
                          ; goto popped program counter;
RETURNL                   ; goto popped program counter;

SWITCH  r1,uimm16           ; goto PC + MIN(r1, uimm16)
                          ; advance to r1th CASE statement below
CASE    pcr24               ; goto PC + pcr24
                          ; add a case in the previous SWITCH. The first
                            ; entry is case value zero, each next one adds
                            ; one to the case value.
```

7.3.4.3 条件分支

```
JEQ     r1,r2,pcr16           ; if (r1 == r2) goto PC+pcr16;
JNE     r1,r2,pcr16           ; if (r1 != r2) goto PC+pcr16;
JLT     r1,r2,pcr16           ; if (r1 < r2) goto PC+pcr16;
JGE     r1,r2,pcr16           ; if (r1 >= r2) goto PC+pcr16;
JLTU    r1,r2,pcr16           ; if ((unsigned)r1 < (unsigned)r2) goto PC+pcr16;
JGEU    r1,r2,pcr16           ; if ((unsigned)r1 >= (unsigned)r2) goto PC+pcr16;

JEQC    r1,simm11,pcr16       ; if (r1 == simm11) goto PC+pcr16;
JNEC    r1,simm11,pcr16       ; if (r1 != simm11) goto PC+pcr16;
JLTC    r1,simm11,pcr16       ; if (r1 < simm11) goto PC+pcr16;
JGEC    r1,simm11,pcr16       ; if (r1 >= simm11) goto PC+pcr16;
JLTUC   r1,uimm11,pcr16       ; if ((unsigned) r1 < uimm11) goto PC+pcr16;
JGEUC   r1,uimm11,pcr16       ; if ((unsigned) r1 >= uimm11) goto PC+pcr16;
JGTC    r1,simm11,pcr16       ; if (r1 > simm11) goto PC+pcr16;
JLEC    r1,simm11,pcr16       ; if (r1 <= simm11) goto PC+pcr16;
JGTUC   r1,uimm11,pcr16       ; if ((unsigned) r1 > uimm11) goto PC+pcr16;
JLEUC   r1,uimm11,pcr16       ; if ((unsigned) r1 <= uimm11) goto PC+pcr16;
```

7.3.4.4 基于与常量的内存比较的条件分支

```
JWEQC   r1,simm11,pcr16       ; if (MEM4(r1) == simm11) goto PC+pcr16;
JWNEC   r1,simm11,pcr16       ; if (MEM4(r1) != simm11) goto PC+pcr16;
```

这些从内存读取字，并与常量作比较。

7.3.4.5 目前的条件分支

对于以上所描述的每个条件分支，有个目前版本，为24位位移。汇编器应选择适合的最短版本。

7.3.5 加载与存储指令

7.3.5.1 寄存器 + 位移

```
LDSBI    r1,imm32,rd                  ; rd = SMEM1(r1 + imm32);
LDUBI    r1,imm32,rd                  ; rd = UMEM1(r1 + imm32);
LDSHI    r1,imm32,rd                  ; rd = SMEM2(r1 + imm32);
LDUHI    r1,imm32,rd                  ; rd = UMEM2(r1 + imm32);
LDWI     r1,imm32,rd                  ; rd = MEM4 (r1 + imm32);

STBI     rd,r1,imm32                  ; MEM1(r1 + imm32) = low8(rd);
STHI     rd,r1,imm32                  ; MEM2(r1 + imm32) = low16(rd);
STWI     rd,r1,imm32                  ; MEM4(r1 + imm32) = rd;
```

7.3.5.2 寄存器 + 短位移

```
LDSBC    r1,uimm8,rd                  ; rd = SMEM1(r1 + uimm8);
LDUBC    r1,uimm8,rd                  ; rd = UMEM1(r1 + uimm8);
LDSHC    r1,uimms9,rd                 ; rd = SMEM2(r1 + uimms9);
LDUHC    r1,uimms9,rd                 ; rd = UMEM2(r1 + uimms9);
LDWC     r1,uimms10,rd                ; rd = MEM4 (r1 + uimms10);

STBC     rd,r1,uimm8                  ; MEM1(r1 + uimm8) = low8(rd);
```

```
STHC     rd,r1,uimms9                    ; MEM2(r1 + uimms9) = low16(rd);
STWC     rd,r1,uimms10                   ; MEM4(r1 + uimms10) = rd;
```

7.3.5.3 寄存器索引

```
LDUB     r1,r2,rd                        ; rd = UMEM1(r1 + r2);
LDSB     r1,r2,rd                        ; rd = SMEM1(r1 + r2);
LDUH     r1,r2,rd                        ; rd = UMEM2(r1 + 2 * r2);
LDSH     r1,r2,rd                        ; rd = SMEM2(r1 + 2 * r2);
LDW      r1,r2,rd                        ; rd = MEM4(r1 + 4 * r2);

STB      rd,r1,r2                        ; MEM1(r1 + r2) = rd;
STH      rd,r1,r2                        ; MEM2(r1 + 2 * r2) = rd;
STW      rd,r1,r2                        ; MEM4(r1 + 4 * r2) = rd;

LDW1     r1,r2,rd                            ; rd = MEM4(r1 + r2);
STW1     rd,r1,r2                            ; MEM4(r1 + r2) = rd;
```

7.3.5.4 绝对索引

```
LDSHAX   imm32,r1,rd                         ; rd = SMEM2(imm32 + 2 * r1);
LDUHAX   imm32,r1,rd                         ; rd = UMEM2(imm32 + 2 * r1);
LDWAX    imm32,r1,rd                         ; rd = MEM4(imm32 + 4 * r1);
STHAX    rd,imm32,r1                         ; MEM2(imm32 + 2 * r1)= rd;
STWAX    rd,imm32,r1                         ; MEM4(imm32 + 4 * r1)= rd;
```

应注意，并不需要绝对索引的字节负载。比如，LDSBAX等于LDSBI。

7.3.5.5 专用堆栈访问

这些是隐性使用帧指针的字负载和储存。

```
LDFP     simm16,r1                           ; r1 = MEM4(FP + simm16);
STFP     r1,simm16                       ; MEM4(FP + simm16) = r1;
```

7.3.5.6 转储

用于编辑器生成的块复制的块复制指令。

```
COPY     r1,s:uimm32,r2,o:uimm32      ; copy s bytes from r1 to r2+o
```

7.3.6 复杂指令

这些指令执行一系列操作，通常均带有立即操作数。在本总结中，每个指定为i1、i2等的操作数均是32位的立即操作数（imm32）。

```
ADDANDI2   r1,i1,i2,rd                   ; rd = (r1 + i1) & i2;
ADDMULI2   r1,i1,i2,rd                   ; rd = (r1 + i1) * i2;
ADDORI2    r1,i1,i2,rd                       ; rd = (r1 + i1) | i2;
ADDXORI2   r1,i1,i2,rd                   ; rd = (r1 + i1) ^ i2;

MULADDI2   r1,i1,i2,rd                   ; rd = (r1 * i1) + i2;
MULANDI2   r1,i1,i2,rd                   ; rd = (r1 * i1) & i2;
MULORI2    r1,i1,i2,rd                       ; rd = (r1 * i1) | i2;
MULXORI2   r1,i1,i2rd                    ; rd = (r1 * i1) ^ i2;

RSUBANDI2  r1,i1,i2,rd                   ; rd = (i1 - r1) & i2;
RSUBORI2   r1,i1,i2,rd                   ; rd = (i1 - r1) | i2;
RSUBXORI2  r1,i1,i2,rd                   ; rd = (i1 - r1) ^ i2;

ORADDI2    r1,i1,i2,rd                       ; rd = (r1 | i1) + i2;
ORMULI2    r1,i1,i2,rd                       ; rd = (r1 | i1) * i2;

SLLADDI2   r1,s1:uimm5,i2,rd             ; rd = (r1 << s1) + i2;
SLLANDI2   r1,s1:uimm5,i2,rd             ; rd = (r1 << s1) & i2;
SLLORI2    r1,s1:uimm5,i2,rd                 ; rd = (r1 << s1) | i2;
SLLRSUBI2  r1,s1:uimm5,i2,rd             ; rd = i2 - (r1 << s1);

ANDSLLI2   r1,i1,s2:uimm5,rd             ; rd = (r1 & i1) << s2;

MAMI3      r1,i1,i2,i3,rd                ; rd = ((r1 * i1) & i2) * i3;
MPMI3      r1,i1,i2,i3,rd                ; rd = ((r1 * i1) + i2) * i3;
MOMI3      r1,i1,i2,i3,rd                ; rd = ((r1 * i1) | i2) * i3;

MPAI3      r1,i1,i2,i3,rd                ; rd = ((r1 * i1) + i2) & i3;
MPOI3      r1,i1,i2,i3,rd                ; rd = ((r1 * i1) + i2) | i3;
RORI3      r1,i1,i2,i3,rd                ; rd = i3 - ((i1 - r1) | i2);

AMPI3      r1,i1,i2,i3,rd                ; rd = ((r1 & i1) * i2) + i3;
```

```
LPAI3      r1,s1:uimm5,i2,i3,rd            ; rd = ((r1 << s1) + i2) & i3;

MPMPI4 r1,i1,i2,i3,i4,rd                   ; rd = (((r1 * i1) + i2) * i3) + i4;
MPOMI4 r1,i1,i2,i3,i4,rd                   ; rd = (((r1 * i1) + i2) | i3) * i4;
```

7.3.7 其他

7.3.7.1 系统调用

系统调用可实施多种业务。

```
SYSCALL  uimm16                             ; system service uimm16
```

实施便携式操纵系统接口（POSIX）系统调用的最小集是直接映射到基础操作系统。可添加更多的针对应用的业务。

7.3.7.2 伪指令

一些操作可通过另一些操作来表达。以下的伪操作码可供使用：

```
SUBI    r1,imm32,rd    = ADDI  r1,-imm32,rd
GT      r1,r2,rd       = LT    r2,r1,rd
LE      r1,r2,rd       = GE    r2,r1,rd
GTU     r1,r2,rd       = LTU   r2,r1,rd
LEU     r1,r2,rd       = GEU   r2,r1,rd
```

8 ECI客户端和ECI主机间的交互

8.1 一般性原则

在执行SYSCALL指令时会出现系统调用。指令包括可识别系统调用的立即操作数。系统调用是经过第7.2.8条中所描述的参数来对标准程序库进行的有效调用。

在寄存器R1..R8中推送的是前7个参数（字或指针）。若实际参数类型是8位或16位标量，则这些参数都是已签并扩展至32位的值。返回值（字或指针）应在R1里。

除非另有说明，所有存储地址都是指虚拟机的内存空间。

出于未来兼容性的原因，**ECI客户端**须清理所有寄存器R1..R8至零，而不是用于传递参数。所有寄存器的内容可通过库函数进行丢弃。

以下列出所有符合的实施须支持的强制性库系统调用。所用格式提供了：

– 库函数的描述。
– 库函数的描述。
– C语义中的声明。
– 参数和返回值得描述。
– 任何附加的注释。

使用以下惯例键入参数和返回值：

– **uint*nn***代表*nn*位的无符号整数（*nn*是8、16或32中的一个）。少于32位的值在放入寄存器中时须是0且可扩展至32位。
– **int*nn***代表带符号整数。少于32位的值放到寄存器中时须是带有符号且可扩展的。
– **void ***代表的是通用指针。

– **[u]int*nn* *** 代表[u]int*nn*类型一个值或一个阵列的指针。

– **struct struct_type *** 指的是内存中一个结构（或结构阵列）的指针 – 结构总是利用本惯例按引用传递。

8.2 错误值

大部分SYSCALL返回的是一个否定字，表明发现了一种错误条件。表1列出了错误值

表1 – 错误值

值	符号名称	含义
−49	EPERM	调用不存在的SYSCALL 或CLIB函数。
−50	EINVAL	其中一个参数不正确
−51	ERRSYSCALLMSGQUEUE	**ECI主机**超过其缓冲容量而发送的信息数量。
−52	ERRHEAPSIZE	要求堆大小的不恰当值
−53	ERRSTACKSIZE	要求栈大小的不恰当值

8.3 SYS_EXIT

SYSCALL ID：0x0001

描述：终止虚拟机，提供理由码。

声明：空的 SYS_EXIT（uint32理由）。

操作对象：终止的理由。

返回值：无。

注 – **理由**从表2列出了的值

表2 – SYS_EXIT理由值

理由	意思
0	正常终止
0x00000001..0x7FFFFFFF	错误条件，具体针对**ECI客户端**提供商
0x80000000..0xFFFFFFFF	预留共未来使用

8.4 SYS_PUTMSG

SYSCALL ID：0x0003

描述：发送一个异步消息（请求或响应）。

声明：int32 SYS_**PUTMSG**（MessageBuffer *msg_buffer）。

MessageBuffer的格式在[ITU-T J.1012]中定义。

操作对象：**msg_buffer**是指向信息缓冲块的指针。

返回值：**ECI主机**所分配信息的（非负16 位值）或以下任何的错误值（负）：**ERRSYSCALLMSGQUEUE**（表1）。

注 – 在正常的**ECI主机**操作条件下，本呼叫被视为未阻止。MessageBuffer的格式在[ITU-T J.1012]中定义。**ECI主机**复制msg_buffer内容，可被**ECI客户端**在 SYSCALL返回后立即再次使用。

8.5 SYS_GETMSG

SYSCALL ID：0x0004

描述：从**ECI主机**检索下一条信息（无论是请求还是结果）。若无信息提供，则SYSCALL将阻止。

声明：（**MessageBuffer** *） SYS_GETMSG()

MessageBuffer的格式定义在[ITU-T J.1012]。

操作对象：无

返回：包含**ECI主机**的下一条信息或表1中列出的任何错误值的缓冲指针。

注 – 若**ECI 主机**没有针对**ECI客户端**的信息排队，则调用将阻止。**ECI主机**无法更改信息缓冲内容，直到下一个SYS_GETMSG SYSCALL。在下一个SYS_GETMSG 调用后欲获得信息数据的**ECI客户端**需复制本数据。

8.6 SYS_HEAPSIZE

SYSCALL ID：0x0100

描述：向**ECI主机**发出的更改堆大小至所提供的参数的请求。

声明：int32 SYS_**HEAPSIZE**（uint32堆大小）

操作对象：**堆大小**：设置**ECI客户端**堆的大小。须是非负值，4的倍数，且不会导致堆栈段中堆的溢出。

返回：内存位置从DATA_BASE_ADDRESS按字节的位移，是可寻址内存中最低的非堆内存地址，或以下任何的错误值（负）：

ERRHEAPSIZE（表1）。

注 – 若**ECI主机**没有针对**ECI客户端**的信息排队，则调用将阻止。**ECI客户端**初始化SYS_HEAPSIZE（0）将返回离堆段开头的位移（当时是0大小）。

8.7 SYS_STACKSIZE

SYSCALL ID：0x0200

描述：向**ECI主机**发出的更改栈大小至所提供的参数的请求。

声明：int32 SYS_**STACKSIZE**（uint32 栈大小）

操作对象：**栈大小**：设置**ECI客户端**堆的大小。须是非负值，是4的倍数，且不会导致堆段中的栈的溢出。

返回：内存**位置**从DATA_BASE_ADDRESS按字节的位移，是可寻址内存中最低的栈内存地址，或以下任何的错误值（负）：**ERRSTACKSIZE**（表1）。

注 – 若**ECI主机**没有针对**ECI客户端**的信息排队，则调用将阻止。

8.8 SYS_SYNCCALL

SYSCALL ID：0x1000

描述：**ECI客户端**向**ECI主机**发送同步信息，并暂停执行直到系统调用返回。

声明：int32 SYS_SYNCCALL（uint32 tag, p1, p2, p3, …, pn）

操作对象：**tag**：与MessageBuffer结构的MsgTag字段相同的定义。MsgFlags字段设置为0，**ECI主机**应忽略。**p1…pn**：同步调用的参数。对于其结果大于32位实体的获取消息，**p1** i是返回结果的内存位置的起始地址，而**p2..pn**是设置消息的参数。所有常规参数包括structs，阵列按引用传递。

返回：对于返回符合32位的结果的收信，可返回结果。否则无返回结果。所有错误均忽略；错误的参数配置仅仅不产生结果和/或无影响。调用消息返回的是其具体语义所定义的状态代码。结果返回到具体消息语义所定义的SYSCALL指针参数的位置。

注 – 该SYSCALL将不阻止。

8.9 SYS_CLIB

SYSCALL ID：0x0300

描述：该SYSCALL充当应用编程接口（API），以使**ECI客户端**能使用标准的C库函数。所支持的函数集详见附件C。

声明：SYS_CLIB（uint32 clibfunc等等）

操作对象：正如附件C中所描述的，**clibfunc**确认的是被调用的C 库函数。所有其他操作对象均在C函数调用列表中有所定义。

返回：返回值详见附件C的库，或者表1所列出的任一错误值。

注 – 不同的C库函数使用不同的参数数量和类型，在本建议书中并未明确描述。附件详述了所有操作对象的格式。所有SYS_CLIB的操作对象均是标量值或虚拟机内存中非标量值的指针。由于一些C库函数可采用非标量参数，则虚拟机应在将执行传递给库之前，将按应用传递的参数转化为按值传递的参数。

9 字节码的生命周期

9.1 简介

虚拟机作为**ECI主机**的一部分固件来进行实施。当**ECI客户端**需要执行时，它通过**ECI主机**操作系统来进行动态加载/执行。若要求不同**ECI客户端**同时可用，则可为其提供多个实例。

如上所描述的，**ECI客户端**写在虚拟机的指令集中。由CP系统厂商（CA提供商或DRM运营商）筹备并作为供**ECI主机**使用的逻辑代码图像。本地进行转换以适应ECI主机及其操作环境的具体设计，并在需要时加载到虚拟机中。这在虚拟机内执行，直到故意终止（或出现错误情况），然后暂停**ECI客户端**的执行，虚拟机终止。

9.2 将新的ECI客户端装载到虚拟机

虚拟机充当外部所提供**ECI客户端**的中间主机，就像**ECI客户端**是在**ECI主机**设备上执行的本机应用一样。唯一的不同时**ECI客户端**由**ECI主机**设备的操作系统安装至虚拟机中，而不是本机应用。

为了装载**ECI客户端**，虚拟机的分系统首先创建一个虚拟处理器语境。加载需要分配虚拟机的内存和安装代码和数据段内容到内存，就如同是本机应用一样，但可执行和可连接的格式（ELF）[ETSI GS ECI 001-4]文档中所提供的代码和初始化数据（详见附件D）转到分配给虚拟机的内存。

由于程序中无法获得代码段，执行时可选择在加载时对代码进行某种预加工（如优化）。事实上，本建议书只描述图像中程序的格式。内部表征完全是具体针对实施的。唯一的条件是所有代码引用对于带有同样语义的程序而言均无法使用。

或者，**ECI客户端**图像在**ECI主机**设备先进行检索，然后存储起来随时等待按需加载时可进行预处理。这是若定期出现卸载和重新加载的情况保留和建立**ECI客户端**更为高效的方式。

9.3 VM初始化

需创建虚拟机的CPU通用语境 – 是寄存器文档、控制栈、数据和堆栈区域和程序计数器（PC），加上任意控制/状态逻辑和标记。本建议书中对此没有详述，因为它们取决于具体实施。

建立寄存器文档，以使R0位于寄存器文档空间的开头。所有寄存器设为0。因此，首个窗口中的帧指针和堆栈指针的寄存器（分别为R0和R16）设置为当在堆栈上推送首个字时，栈指针算前减量至-4（0xFFFFFFFC），字存储到此。

程序计数器的初始化至代码段的开头，除非**ECI客户端**图像文件中的ELF标题[ETSI GS ECI 001-4]的*e_entry*构件存在非零值，这时程序计数器设置为*e_entry*构件所规定的值（虚拟地址）。然后控制传递给虚拟机的执行组件，称之为“中央运行循环”。

9.4 中央运行循环

虚拟机的本质是中央运行循环，即将每一个顺序指令读入和译入寄存器上、虚拟机的内存和/或数据库调用中相应的一系列行动当中。循环执行指令直到出现异常情况。程序终止可是正常执行实践的一部分，比如，若程序执行SYS_EXIT系统调用，或作为错误情况的结果显现，比如控制栈上溢了。

若终止了“中央运行循环”，则虚拟机关闭，若在未来某个时间需要**ECI客户端**，则需要对其重新实例化。

附件A

虚拟机系统资源

（本附件构成本建议书的组成部分）

以下参数在本建议书中用于定义虚拟机（VM）的性能。在[b-ITU-T J Suppl. 7]中可找到参数的推荐值。

– REGISTER_FILE_SIZE
– CONTROL_STACK_SIZE = REGISTER_FILE_SIZE/16
– DATA_BASE_ADDRESS
– ADDRESSABLE_DATA_SIZE
– VM_RESERVED_SIZE
– CODE_SIZE
– DEFAULT_STACK_SIZE
– MIN_RAM

附件B

虚拟机的操作码

（本附件构成本建议书的组成部分）

以下的编码描述的是指令如何在二进制图像中进行编码。以下综述显示的是不同的格式。总结行列出了组成指令的字段，用逗号隔开。字段或为显性码，或为其后是字段宽度指示器的字段名称。同一格式不同的操作对象具有占据'op（操作）'字段的对应模式。位数是按大端法列出的。

比如，SUB R3、R5、R17的代码为：

```
1011 00001 00011 00101 10001
```

或按半字节：

```
1011 0000 1000 1100 1011 0001
```

或按字节：

```
0xB0, 0x8C, 0xB1
```

每个指令名后面有一个数字，是所定义的该指令的操作码数量。该数量对于所有未来版本的虚拟机指令集而言应是相同的。

操作码字段不能超过4个字节。因此，32位字段在字节界开始。字段均不超过32位。

```
0, op:5, r1:5, rd:5
    00000       MOV    16
    00001       ADD2   17
    00010       SUB2   18
    00011       MUL2   19
    00100       AND2   20
    00101       OR2    21
    00110       XOR2   22
    00111       SLL2   23
    01000       SRL2   24
    01001       SRA2   25
    01010       NE2    26
    01011       EQ2    27
    01100       NEZ    28
    01101       EQZ    29
    01110       LTZ    30
    01111       GEZ    31
    10000       GTZ    32
    10001       LEZ    33
    10010       EXTB   34
    10011       EXTH   35
    10100       ZEXTB  36
    10101       ZEXTH  37
    10110       ABS    38
    10111       NEG    39
    11000       NOT    40
    11001       XNOR2  41
    11010       MASKHI 42

100, op:3, r1:5, rd:5, imm:32
    000         ADDI   136
    001         RSUBI  137
    010         ANDI   138
    011         ORI    139
    100         XORI   140
    101         MULI   141
    110         MACI   142
    111         ADDMXI 143

101000, op:2
    00          ENTER0 0
    01          RETURN 1
    10          RETURNL    2
```

```
    11          LEAVE  3

10100100, op:3, rd:5
    000         INC    8
    001         DEC    9
    010         JMPR   10
    011         CALLR  11
    100         CLR    12

1010010100000, op:4, r1:5, r2:5, rd:5
    0000        SDIV   80
    0001        SMOD   81
    0010        UDIV   82
    0011        UMOD   83
    0100        TESTBC 84

101001010001, op:4, r1:5, imm:11, pcr:24
    0000        JFNEC  560
    0001        JFEQC  561
    0010        JFLTC  562
    0011        JFGEC  563
    0100        JFGTC  564
    0101        JFLEC  565
    0110        JFLTUC 566
    0111        JFGEUC 567
    1000        JFLEUC 568
    1001        JFGTUC 569
    1010        JFWNEC 570
    1011        JFWEQC 571

10101000, op:3, r1:5, imm:16
    000         STFP   96
    001         LDFP   97
    010         MOVC   98
    011         SWITCH 99

1011, op:5, r1:5, r2:5, rd:5
    00000       ADD    48
    00001       SUB    49
    00010       MUL    50
    00011       AND    51
    00100       OR 52
    00101       XOR    53
    00110       SLL    54
    00111       SRA    55
    01000       SRL    56
    01001       SLLI   57
    01010       SRAI   58
    01011       SRLI   59
    01100       NE     60
    01101       EQ     61
    01110       LT     62
    01111       GE     63
    10000       LTU    64
    10001       GEU    65
    10010       ANDB   66
    10011       ORB    67
    10100       XORB   68
    10101       LDSB   69
    10110       LDUB   70
    10111       LDSH   71
    11000       LDUH   72
    11001       LDW    73
    11010       LDW1   74
    11011       STB    75
    11100       STH    76
    11101       STW    77
    11110       STW1   78
    11111       TESTB  79

110000, op:2, imm:24
    00          JMP    104
    01          CALL   105
    10          CASE   106

110001000, op:2, rd:5, imm:32
    00          MOVI   108
    01          MOVF   109

110001001, op:5, r1:5, rd:5, imm:32
```

```
    00000       NANDI  144
    00001       NORI   145
    00010       XNORI  146
    00011       NEI    147
    00100       EQI    148
    00101       LTI    149
    00110       GEI    150
    00111       GTI    151
    01000       LEI    152
    01001       LTUI   153
    01010       GEUI   154
    01011       GTUI   155
    01100       LEUI   156
    01101       SMODI  157
    01110       SDIVI  158
    01111       UMODI  159
    10000       UDIVI  160
    10001       STBI   161
    10010       STHI   162
    10011       STWI   163
    10100       LDSBI  164
    10101       LDUBI  165
    10110       LDSHI  166
    10111       LDUHI  167
    11000       LDWI   168
    11001       LDSHAX 169
    11010       LDUHAX 170
    11011       LDWAX  171
    11100       STHAX  172
    11101       STWAX  173

11001000000, op:3, r1:5, rd:5, imm:24
    000         JFNE   576
    001         JFEQ   577
    010         JFLT   578
    011         JFGE   579
    100         JFLTU  580
    101         JFGEU  581

11001000110, op:3, r1:5, r2:5, imm:16
    000         JNE    120
    001         JEQ    121
    010         JLT    122
    011         JGE    123
    100         JLTU   124
    101         JGEU   125

11001000111000, r1:5, r2:5, s:32, o:32
                COPY   112

11001001000, op:3, r1:5, r2:5, imm:8
    000         STBC   128
    001         STHC   129
    010         STWC   130
    011         LDSBC  131
    100         LDUBC  132
    101         LDSHC  133
    110         LDUHC  134
    111         LDWC   135

11001100000000000, op:5, r1:5, rd:5, imm1:32, imm2:32
    00000       ADDANDI2   200
    00001       ADDMULI2   201
    00010       ADDORI2    202
    00011       ADDXORI2   203
    00100       MULADDI2   204
    00101       MULANDI2   205
    00110       MULORI2    206
    00111       MULXORI2   207
    01000       RSUBANDI2  208
    01001       RSUBORI2   209
    01010       RSUBXORI2  210
    01011       ORADDI2    211
    01100       ORMULI2    212

11001100000000010000, op:5, r1:5, imm1:5, rd:5, imm2:32
    00000       SLLADDI2   232
    00001       SLLANDI2   233
    00010       SLLORI2    234
    00011       SLLRSUBI2  235
```

```
    00100       ANDSLLI2   236

1100110000000010001, op:5, r1:5, imm1:5, rd:5, imm2:32, imm3:32
    00000       LPAI3  392

110011000000001, op:7, r1:5, rd:5, imm1:32, imm2:32, imm3:32
    0000000     MAMI3  264
    0000001     MPMI3  265
    0000010     MOMI3  266
    0000011     MPAI3  267
    0000100     MPOI3  268
    0000101     RORI3  269
    0000110     AMPI3  270

110011000000010, op:7, r1:5, rd:5, imm1:32, imm2:32, imm3:32, imm4:32
    0000000     MPMPI4 424
    0000001     MPOMI4 425

1101, op:4, r1:5, imm:11, imm:16
    0000         JNEC  1
84
    0001         JEQC  185
    0010         JLTC  186
    0011         JGEC  187
    0100         JGTC  188
    0101         JLEC  189
    0110         JLTUC 190
    0111         JGEUC 191
    1000         JLEUC 192
    1001         JGTUC 193
    1010         JWNEC 194
    1011         JWEQC 195

11100000, uimm:8 ENTERC    5

1110001, op:1, uimm:16
    0            ENTER     6
    1            SYSCALL   7
```

附件C

标准C库存程序

（本附件构成本建议书的组成部分）

C.1 简介

本附件描述的是供**ECI客户端**使用的一系列标准C99库例程[b-ISO IEC 9899]。有关每种功能，**ECI客户端**所传递的运算对象和返回值的详情做了定义。

注意“串”是指0字节所终止的一系列非0字节。

以下详述的功能均显示为标准的C库存调用。所有情况下，首个参数将进入R2（因R1将包括功能ID，clibfunc）。声明将假定所有传递的值为标量值或非标量值的指针。若库函数请求值传递的是非标量参数，则SYSCALL将按引用传递，虚拟机须转换库所要求的参数。

返回值总是标量值或R1中所返回的指针。

注 – clibfunc的所选值由以下部分组成：

((clibfunc >> 8) & 0x000000FF) =关于库函数的C标准一章的分章数量，代码为二进制编码十进制数（对于C99，为第7章，而<string.h>库在第21分章）

((clibfunc >> 4) & 0x0000000F) =库里函数的类型 – 分章数量后面的数量。

(clibfunc & 0x0000000F) =库里某一特定种类的函数数量 – 函数种类数量后面的数量。

比如，memmove()在[b-ISO IEC 9899]中7.21.2.2标题下有所描述。因此，clibfunc编码为0x00002122。

C.2 memmove

Clibfunc: 0x00002122

描述：从s2所指的内存复制n字节到s1的内存指针里。内存可重叠。

声明：uint8 * memmove（uint8 * s1, uint8 * s2, uint32 n）

返回值：s1

C.3 strcpy

clibfunc: 0x00002123

描述：复制s2所指的串到s1所指的内存（包括终结字符）。若内存区域重叠，则结果是未定义的。

声明： uint8 * strcpy（uint8 * s1, uint8 * s2）

返回值：s1

C.4 strncpy

clibfunc: 0x00002124

描述：关于strcpy()，最多复制n字节。若s2的长度大于n，则将附加无效字节（在s1[n]）。

声明：uint8 * strncpy（uint8 * s1, uint8 * s2, uint32 n）

返回值：s1

C.5 strcat

clibfunc: 0x00002131

描述：附加s2所指串的副本（包括终结字符）到s1所指的串的结尾。若内存区域重叠，结果是未定义的。

声明：uint8 * strcat（uint8 * s1, uint8 * s2）

返回值：s1

C.6 strncat

clibfunc: 0x00002132

描述：附加s2所指串的副本（包括终结字符）到s1所指的串的结尾。但最多复制n字节。若s2的长度大于n，则将附加一个无效的字节（在n+1字节，在原始s1最后一个非无效字节的后面）。若内存区域重叠，则未定义结果。

声明： uint8 * strncat（uint8 * s1, uint8 * s2, uint32 n）

返回值：s1

C.7 memcmp

clibfunc: 0x00002141

描述：比较s1所指的前n个字节和s2所指的前n个字节。

声明：uint32 memcmp（uint8 *s1, uint8 *s2, uint32 n）

返回值：R1=0 若匹配，否则 R1取决于从左数第一个其值不匹配的位置。

R1>0 若s1在该位置的字节大于s2的字节。

R1<0 若s1在该位置的字节大于s2的字节。

C.8 strcmp

clibfunc: 0x00002142

描述：比较s1和s2所指的串。

声明：uint32 strcmp（uint8 * s1, uint8 * s2）

返回值：R1==0 若匹配，否则R1取决于从左算起第一个其值不匹配的位置。

R1>0 若s1在该位置的字节大于s2的字节。

R1<0 若s1在该位置的字节大于s2的字节。

C.9 strncmp

clibfunc: 0x00002144

描述：比较s1和s2所指的串，但最多至n字节。

声明：uint32 strncmp（uint8 * s1, uint8 * s2, uint32 n）

返回值：R1==0 若匹配，否则R1取决于从左算起第一个其值不匹配的位置。

R1>0 若s1在该位置的字节大于s2的字节。

R1<0 若s1在该位置的字节大于s2的字节。

C.10 memchr

clibfunc: 0x00002151

描述：找到s所指的n字节内c里首次出现字节的位置。

声明：uint8 * memchr（uint8 *s, uint8 c, uint32 n）

返回值：所定位字节的指针，或未找到此字节时为0。

C.11 strchr

clibfunc: 0x00002152

描述：找到s所指的n字节内c里首次出现字节的位置。

声明：uint8 * memchr（uint8 *s, uint8 c, uint32 n）

返回值：所定位字节的指针，或未找到此字节时为0。

C.12 strcspn

clibfunc: 0x00002153

描述：计算s1所指串的最大起始段（整个包括不属于s2所指串的字节）的长度。

声明：uint32 strcspn（uint8 * s1, uint8 * s2）

返回值：计算的长度。

C.13 strpbrk

clibfunc: 0x00002154

描述：找到s2所指的串中任何字节的s1所指的串首次出现的位置。

声明： uint32 strpbrk（uint8 * s1, uint8 * s2）

返回值：满足条件的首个字节的位置，或未找到这样字节时为0。

C.14 strrchr

clibfunc: 0x00002155

描述：找到s所指串内c中字节最后出现的位置，达到和包括终止中（无效）字节。

声明：uint8 * strrchr（uint8 * s, uint8 c）

返回值：所定位字节的指针，或未找到此字节时为0。

C.15 strspn

clibfunc: 0x00002156

描述：计算s1所指串的最大起始段（整个包括属于s2所指串的字节）的长度。

声明：uint32 strspn（uint8 * s1, uint8 * s2）

返回值：计算的值。

C.16 strstr

clibfunc: 0x00002157

描述：找到s2所指的串中s1所指的串首次出现的位置（不包括终止中的字节）

声明：uint8 strstr（uint8 * s1, uint8 * s2）

返回值：设置位置的指针，或未找到为0。

C.17 memset

clibfunc: 0x00002161

描述：复制c最无效的字节到s所指的内存里n次。

声明：uint8 * memset（uint8 * s, uint8 c, uint32 n）

返回值：s

附件D

ECI客户端的文件格式

（本附件构成本建议书的组成部分）

ECI客户端的图像文档须符合ELF目标文档的格式规格[ETSI GS ECI 001-4]。本附件描述的是符合虚拟机规格所需的具体信息。由于虚拟机支持32位的架构和小端模式，因此*e_ident* [ETSI GS ECI 001-4] 中的ELF文件识别须使用表D.1中的值。

表D.1 – 符合ECI的*e_ident*设置

名称	值
e_ident[EI_CLASS]	ELFCLASS32
e_ident[EI_DATA]	ELFDATA2LSB

表D.2列出用于一些ELF标题下成员的值。

表D.2 – 针对ELF标题成员且符合ECI的设置

名称	值
e_type	ET_EXEC
e_machine	ET_NONE
e_version	EV_CURRENT

载入器应拒绝值与本附件中所给出的不同的任何**ECI客户端**图像文档。

附录 I

有待进一步发展的领域

（本附录不构成本建议书不可分割的部分）

已经确定，本建议书需要做进一步的开发和验证，才能满足[ITU-T J.1010]中规定的要求，并且[ITU-T J.1010]需要进行更新，以反映MovieLabs增强型内容保护（ECP）规范[b-ECP]的要求。[b-ITU-T J.1011]、[ITU-T J.1012]、ITU-T J.1013、[b-ITU-T J.1014]、[b-ITU-T J.1015]和 [b-ITU-T J.1015.1]建议书未来应予更新，以反映对[ITU-T J.1010]的那些更新。

国际电联的许多成员国以及各行各业的利益攸关方 – 包括设备和电子组件的制造商、受版权保护的内容的所有者和被许可者、机顶盒（OTT）和线性电视服务的提供商以及全球各地基于有条件访问系统（CAS）和数字版权管理（DRM）解决方案的提供商都对嵌入式通用接口（ECI）不能完全满足ECP要求以及更广泛的行业内容保护要求表示了担忧。

更具体地说，其对ITU-T第9研究组（SG9）会议（2020年4月16-23日）的文稿引起了人们的关注。来自以色列、澳大利亚、ITU-T部门成员三星公司以及SG9准成员Sky Group和MovieLabs的文稿提议在ECI建议书中纳入一系列更改，但未就此达成共识。这些项目在[b-SG9 Report 17 Ann.1]中进行了清点。

这些建议包括：

1） 通过缩小ECI范围来简化ECI系统；

2） 取消DRM；

3） 取消对内容的重加密；

4） 取消软件管理；

5） 增加用于安全存储和加密操作的API；

6） 允许供应商特定的密钥阶梯；

7） 采用ITU-T J.1207 TEE要求；

8） 包括VM的TEE实施方案；

9） 升级密码算法的强度，例如，使用SHA-384；

10） 使用标准证书，例如，ITU-T X.509；

11） 重新考虑客户端之间的通信；

12） 与ETSI进行其他联络；

13） 进行额外的同行评审；

14） 探索信托机构模型的替代方案；

15） 进一步定义ECI合规性和稳健性规则的技术方面问题；

16） 增加有关多样性的要求，例如，地址空间随机化；

17） 增加有关运行时完整性检查的要求。

这些建议反映出内容保护及其违背之可能带来的威胁正在不断演变。ECI最初是在批准本ITU-T建议书之前近十年来构思的。对像ECI这样的系统，需要定期根据攻击技术和行业保护要求的最新状况进行评估。

存在其他机制以实现互操作性。特别是对DRM用例，大多数互联网视频服务已经部署了其他解决方案，以提供互操作性并满足其需求。

由于许多成员国将国际电联标准视为对其市场和行业发展有影响力的指导来源，因此进一步的澄清很重要。关注清单确保ECI在其国内市场中的实施，这可涉及对本ITU T建议书含义的全面理解，并确保在考虑立法、法规或有关要求消费者数字电视设备可互操作的市场需求时能虑及这些问题。它还确保技术设备制造商（它们可能更喜欢使用一组独特的要求或其他标准来设计产品）在为不同市场开发产品时可以考虑到这些问题。

参考书目

[b-ITU-T J.1010] ITU-T J.1010建议书（2016年），用于可交换有条件访问/数字版权管理（CA/DRM）解决方案的嵌入式通用接口（ECI）；使用案例和要求。

[b-ITU-T J.1011] ITU-T J.1011建议书（2016年），用于可交换有条件访问/数字版权管理（CA/DRM）解决方案的嵌入式通用接口（ECI）；架构、定义和概述。

[b-ITU-T J.1014] ITU-T J.1014建议书（2020年），用于可交换有条件访问/数字版权管理（CA/DRM）解决方案的嵌入式通用接口（ECI） -ECI特定的功能。

[b-ITU-T J.1015] ITU-T J.1015建议书（2020年），用于可交换有条件访问/数字版权管理（CA/DRM）解决方案的嵌入式通用接口（ECI）；高级安全系统 – 密钥阶梯块。

[b-ITU-T J.1015.1] ITU-T J.1015.1建议书（2020年），用于可交换有条件访问/数字版权管理（CA/DRM）解决方案的嵌入式通用接口（ECI）；高级安全系统 – 密钥阶梯块：控制字使用规则信息和相关数据1的验证。

[b-ITU-T J-Suppl.7] ITU-T 系列J建议书 – 增补7（2020年），用于可交换有条件访问/数字版权管理（CA/DRM）解决方案的嵌入式通用接口（ECI）；ECI实施导则（EG）。

[b-SG9 Report 17 Ann.1] ITU-T SG9会议报告，SG9-R17-附件1（2020年），2020年4月16-23日召开的SG9全虚拟会议第17号报告附件1。
https://www.itu.int/md/T17-SG09-R-0017/en

[b-ECP] MovieLabs Specification for Enhanced Content Protection – Version 1.2
Available at: <https://movielabs.com/ngvideo/MovieLabs_ECP_Spec_v1.2.pdf>

[b-ETSI GS ECI 001-3] ETSI GS ECI 001-3 (2017),*Embedded Common Interface (ECI) for exchangeable CA/DRM solutions; Part 3: CA/DRM Container, Loader, Interfaces, Revocation.*

[b-ISO/IEC 9899] ISO/IEC 9899:2018*Information technology – Programminglanguages – C.*

[b-TIS-ELF] TIS Committee (1995), *Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification, version 1.2.* Available at <https://refspecs.linuxfoundation.org/elf/elf.pdf>.

ITU-T建议书系列

系列A	ITU-T工作的组织
系列D	资费及结算原则和国际电信/ICT的经济和政策问题
系列E	综合网络运行、电话业务、业务运行和人为因素
系列F	非话电信业务
系列G	传输系统和媒介、数字系统和网络
系列H	视听及多媒体系统
系列I	综合业务数字网
系列J	**有线网络和电视、声音节目及其他多媒体信号的传输**
系列K	干扰的防护
系列L	环境与ICT、气候变化、电子废物、节能；线缆和外部设备的其他组件的建设、安装和保护
系列M	电信管理，包括TMN和网络维护
系列N	维护：国际声音节目和电视传输电路
系列O	测量设备的技术规范
系列P	电话传输质量、电话设施及本地线路网络
系列Q	交换和信令，以及相关联的测量和测试
系列R	电报传输
系列S	电报业务终端设备
系列T	远程信息处理业务的终端设备
系列U	电报交换
系列V	电话网上的数据通信
系列X	数据网、开放系统通信和安全性
系列Y	全球信息基础设施、互联网协议问题、下一代网络、物联网和智慧城市
系列Z	用于电信系统的语言和一般软件问题

瑞士印刷
2020年，日内瓦