TCG Software Stack (TSS) Specification

Version 1.10 Golden
August 20, 2003

Copyright © 2003 Trusted Computing Group, Incorporated.

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Without limitation, TCG disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

No license, express or implied, by estoppel or otherwise, to any TCG or TCG member intellectual property rights is granted herein.

Except that a license is hereby granted by TCG to copy and reproduce this specification for internal use only.

Contact the Trusted Computing Group at http://trustedcomputinggroup.org for information on specification licensing through membership agreements.

Any marks and brands contained herein are the property of their respective owners.

Change History

Version	Date	Description
1.10	Aug 20, 2003	Initial release. No version prior to this exists.

Copyright TCG 2003

Table of Contents

TCG SOFTWARE STACK (TSS) SPECIFICATION 1

1. THE TCG SOFTWARE STACK (TSS) 11

1.1 General Introduction 11
1.2 Introduction to the TSS 11
1.3 TSS functions defined are not exclusive 12
1.4 Platform Architecture 13
1.4.1 Platform Modes 13
1.4.2 Procedure Calls 13
1.4.2.1 Local Procedure Call (LPC) 13
1.4.2.2 Remote Procedure Call (RPC) 14
1.5 Trust Boundaries 14
1.6 Roles 14
1.6.1 TPM Owner 14
1.6.2 TPM User 14
1.6.3 Platform Administrator 15
1.6.4 Platform User 15
1.6.5 Operator 15
1.6.6 Public 15
1.7 TSS Architecture 16
1.7.1 TCG Service Provider (TSP) 17
1.7.1.1 TSP Interface (TSPI) 18
1.7.1.2 TSP Context Manager (TSPCM) 18
1.7.1.3 TSP Cryptographic Functions (TSPCF) 18
1.7.2 TCG Core Services (TCS) 18
1.7.2.1 TCS Interface (Tcsi) 19
1.7.2.2 TCS Context Manager (TCSCM) 19
1.7.2.3 TCS Key & Credential Manager (TCSKCM) 19
1.7.2.4 TCS Event Manager (TCSEM) 19
1.7.2.5 TCS TPM Parameter Block Generator (TcsipBG) 20
1.7.3 TCG Device Driver Library (TDDL) 20
1.7.3.1 TDDL Interface (Tddli) 20
1.7.4 TPM Device Driver (TDD) 21
1.7.4.1 TDD Interface (TDDI) 21
1.8 Cryptographic Infrastructures 22

2. COMMON ENVIRONMENT 23

2.1 Naming Conventions 23
2.2 Abbreviations 23
2.3 Definitions 24
2.3.1 Data Types 24
2.3.2 Defined Constants 25
2.3.2.1 Object Type Definitions 25
2.3.2.2 Object Initialization Definitions 25
2.3.2.3 Attribute Definitions for a Context Object 26
2.3.2.4 Attribute Definitions for a TPM Object 26
2.3.2.5 Attribute Definitions for a Policy Object 26
2.3.2.6 Attribute Definitions for a Key Object 27
2.3.2.7 Attribute Definitions for a Data Object 28
2.3.2.8 Policy Definitions for Secret Mode 28
2.3.2.9 Policy Definition for Secret Lifetime 29
2.3.2.10 TPM Status Flags Definitions 29
2.3.2.11 Algorithm ID Definitions 31
2.3.2.12 Capability Flag Definitions 31
2.3.2.13 Sub-Capability Flag Definitions 32
2.3.2.14 Persistent Storage Flag Definitions 33

 Copyright TCG 2003

2.3.2.15 Migration Scheme Definitions 33
2.3.2.16 Key Usage Definitions 34
2.3.2.17 Key Size Definitions 34
2.3.2.18 Key Type Flags 35
2.3.2.19 Key Authorization 35
2.3.2.20 Key Encryption Scheme Definitions 35
2.3.2.21 Key Signature Scheme Definitions 36
2.3.2.22 Event Type Definitions 36
2.3.2.23 Well Known Secret 38
2.4 RETURN CODES 39
2.4.1 Return Codes Scheme 39
2.4.2 Common Return Code Defines 40
2.4.3 Common Return Code Rules 41
2.4.4 OS Specific Error Information: 41
2.4.4.1 Windows Operating System: 41
2.4.4.2 Linux 42
2.4.5 ErrorSample 43
2.5 STRUCTURES 45
2.5.1 TSS_VERSION 45
2.5.2 TSS_PCR_EVENT 46
2.5.3 TSS_EVENT_CERT 47
2.5.4 TSS_UUID 48
2.5.5 TSS_KM_KEYINFO 49
2.5.6 TSS_VALIDATION 50
2.6 PERSISTENT STORAGE 51
2.7 KEY MANAGEMENT 51
2.7.1 TSS Load Key Command Flow 53
2.7.2 TSS Load Key Flow Diagram 54
2.7.3 Key Handles 56
2.8 PORTABLE DATA 57

3. TCG SERVICE PROVIDER (TSP) 58

3.1 THEORY OF OPERATION 58
3.1.1 Functional Overview 58
3.1.2 Interface Design 58
3.1.2.1 Classes 58
3.1.2.2 Object Relationship 59
3.1.3 Authorization Data Handling 62
3.1.3.1 Secrets Handled by Service Provider 63
3.1.3.2 Secrets Handled by Application 63
3.1.4 Implementation Considerations 63
3.1.5 User Interface Elements 64
3.1.6 Runtime Considerations 64
3.2 TSPI-SPECIFIC RETURN CODE DEFINES 64
3.3 INTERFACE DESCRIPTION 66
3.3.1 Syntax 66
3.3.2 Calling Conventions regarding Memory Management 66
3.3.3 Classes and Methods 67
3.3.3.1 Common Methods Definition 67
3.3.3.1.1 Tspi_SetAttribUint32 67
3.3.3.1.2 Tspi_GetAttribUint32 68
3.3.3.1.3 Tspi_SetAttribData 69
3.3.3.1.4 Tspi_GetAttribData 70
3.3.3.1.5 Tspi_ChangeAuth 71
3.3.3.1.6 Tspi_ChangeAuthAsym 72
3.3.3.1.7 Tspi_GetPolicyObject 73
3.3.3.2 Tspi_Context Class Definition 74
3.3.3.2.1 Tspi_Context_Create 74

3.3.3.2.2 Tspi_Context_Close 75
3.3.3.2.3 Tspi_SetAttribUint32 76
3.3.3.2.4 Tspi_GetAttribUint32 77
3.3.3.2.5 Tspi_SetAttribData 78
3.3.3.2.6 Tspi_GetAttribData 79
3.3.3.2.7 Tspi_Context_Connect 80
3.3.3.2.8 Tspi_Context_FreeMemory 81
3.3.3.2.9 Tspi_Context_GetDefaultPolicy 82
3.3.3.2.10 Tspi_Context_CreateObject 83
3.3.3.2.11 Tspi_Context_CloseObject 84
3.3.3.2.12 Tspi_Context_GetCapability 85
3.3.3.2.13 Tspi_Context_GetTPMObject 87
3.3.3.2.14 Tspi_Context_LoadKeyByBlob 88
3.3.3.2.15 Tspi_Context_LoadKeyByUUID 89
3.3.3.2.16 Tspi_Context_RegisterKey 90
3.3.3.2.17 Tspi_Context_UnregisterKey 91
3.3.3.2.18 Tspi_Context_GetKeyByUUID 92
3.3.3.2.19 Tspi_Context_GetKeyByPublicInfo 93
3.3.3.2.20 Tspi_Context_GetRegisteredKeysByUUID 94
3.3.3.3 TSPI_Policy Class Definition 96
3.3.3.3.1 Tspi_SetAttribUint32 96
3.3.3.3.2 Tspi_GetAttribUint32 98
3.3.3.3.3 Tspi_SetAttribData 99
3.3.3.3.4 Tspi_GetAttribData 100
3.3.3.3.5 Tspi_Policy_SetSecret 101
3.3.3.3.6 Tspi_Policy_FlushSecret 102
3.3.3.3.7 Tspi_Policy_AssignToObject 103
3.3.3.4 Tspi_TPM Class Definition 104
3.3.3.4.1 Tspi_SetAttribUint32 104
3.3.3.4.2 Tspi_GetAttribUint32 105
3.3.3.4.3 Tspi_SetAttribData 106
3.3.3.4.4 Tspi_GetAttribData 107
3.3.3.4.5 Tspi_TPM_CreateEndorsementKey 108
3.3.3.4.6 Tspi_TPM_GetPubEndorsementKey 109
3.3.3.4.7 Tspi_TPM_TakeOwnership 110
3.3.3.4.8 Tspi_TPM_CollateIdentityRequest 111
3.3.3.4.9 Tspi_TPM_ActivateIdentity 113
3.3.3.4.10 Tspi_TPM_ClearOwner 115
3.3.3.4.11 Tspi_TPM_SetStatus 116
3.3.3.4.12 Tspi_TPM_GetStatus 118
3.3.3.4.13 Tspi_TPM_SelfTestFull 120
3.3.3.4.14 Tspi_TPM_CertifySelfTest 121
3.3.3.4.15 Tspi_TPM_GetTestResult 122
3.3.3.4.16 Tspi_TPM_GetCapability 123
3.3.3.4.17 Tspi_TPM_GetCapabilitySigned 125
3.3.3.4.18 Tspi_TPM_CreateMaintenanceArchive 127
3.3.3.4.19 Tspi_TPM_KillMaintenanceFeature 128
3.3.3.4.20 Tspi_TPM_LoadMaintenancePubKey 129
3.3.3.4.21 Tspi_TPM_CheckMaintenancePubKey 130
3.3.3.4.22 Tspi_TPM_GetRandom 131
3.3.3.4.23 Tspi_TPM_StirRandom 132
3.3.3.4.24 Tspi_TPM_AuthorizeMigrationTicket 133
3.3.3.4.25 Tspi_TPM_GetEvent 134
3.3.3.4.26 Tspi_TPM_GetEvents 135
3.3.3.4.27 Tspi_TPM_GetEventLog 136
3.3.3.4.28 Tspi_TPM_Quote 137
3.3.3.4.29 Tspi_TPM_PcrExtend 138
3.3.3.4.30 Tspi_TPM_PcrRead 139
3.3.3.4.31 Tspi_TPM_DirWrite 140
3.3.3.4.32 Tspi_TPM_DirRead 141
3.3.3.4.33 Tspi_ChangeAuth 142
3.3.3.4.34 Tspi_GetPolicyObject 143

3.3.3.5 Tspi_Key Class Definition 144
3.3.3.5.1 Tspi_SetAttribUint32 144
3.3.3.5.2 Tspi_GetAttribUint32 146
3.3.3.5.3 Tspi_SetAttribData 148
3.3.3.5.4 Tspi_GetAttribData 149
3.3.3.5.5 Tspi_Key_LoadKey 150
3.3.3.5.6 Tspi_Key_UnloadKey 151
3.3.3.5.7 Tspi_Key_GetPubKey 152
3.3.3.5.8 Tspi_Key_CertifyKey 153
3.3.3.5.9 Tspi_Key_CreateKey 154
3.3.3.5.10 Tspi_Key_WrapKey 155
3.3.3.5.11 Tspi_Key_CreateMigrationBlob 156
3.3.3.5.12 Tspi_Key_ConvertMigrationBlob 158
3.3.3.5.13 Tspi_ChangeAuth 159
3.3.3.5.14 Tspi_ChangeAuthAsym 160
3.3.3.5.15 Tspi_GetPolicyObject 161
3.3.3.6 Tspi_Hash Class Definition 162
3.3.3.6.1 Tspi_SetAttribData 162
3.3.3.6.2 Tspi_Hash_Sign 162
3.3.3.6.3 Tspi_Hash_VerifySignature 164
3.3.3.6.4 Tspi_Hash_SetHashValue 165
3.3.3.6.5 Tspi_Hash_GetHashValue 166
3.3.3.6.6 Tspi_Hash_UpdateHashValue 167
3.3.3.7 Tspi_Data Class Definition 168
3.3.3.7.1 Tspi_SetAttribUint32 168
3.3.3.7.2 Tspi_GetAttribUint32 169
3.3.3.7.3 Tspi_SetAttribData 170
3.3.3.7.4 Tspi_GetAttribData 171
3.3.3.7.5 Tspi_Data_Bind 172
3.3.3.7.6 Tspi_Data_Unbind 173
3.3.3.7.7 Tspi_Data_Seal 174
3.3.3.7.8 Tspi_Data_Unseal 176
3.3.3.7.9 Tspi_ChangeAuth 178
3.3.3.7.10 Tspi_ChangeAuthAsym 179
3.3.3.7.11 Tspi_GetPolicyObject 180
3.3.3.8 Tspi_PcrComposite Class Definition 181
3.3.3.8.1 Tspi_PcrComposite_SelectPcrIndex 181
3.3.3.8.2 Tspi_PcrComosite_SetPcrValue 182
3.3.3.8.3 Tspi_PcrComposite_GetPcrValue 183
3.3.4 Callback Function Definitions 184
3.3.4.1 Tspicb_CallbackHMACAuth 184
3.3.4.2 Tspicb_CallbackXorEnc 187
3.3.4.3 Tspicb_CallbackTakeOwnership 190
3.3.4.4 Tspicb_CallbackChangeAuthAsym 192
3.3.4.5 Tspicb_CollateIdentity 194
3.3.4.6 Tspicb_ActivateIdentity 195

4. TCG CORE SERVICES (TCS) 196

4.1 TCS ARCHITECTURE AND INTERFACE DESCRIPTION 196
4.1.1 TCS Memory Manager 196
4.1.2 TCS Data Marshalling 196
4.1.3 TCS Interface Dynamics 196
4.2 TCS-SPECIFIC RETURN CODE DEFINES 197
4.3 TSPI-SPECIFIC RETURN CODE RULES 197
4.4 STRUCTURES AND DEFINITIONS 197
4.4.1 Data Types of the Tcsi 197
4.4.2 TCS_LOADKEY_INFO 198
4.5 TCS CONTEXT MANAGER 198
4.5.1 TCS Context Manager Functions and Operations 198
4.5.2 TCS Context Manager Interface 199

4.5.2.1 Tcsi_OpenContext....199
4.5.2.2 Tcsi_CloseContext....200
4.5.2.3 Tcsi_FreeMemory....201
4.5.2.4 Tcsi_GetCapability....202
4.6 TCS KEY AND CREDENTIAL MANAGER....204
4.6.1 TCS Key & Credential Manager Functions and Operations....204
4.6.1.1 TCS Key Manager....204
4.6.1.2 TCS Key Cache Manager....204
4.6.1.3 TCS Credential Management....206
4.6.2 TCS Key and Credential Manager Interface....207
4.6.2.1 Interfaces....207
4.6.2.2 Key Registration....207
4.6.2.2.1 Tcsi_RegisterKey....207
4.6.2.2.2 Tcsip_UnregisterKey....209
4.6.2.3 TCS Get Key Hierarchy Information....210
4.6.2.3.1 Tcsi_EnumRegisteredKeys....210
4.6.2.3.2 Tcsi_GetRegisteredKey....212
4.6.2.3.3 Tcsi_GetRegisteredKeyBlob....213
4.6.2.3.4 Tcsip_GetRegisteredKeyByPublicInfo....214
4.6.2.4 TCS Loading a Key....216
4.6.2.4.1 Tcsip_LoadKeyByBlob....216
4.6.2.4.2 Tcsip_LoadKeyByUUID....218
4.6.2.4.3 Tcsip_EvictKey....220
4.6.2.5 TCS Creating a Key....221
4.6.2.5.1 Tcsip_CreateWrapKey....221
4.6.2.6 TCS Working with Keys....223
4.6.2.6.1 Tcsip_GetPubKey....223
4.6.2.7 TCS Credential Management....225
4.6.2.7.1 Tcsip_MakeIdentity....225
4.6.3 TCS Use Models....228
4.6.3.1 TCS Load Key by UUID....228
4.7 TCS EVENT MANAGER....229
4.7.1 TCS Event Manager Functions and Operations....229
4.7.2 TCS Event Manager Interface....229
4.7.2.1 TCS Event Manager Interface Structures and Definitions....229
4.7.2.1.1 TCS The Event Log....229
4.7.2.2 TCS Event Manager Interface Functions....230
4.7.2.2.1 Tcsi_LogPcrEvent....230
4.7.2.2.2 Tcsi_GetPcrEvent....232
4.7.2.2.3 Tcsi_GetPcrEventsByPcr....234
4.7.2.2.4 Tcsi_GetPcrEventLog....236
4.8 TCS AUDIT MANAGER....237
4.8.1 TCS Audit Manager Functions and Operations....237
4.9 TCS TPM PARAMETER BLOCK GENERATOR....237
4.9.1 TCS TPM Parameter Block Generator Functions and Operations....237
4.9.2 TCS TPM Parameter Block Generator Interface....237
4.9.2.1 Functions....237
4.9.2.2 TPM Ownership, Authorization, and Identity....239
4.9.2.2.1 Tcsip_SetOwnerInstall....239
4.9.2.2.2 Tcsip_TakeOwnership....240
4.9.2.2.3 Tcsip_OIAP....242
4.9.2.2.4 Tcsip_OSAP....243
4.9.2.2.5 Tcsip_ChangeAuth....244
4.9.2.2.6 Tcsip_ChangeAuthOwner....246
4.9.2.2.7 Tcsip_ChangeAuthAsymStart....247
4.9.2.2.8 Tcsip_ChangeAuthAsymFinish....249
4.9.2.2.9 Tcsip_TerminateHandle....251
4.9.2.2.10 Tcsip_ActivateTPMIdentity....252
4.9.2.3 TPM Mandatory....254
4.9.2.3.1 Tcsip_Extend....254
4.9.2.3.2 Tcsip_PcrRead....255

4.9.2.3.3 Tcsip_Quote 256
4.9.2.3.4 Tcsip_DirWriteAuth 258
4.9.2.3.5 Tcsip_DirRead 259
4.9.2.3.6 Tcsip_Seal 260
4.9.2.3.7 Tcsip_Unseal 262
4.9.2.3.8 Tcsip_UnBind 264
4.9.2.3.9 Tcsip_CreateMigrationBlob 265
4.9.2.3.10 Tcsip_ConvertMigrationBlob 267
4.9.2.3.11 Tcsip_AuthorizeMigrationKey 269
4.9.2.4 TPM Cryptographic Capabilities 271
4.9.2.4.1 Tcsip_CertifyKey 271
4.9.2.4.2 Tcsip_Sign 273
4.9.2.4.3 Tcsip_GetRandom 274
4.9.2.4.4 Tcsip_StirRandom 275
4.9.2.4.5 Tcsip_GetCapability 276
4.9.2.4.6 Tcsip_GetCapabilitySigned 277
4.9.2.4.7 Tcsip_GetCapabilityOwner 279
4.9.2.5 TPM Endorsement Credentials 280
4.9.2.5.1 Tcsip_CreateEndorsementKeyPair 280
4.9.2.5.2 Tcsip_ReadPubek 281
4.9.2.5.3 Tcsip_DisablePubekRead 282
4.9.2.5.4 Tcsip_OwnerReadPubek 283
4.9.2.6 TPM Self-Test and Management 284
4.9.2.6.1 Tcsip_SelfTestFull 284
4.9.2.6.2 Tcsip_CertifySelfTest 285
4.9.2.6.3 Tcsip_ContinueSelfTest 286
4.9.2.6.4 Tcsip_GetTestResult 287
4.9.2.6.5 Tcsip_OwnerSetDisable 288
4.9.2.6.6 Tcsip_OwnerClear 289
4.9.2.6.7 Tcsip_DisableOwnerClear 290
4.9.2.6.8 Tcsip_ForceClear 291
4.9.2.6.9 Tcsip_DisableForceClear 292
4.9.2.6.10 Tcsip_PhysicalDisable 293
4.9.2.6.11 Tcsip_PhysicalEnable 294
4.9.2.6.12 Tcsip_PhysicalSetDeactivated 295
4.9.2.6.13 Tcsip_SetTempDeactivated 296
4.9.2.6.14 Tcsip_PhysicalPresence 297
4.9.2.6.15 Tcsip_FieldUpgrade 298
4.9.2.6.16 Tcsip_SetRedirection 299
4.9.2.7 TPM Optional 300
4.9.2.7.1 Tcsip_CreateMaintenanceArchive 300
4.9.2.7.2 Tcsip_LoadMaintenanceArchive 301
4.9.2.7.3 Tcsip_KillMaintenanceArchive 302
4.9.2.7.4 Tcsip_LoadManufaturerlMaintenancePub 303
4.9.2.7.5 Tcsip_ReadManufacturerMaintenancePub 304

5. TCG DEVICE DRIVER LIBRARY (TDDL) 305

5.1 TDDL ARCHITECTURE 305
5.2 MEMORY MANAGEMENT 305
5.3 TDDL ERROR CODE DEFINES 305
5.4 TDDL-SPECIFIC RETURN CODE RULES 306
5.5 TDDL INTERFACE 307
5.5.1 Tddli_Open 307
5.5.2 Tddli_Close 308
5.5.3 Tddli_Cancel 309
5.5.4 Tddli_GetCapability 310
5.5.5 Tddli_SetCapability 312
5.5.6 Tddli_GetStatus 313
5.5.7 Tddli_TransmitData 314

List of Figures

Figure 1-1 TSS Block Diagram and Description of Sections 12
Figure 1-2 TSS Architecture Diagram 17
Figure 2-1 Load Key Flow Diagram 52
Figure 2-2 Load Key Flow Diagram 55
Figure 3-1 Object Relationship 60
Figure 3-2 Non authorized working object relationship 60
Figure 3-3 Authorized working object relationship 61
Figure 4-1 TCS Context Manager and Operations 198

 Copyright TCG 2003

Preface

This document is an industry specification that enables trust in computing platforms in general.

This specification defines a TCG Software Stack (TSS) that is an integral part of each platform, and provides functions that can be used by enhanced operating systems and applications. The software stack employs cryptographic methods when establishing trust, and while this does not in itself convert a platform into a secure computing environment, it is a significant step in that direction.

Standardization is necessary so that the security and cryptographic community can assess the mechanisms involved, and so that customers can understand and trust the effectiveness of new features. Manufacturers will compete in the marketplace by installing software stacks with varying capabilities and cost points. The software stack itself will have basic functions that maintain privacy, yet support the identity and authentication of entities such as the platform, the user, and other entities. The software stack will have other capabilities to protect data and verify certain operational aspects of the platform. It can be a separate device or devices, or it can be integrated into some existing component or components provided the implementation meets the requirements of this specification. This is necessary to achieve the fundamental goal of ubiquity.

Please note a very important distinction between different sections of text throughout this document. Beginning after this section, you will encounter two distinctive kinds of text: informative comment and normative statements. Because most of the text in this specification will be of the kind normative statements, the authors have informally defined it as the default and, as such, have specifically called out text of the kind informative comment. They have done this by flagging the beginning and end of each informative comment and highlighting its text in gray. This means that unless text is specifically marked as of the kind informative comment, you can consider it of the kind normative statements.

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in the chapters 2-10 normative statements are to be interpreted as described in [RFC-2119].

For example:

Informative Statements:

Start of informative comment:

This is the first paragraph of 1–n paragraphs containing text of the kind informative comment,

This is the second paragraph of text of the kind informative comment,

This is the nth paragraph of text of the kind informative comment,

End of informative comment.

Normative Statements:

To understand the TCPA specification the user must read the specification. (This use of MUST does not require any action).

This is the first paragraph of one or more paragraphs (and/or sections) containing the text of the kind normative statements...

To understand the TCPA specification the user MUST read the specification. (This use of MUST indicates a keyword usage and requires an action).

Copyright TCG 2003

1. The TCG Software Stack (TSS)

1.1 General Introduction

The TCG Software Stack (the TSS) is the supporting software on the platform supporting the platform's TPM.

This document is written with the assumption that the reader or TSS implementer has an understanding of the TCPA 1.1b Main Specification (Published under this name by the Trusted Computing Group, Inc.) and cryptographic infrastructures in general.

A note on specification, organization and architectural naming: The Trusted Computing Group has adopted the specification titled "TCPA 1.1b Main Specification". Although the name of the specification remains the TCPA 1.1b Main Specification the architecture and organization are named TCG architecture and the Trusted Computing Group (or TCG), respectively. The name of the specification and definition labels (e.g., structure names, etc.) were retained to preserve existing references and implementations and are therefore retained in this specification.

Start of informative comment:

The TCPA Main specification defines a subsystem with protected storage and protected capabilities. This subsystem is the Trusted Platform Module (TPM). Since the TPM is both a subsystem intended to provide trust and be an inexpensive component, resources within it are restricted. This narrowing of the resources, while making the security properties easier and cheaper to build and verify, causes to the interfaces and capabilities to be cumbersome. The TCG architecture has solved this by separating the functions requiring the protected storage and capabilities from the functions that do not; putting those that do not into the platform's main processor and memory space where processing power and storage exceed that of the TPM. The modules and components that provide this supporting functionality comprise the TSS.

End of informative comment.

1.2 Introduction to the TSS

Start of informative comment:

The block diagram in Figure 1-1 TSS Block Diagram and Description of Sections illustrates the TSS modules, components and there relationships. TSS modules are the major parts of the TSS providing fundamental resources to support the TPM. Each module is comprised of components, which provide the more specialized functions of the modules.

The primary design goals are:

- Supply one entry point for applications to the TPM functionality
- Provide synchronized access to the TPM
- Hide building command streams with appropriate byte ordering and alignment from the applications
- Manage TPM resources
- Release TPM resources when appropriate

End of informative comment.

The TSS is comprised of discreet modules and are designed to be discreet subsystems, therefore, interfaces between them are defined in this specification to provide interoperability. The horizontal dashed lines represent these interfaces. While architecturally there is no constraint on the nature of the interfaces, 'C' and IDL are specifically defined in this revision of the TSS Specification. Modules are comprised of one or more internal components. Interfaces between components within each

Copyright TCG 2003

modules is a design consideration for each module and does not affect interoperability, therefore, this specification does not address these interfaces.

Figure 1-1 TSS Block Diagram and Description of Sections depicts the components of the TCG software stack and their related sections.

1.3 TSS functions defined are not exclusive

By providing all the functions listed within the normative text the TSS implementation SHALL be considered a complete and valid implementation (compliance testing, if any that may be defined, notwithstanding.) The list of functions defined within this specification (at any level) may be augmented by a TSS component provider to add extra functionality. The addition of any functions do not exclude a TSS implantation form being considered a valid implementation.

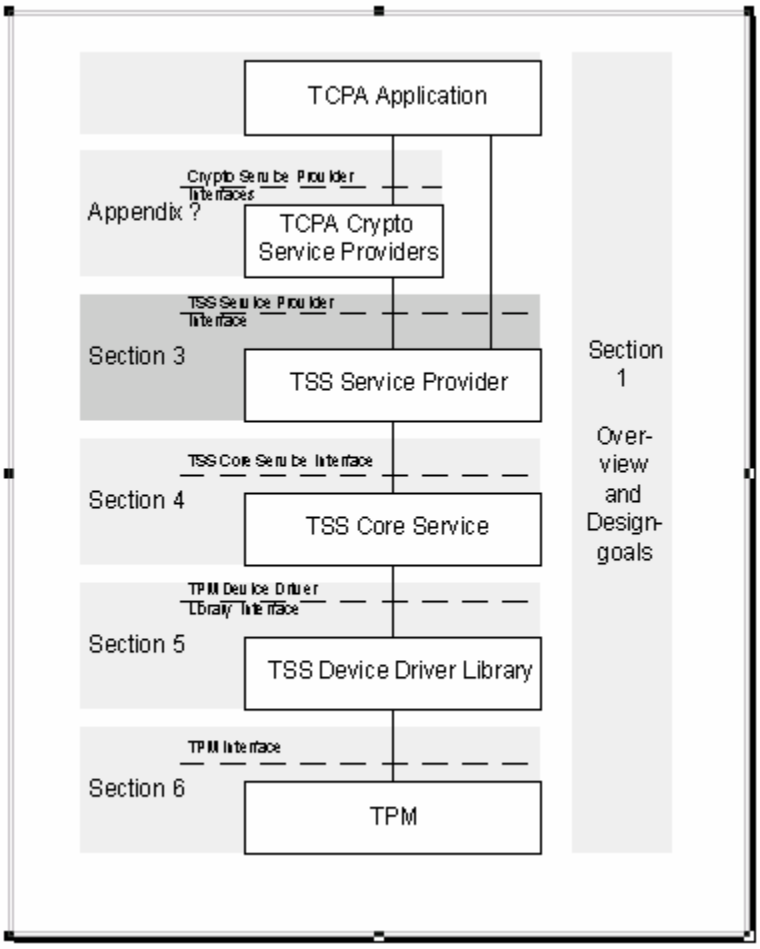


Figure 1-1 TSS Block Diagram and Description of Sections

1.4 Platform Architecture

Start of informative comment:

The architecture of the TSS defined in this specification is intended to be neither platform nor operating system dependent. While the nature and extent of the modules and components will be vary for each platform and operating system, the modules, their interactions, and relationship will all be the same whether this platform is a large server or a personal digital assistant. While the nature of the platforms and it operating systems will vary, the TSS architecture assumes all will have the attributes described to the left of the diagram in Figure 1-2 TSS Architecture Diagram.

End of informative comment.

1.3.1 Platform Modes

Start of informative comment:

Kernel Mode: This is where the device drivers and the core components of the operating system reside. Code in this area services and protects the applications running in User Mode. Code in this area usually requires some type of administration authorization to modify or update. Kernel mode typically doesn't have separate processes separating the execution of the executable within it.

User Mode: This is where user applications and services execute. These applications and services are usually loaded and executed at the request of the user but may be loaded as part of a startup sequence. The operating system provides one or more protected areas called processes for different application or services to execute providing protection from each other. Within User Mode there are typically two classes of applications:

User Applications: These are executable code run at the request or behalf of the user. Due to the fact that this code is initiated and may be provided by the user, it may not be as trusted as other code executing on the platform.

System Services: These applications are generally started during initialization of the operating system as part of a startup script or may be initiated as a result of a request of the server. These typically provide common and usually trusted services for the User Applications. Since this code executes in its own process, separate from the User Application and is started by the operating system, it is considered more trusted than User Application and my be as trusted as Kernel Mode executables. In a Windows operating system this is a "System Service" while in a Unix operating system this would a be a daemon.

End of informative comment.

1.3.2 Procedure Calls

Start of informative comment:

There are two methods a routine calls another routine: Local Procedure Call (LPC) or Remote Procedure Call (RPC).

End of informative comment.

1.3.2.1 Local Procedure Call (LPC)

Start of informative comment:

An LPC is a direct call from one application, module, or component to another within the same process or address space allocated by the OS. In an LPC the calling routine can resolve directly or indirectly the locally address of the routine to call or pass control to.

End of informative comment.

Copyright TCG 2003

1.3.2.2 Remote Procedure Call (RPC)

Start of informative comment:

An RPC provides the interaction between processes. It is:

- A set of rules for marshalling and unmarshalling parameters and results;
- A set of rules for encoding and decoding information transmitted between two processes;
- A few primitive operations to invoke an individual call, to return its results, and to cancel it;
- Provision in the operating system and process structure to maintain and reference state that is shared by the participating processes.

RPC requires a communications infrastructure to set up the path between the processes and provide a framework for naming and addressing. Anything that provides these services will do.

End of informative comment.

1.4 Trust Boundaries

Start of informative comment:

Security architectures are best done using a layered and simple approach separating the components that require trust from the components that don't. An architecture where lines can be drawn around and between the trusted and undusted modules allows for easier maintenance and validation of the security properties of the system. This line is historically called the Trusted Computing Base or TCB. The components below or within the TCB are small, simple and trusted while the components outside the TCB can be more complex and larger. In the TCG architecture, the boundary around the TPM is the TCB. All components outside the TPM (i.e., the TCB) are untrusted such as the TSS, OS, and applications.

End of informative comment.

1.5 Roles

Start of informative comment:

Entities (human or machine) perform functions on the TSS and the TPM. These functions are performed while acting in one of the following roles:

End of informative comment.

1.5.1 TPM Owner

Start of informative comment:

This is the entity that owns and has "title" to the platform. There is only one TPM owner of the platform. Since proof of ownership is made by the presentation of authentication data, if the owner chooses, the owner's authentication data can be shared among other entities allowing a form of delegation.

Examples:

In a corporate environment, the TPM owner would be the corporation or IT department.

In a home environment, the TPM owners would be the person who purchased the platform.

End of informative comment.

1.5.2 TPM User

Start of informative comment:

These are entities that can load or use TPM objects such as TPM keys. It is important to understand that the TPM itself does not maintain a list of TPM users. A TPM user is any entity that can present the authentication data for an object. The first TPM user is created by the TPM owner. Thereafter more TPM users can be created by either the TPM owner or other TPM users.

A TPM user is not necessarily a human. A TPM user may be service provider such as a mail or file server.

Examples:

In a corporate environment, a TPM user would be an employee of the company or the servers of the corporation.

In a home environment, a TPM user would be a member of the household that the TPM owner allows access to TPM objects or external service providers such as a bank or stock broker.

End of informative comment.

1.5.3 Platform Administrator

Start of informative comment:

This is the entity that controls the platform's OS, filesystem, or data. The Platform administrator may or may not be the TPM owner.

End of informative comment.

1.5.4 Platform User

Start of informative comment:

These are entities that the Platform administrator allows use of the data or resources of the platform. The Platform users may or may not be TPM users.

End of informative comment.

1.5.5 Operator

Start of informative comment:

The human physically at the platform able to directly operate it and observe physical indications. The operator may or may not be a TPM owner or TPM user but will likely be either a Platform administrator or Platform user.

End of informative comment.

1.5.6 Public

Start of informative comment:

Performs any function on either the platform's OS, filesystem, or data allowed without an identity or authentication. Performs any function on the TPM that does not require authentication.

End of informative comment.

Copyright TCG 2003

1.6 TSS Architecture

Start of informative comment:

Describing the modules starting from the bottom up:

- The TPM Device driver is typically provide by the TPM manufacture and incorporates code that has understanding of the specific behavior of the TPM. This code is expected to be loaded and function in Kernel Mode. Since User Mode executables cannot directly access Kernel Mode executables, the manufacture also provides the TCG Device Driver Library. The TSS exclusively opens the TPM device driver; the driver does not allow any applications to have an additional connection to the TPM device besides the TSS.
- The TCG Device Driver Library (TDDL) provides two functions:
    - A standard interface defined in this specification for the TPM so all TPMs look and behave the same at this interface (Tddli).
    - Provides the transition between the User Mode and Kernel Mode. There will typically be one executable image of each of these per TPM on the platform.
- The TSS Core Services (TCS) resides in User Mode and communicates to the TPM via the TPM Device Drivers Library Interface (Tddli) provided by the TDDL. There will typically be one image of this component per platform and it executes as a system service. This module provides all the primitives and more sophisticated functions such as key management required to efficiently manage the TPM's limited resources. The interface to the TCS is the TSS Core Service Interface (Tcsi). This interface is designed to provide a straight forward, simple method for controlling and requesting services form the TPM. The functions are designed to be atomic in natures requiring little setup and overhead.
- TSS Service Providers (TSP) are the top-most modules and provide a rich, object-oriented interface for applications to incorporate the full capabilities of a TCG-enabled platform. The interface used by the applications to access the TSP is the TSS Service Provider Interface (Tspi). While not an architecturally requirement, it is intended that the TSP obtain many TCG services such as TPM byte stream generation, key management, etc from the TCS.

Another type of module that may make us the TCS is an RPC server. This module marshals the TCS functions and data from the TCG platform to another platform or device.

It's important to emphasize that since the security properties of the TPM's protocols have been specifically designed to not rely on the security properties of the data transport, none of the TSS modules, components or event the RPC communications affect the trusted properties of the TPM. All modules, components and interfaces outside the TPM are considered untrusted in relation to the TPM.

End of informative comment.

Copyright TCG 2003

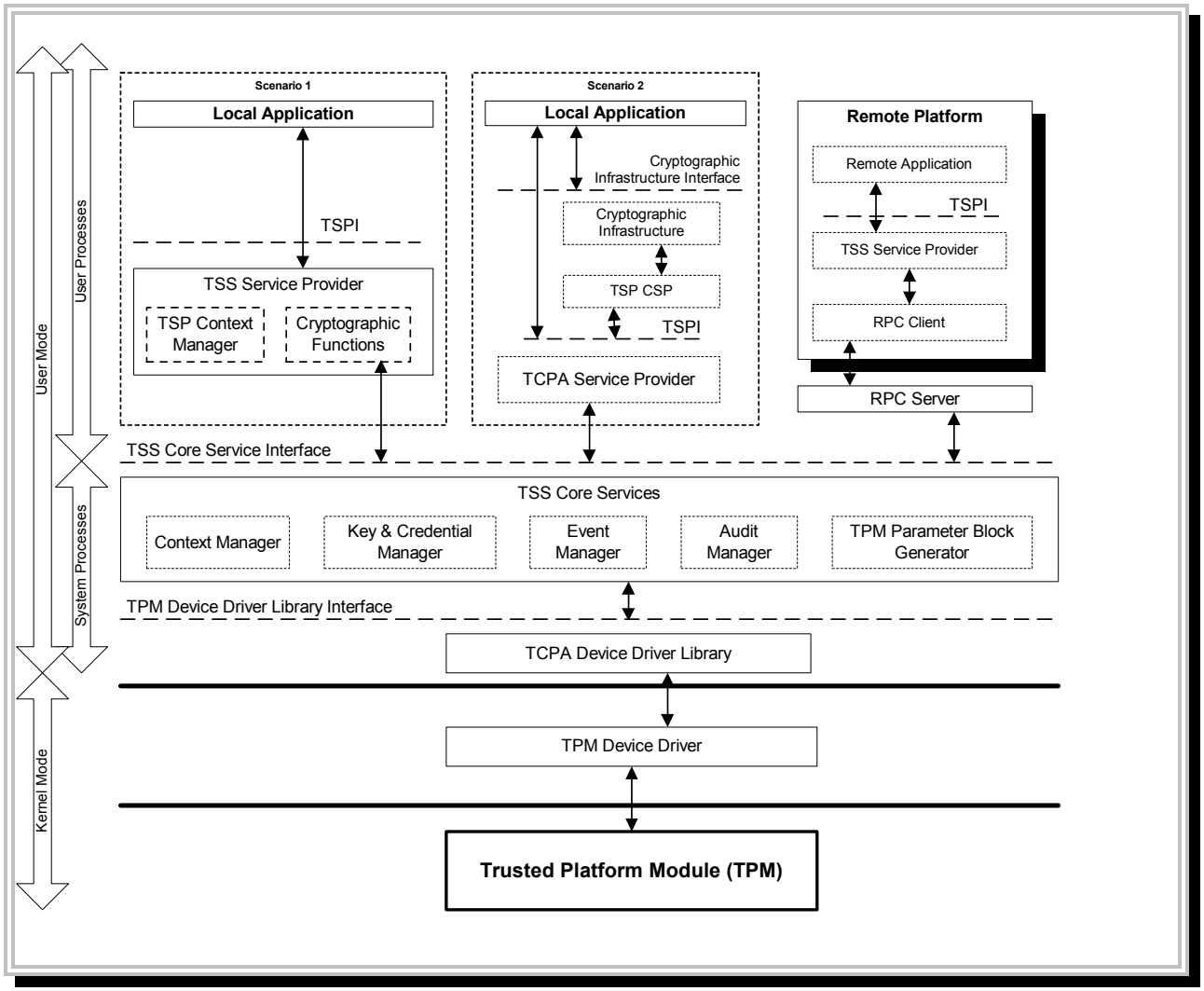


Figure 1-2 TSS Architecture Diagram

1.6.1 TCG Service Provider (TSP)

Start of informative comment:

This module provides TCG services for applications. It provides the high-level TCG functions allowing applications to focus on their specialty while relying on the TSP to perform most of the trusted functions provided by the TPM. This module also provides a small number of auxiliary functions for convenience not provided by the TPM such as hashing.

In environments that provide layers of protections (i.e., rings) or separation of applications into processes, this module is intended to reside within the same ring and process as the application. There will likely be one TSP per application. On operating systems that provide multiple processes, there may be multiple TSPs residing on the platform.

End of informative comment.

A TSS implementation MUST provide the functionality of the TCG Service Provider (TSP). Depending on the implementation, the TSP MAY be a discreet module or MAY be integrated into other platform modules. This module MAY provide protected transfer of information of data between the application by residing within the same process as the application.

The TSP MUST provide the 'C' interface as defined in section 3 TCG Service Provider (TSP) of this specification. For the 'C' interface the TSP MUST provide dynamic linking to the application and MAY provide static linking to the application. If the TSP is implemented in a Microsoft® Windows® application, the TPM MUST, in addition, provide a COM interface.

1.6.1.1 TSP Interface (TSPI)

Start of informative comment:

The interface to the TSP is the TSP Interface (TSPI). This is an object oriented interface. It resides within the same process as the application. In some implementations it may be trusted with sensitive data such as authorization data to the same extent as the application itself. This may be required for functions where the application gathers the object's authentication data from the user an passes it to the TSP for processing into the command's authentication data format.

End of informative comment.

If a TSS provides a TSP, it MUST provide the required interfaces defined in this document.

1.6.1.2 TSP Context Manager (TSPCM)

Start of informative comment:

The TSP Context Manager (TSPCM) provides dynamic handles that allow for efficient usage of both the application's and TSP's resources. Each handle provides context for a set of interrelated TCG operations. Different threads within the application may share the same context or may acquire a separate context per thread.

End of informative comment.

If a TSS provides a TSP, it MUST provide the functions required to establish and maintain context for a set of interrelated functions.

1.6.1.3 TSP Cryptographic Functions (TSPCF)

Start of informative comment:

To make full use the TPM's protected functions, supporting cryptographic functions must be provided. It is not necessary for these supporting functions to be protected. Example functions are: Hashing algorithm and Byte-stream generator.

End of informative comment.

If a TSS provides a TSP, it MUST provide the cryptographic functions required as defined in section 3 TCG Service Provider (TSP).

1.6.2 TCG Core Services (TCS)

Start of informative comment:

A service provider is any component used by the application that allows that application access to the TCS (and thus the TPM) from within the application's process. Service providers, of which the TSP is but one possible instantiation, cannot communicate directly with the TPM. Additionally, there are multiple common services that either must or should be shared among the set of the platform's service providers.

The TCG Core Services (TCS) provides a common set of services per platform for all service providers. Since the TPM is not required to be multithreaded, it provides threaded access to the TPM.

End of informative comment.

Any service provider MUST be allowed to connect to and obtain services from the TCS. The TCS MUST not restrict access to only a TSP.

The TCS MUST provide single threaded access to the TPM.

Copyright TCG 2003

There MUST be only one TCS per platform operating system.

1.6.2.1 TCS Interface (Tcsi)

Start of informative comment:

The interface to the TCS is the TCS Interface (Tcsi). This is a simple 'C' style interface. While it may be allow multithreaded access to the TCS, each operation is intended to be atomic. In most environments it resides as a system process, separate from the application and service provider processes. If the environment provides for the TCS to reside in a system process, communication between the service providers and the TCS would be via an RPC. Because the channel between the service provider and the TCS will be outside the protection of the process, no unprotected data (e.g., raw authentication data) should be transferred directly to the TCS.

End of informative comment.

If a TSS provides a TCS, it MUST provide the required interfaces defined in this document.

If the platform's environment provides, the TCS MUST function as a system service.

If the TCS functions as a system service, the TCS MUST NOT be implemented to allow the transfer of raw authentication data between the service provider and the TCS.

1.6.2.2 TCS Context Manager (TCSCM)

Start of informative comment:

The TCS Context Manager (TCSCM) provides dynamic handles that allow for efficient usage of both the service provider's and TCS's resources. Each handle provides context for a set of interrelated TCG operations. Different threads within the service provider may share the same context or may acquire a separate context per service provider.

End of informative comment.

The TCS MUST provide the functions required to establish and maintain context for a set of interrelated functions.

1.6.2.3 TCS Key & Credential Manager (TCSKCM)

Start of informative comment:

Key and credential may be associated with the platform, the user, or the individual applications. In all cases it may be more convenient for an application to use a common resource to store and manage the keys and credentials. Key and credentials associated with the platform (e.g., The Endorsement, Platform, and Conformance Credentials) should be stored and managed by the Key and Credential Manager to allow multiple application access to them.

The Endorsement and Platform credentials contain information that identifies the specific platform thus these are considered privacy sensitive. Other credentials and keys my also contain privacy sensitive information. The TCSKM must, therefore, provide a mechanism to protect this information from unauthorized access.

End of informative comment.

The TCS MUST provide the functions required to perform Key and Credential Management as described in this document.

The TCSKM MUST provide a mechanism to protect privacy sensitive keys and credentials from unauthorized access.

1.6.2.4 TCS Event Manager (TCSEM)

Start of informative comment:

Copyright TCG 2003

This component manages the TSS_PCR_EVENT structures and their associations with their respective PCR. Since these are associated with the platform and not the application, applications and service providers should retain only copies of these structures.

End of informative comment.

The TCS MUST provide the functions required to store, manage, and report the TSS_PCR_EVENT structures and their associated PCR indexes.

1.6.2.5 TCS TPM Parameter Block Generator (TcsipBG)

Start of informative comment:

Calls into the TCS are 'C'-style functions. Communication to the TPM is via a byte-stream parameter block. This component converts the parameters passed into TCS into the byte-stream expected by the TPM.

End of informative comment.

The TCS MUST provide the functions required to convert input parameters into a TPM byte-stream.

1.6.3 TCG Device Driver Library (TDDL)

Start of informative comment:

The TCG Device Driver Library (TDDL) is an intermediate module that exists between the TCS and the kernel mode TPM Device Driver (TDD). The TDDL provides a user mode interface. Such an interface has several advantages over a kernel mode driver interface:

- It ensures different implementations of the TSS properly communicates with any TPM.
- It provides an OS-independent interface for TPM applications.
- It allows the TPM vendor to provide a software TPM simulator as a user mode component.

Because the TPM is not required to be multithreaded, the TDDL is to be a single-instance, single threaded module. The TDDL expects the TPM command serialization to be performed by the TCS. The exception to the single threaded nature of the TDDL is the Tddli_Cancel operation. The Tddli_Cancel allows the TCS to send an abort operation to the TPM.

The TPM vendor is responsible for defining the interface between the TDDL and the TDD. The TPM vendor can choose the communication and resource allocation mechanisms between this library and any kernel mode TPM device driver or software TPM simulator.

End of informative comment.

If the platform environments provides kernel and user mode separation, the TDDL MUST reside in the user mode.

The TDDL MUST only connect to the TCS.

TPM commands sent to the TDDL MUST be serializes.

The TDDL MUST provide the interface: Tddli_Cancel. This interface MUST function and attempt to send an abort command to the TPM even if a thread is currently busy sending a TPM command.

1.6.3.1 TDDL Interface (Tddli)

Start of informative comment:

The interface to the TDDL is the TDDL Interface (Tddli).

The Tddli is consists of three types of functions:

- Maintenance functions (Open, Close, GetStatus): maintains the communication with the TDD.
- Indirect Functions (GetCapability, SetCapability): gets/sets attributes of the TPM/TDD/TDDL.

- Direct functions (Transmit, Cancel): transmits/cancels transmission of commands to the TPM.

End of informative comment.

1.6.4 TPM Device Driver (TDD)

Start of informative comment:

The TCG Device Driver (TDD) is the kernel mode component that receives byte-streams from the TDDL and sends them to the TPM returning the responses back to the TDDL.

The TDD is TPM vendor and OS specific. It may provides additional functionality such as power management as required by the platform, OS, or operating environment.

End of informative comment.

The TDD MUST be the only platform component to interface with the TPM.

The TDD MUST provide power management functions If the platform's environment provides or requires it.

1.6.4.1 TDD Interface (TDDI)

Start of informative comment:

The interface between the TDD and TDDL is called the TDDI.

End of informative comment.

The TPM vendor is responsible for defining the TDDI. The vendor is also responsible for defining the interface between the TDD and the actual TPM device.

Copyright TCG 2003

1.7 Cryptographic Infrastructures

Start of informative comment:

There are several existing general purpose Cryptographic Infrastructure available to the industry. Examples include: CDSA, MS-CAPI, and PKCS #11. These interfaces usually provide standard underlying interfaces allowing modules that provide specific features or functions to be incorporated and used by the Cryptographic Infrastructure's applications. These custom modules are called service providers.

End of informative comment.

The TSS MAY be one or more of the following:

- A service provider for an existing Cryptographic Infrastructure. This can provide for either all or a subset of the functions required by the Cryptographic Infrastructure. The remaining functions provided by the TSS that do not fit within the scope of the Cryptographic Infrastructure can be provided between the application and the TSP.
- A stand-alone service providing no functions to an existing Cryptographic Infrastructure.

Note: While the TSS does not provide bulk, general purpose symmetric encryption/decryption functions symmetric encryption/decryption functions are required by some TPM and TSS operations. Encryption/decryption functions MAY be provided either internally or externally to the TSS.

Copyright TCG 2003

2. Common Environment

2.1 Naming Conventions

Label	Any Value Appropriate to the Platform Addressing a Handle
TCS	When used as a prefix, this is a structure or definition that applies only the TCS layer.
Tcsi	TCS Interface Prefix where the function does **not** result in a direct call to the TPM.
Tcsip	TCS Interface Prefix where the function **does** result in a direct call to the TPM
Tddli	TSS Device Driver Library Interface Prefix.
Tddi	TSS Device Driver Interface Prefix (To be defined in a future version of the TSS Specification).
Tspi	TSP Interface Prefix.
Tspicb	Application supplied callback functions for the TSP prefix
TSS	When used as a prefix, this is a structure or definition used in the TSP or other TSS layers.

2.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
HMAC	Hashed Message Authentication Code
OIAP	Object Independent Authorization Protocol
OSAP	Object Specific Authorization Protocol
PCR	Platform Configuration Register
TCG	Trusted Computing Group
TCS	TSS Core Services Module.
TDD	TSS Device Driver Module.
TDDL	TSS Device Driver Library Module.
TPM	Trusted Platform Module
TSP	TSS Service Provider
TSS	TCG Software Stack
VE	Validation Entity

Copyright TCG 2003

2.3 Definitions

2.3.1 Data Types

TCG-enabled platforms are expected to be 32-bit or 64-bit systems. This section defines the basic data types on these systems. The section after the next section then contains data type declarations especially required at the TSPI. Data type declarations for the below definitions are not included in the TSS headers files. These are expected to be defined by the platform-specific SDK and header files for the target environment and platform. They MUST, however, conform to the specific definitions below since these are TPM specific definitions.

Pointer Precision:

Pointer precision becomes 32 bits on 32-bit systems and 64 bits with 64-bit system.

Basic Types:

There are some new types for 64-bit systems that were derived from the basic C-language integer and long types, so they work in existing code. These are the expected values and definitions.

Type	Definition
UINT16	Unsigned INT16
UINT32	Unsigned INT32.
BYTE	Unsigned character
TSS_BOOL	Signed character
UNICODE	UNICODE character. UNICODE characters are to be treated as an array of 16 bits. Unless otherwise stated, the size of the UNICODE string includes the 16 bit null terminating character.
PVOID	void Pointer (32 or 64 bit depending on architecture)

Boolean types

Name	Value	Description
TRUE	0x01	Assertion
FALSE	0x00	Contradiction

Derived Types

Type	Definition	Usage
TSS_FLAG	UINT32	Object attributes.
TSS_HOBJECT	UINT32	Basic object handle.
TSS_ALGORITHM_ID	UINT32	Type of TSS Algorithm IDs
TSS_MIGRATE_SCHEME	UINT16	Type of TSS Migration Scheme IDs
TSS_KEY_USAGE_ID	UINT32	Type of TSS Key Usage IDs
TSS_KEY_ENC_SCHEME	UINT16	Type of TSS Encryption Scheme IDs
TSS_KEY_SIG_SCHEME	UINT16	Type of TSS Signature Scheme IDs
TSS_EVENTTYPE	UINT32	Type of TSS event
TSS_RESULT	UINT32	Result of a TSP inteface command

Object Types

Type	Definition	Usage
TSS_HCONTEXT	TSS_HOBJECT	Context object handle.
TSS_HPOLICY	TSS_HOBJECT	Policy object handle.

Copyright TCG 2003

TSS_HTPM	TSS_HOBJECT	TPM object handle.
TSS_HKEY	TSS_HOBJECT	Key object handle.
TSS_HENCDATA	TSS_HOBJECT	Encrypted data object handle.
TSS_HPCRS	TSS_HOBJECT	PCR composite object handle.
TSS_HHASH	TSS_HOBJECT	Hash object handle.

2.3.2 Defined Constants

2.3.2.1 Object Type Definitions

Start of informative comment:

Definition of object types that can be used with the method Tspi_Context_CreateObject( ).

End of informative comment.

The defined object types are based on the data type TSS_FLAG.

Object Type	Description
TSS_OBJECT_TYPE_POLICY	Policy object.
TSS_OBJECT_TYPE_RSAKEY	RSAKey object.
TSS_OBJECT_TYPE_ENCDATA	Encrypted data object; sealed data or bound data.
TSS_OBJECT_TYPE_PCRS	PCR composite object.
TSS_OBJECT_TYPE_HASH	Hash object.

2.3.2.2 Object Initialization Definitions

Start of informative comment:

Definition of object initialization flags that can be used with the method Tspi_Context_CreateObject( ).

End of informative comment.

The defined initialization flags are based on the data type TSS_FLAG.

InitFlag	Description
TSS_KEY_SIZE_DEFAULT	Default size (see remarks below)
TSS_KEY_SIZE_512	Key size 512.
TSS_KEY_SIZE_1024	Key size 1024.
TSS_KEY_SIZE_2048	Key size 2048.
TSS_KEY_SIZE_4096	Key size 4096.
TSS_KEY_SIZE_8192	Key size 8192.
TSS_KEY_SIZE_16384	Key size 16384.
TSS_KEY_TYPE_STORAGE	Key for wrapping keys.
TSS_KEY_TYPE_SIGNING	Key for signing operations.
TSS_KEY_TYPE_BIND	Binding Key.
TSS_KEY_TYPE_IDENTITY	Key for an identity.
TSS_KEY_TYPE_LEGACY	Key that can perform signing and binding.
TSS_KEY_TYPE_AUTHCHANGE	An ephemeral key used to change authorization value
TSS_KEY_NON_VOLATILE	Key is non-volatile. MAY be unloaded at startup
TSS_KEY_VOLATILE	Key is volatile. MUST be unloaded at startup
TSS_KEY_NOT_MIGRATABLE	Key is not migratable (DEFAULT).
TSS_KEY_MIGRATABLE	Key is migratable.
TSS_KEY_NO_AUTHORIZATION	Key needs no authorization (DEFAULT).
TSS_KEY_AUTHORIZATION	Key needs authorization.
TSS_ENCDATA_SEAL	Data object is used for seal operation.

TSS_ENCDATA_BIND	Data object is used for bind operation.
TSS_ENCDATA_LEGACY	Data for legacy bind operation.
TSS_HASH_DEFAULT	Default hash algorithm
TSS_HASH_SHA1	Hash object with algorithm SHA1
TSS_HASH_OTHER	Hash object with other algorithm
TSS_POLICY_USAGE	Policy object used for authorization
TSS_POLICY_MIGRATION	Policy object used for migration
TSS_KEY_EMPTY_KEY	no TCG key template (empty TSP key object)
TSS_KEY_TSP_SRK	use a TCG SRK template (TSP key object for SRK)

Remarks:

TSS_KEY_SIZE_DEFAULT: This is not a fix defined key size. The key size is internally queried from the TSS Core Service running on the TCG system.

2.3.2.3 Attribute Definitions for a Context Object

Attribute flags.

TSS_TSPATTRIB_CONTEXT_SILENT_MODE	Get/set the silent mode of a context object
TSS_TSPATTRIB_CONTEXT_MACHINE_NAME	Get the machine name of the TSS given as a zero terminated UNICODE string the context object is connected with.

Attribute values

TSS_TSPATTRIB_CONTEXT_NOT_SILENT	TSP dialogs are shown asking a user for a secret (Default).
TSS_TSPATTRIB_CONTEXT_SILENT	TSP dialogs are not shown.

Remarks:

The application requests that the TSS Service Provider does not display any user interface (UI) for this context. If the TSP must display a UI to operate, the call fails and the TSS_E_SILENT_CONTEXT error code is returned. TSS_TSPATTRIB_CONTEXT_SILENT is intended for use with applications for which the UI cannot be displayed by the TSP

2.3.2.4 Attribute Definitions for a TPM Object

Attribute flags.

TSS_TSPATTRIB_TPM_CALLBACK_COLLATEIDENTITY	Get/Set the the address of the callback function to be used
TSS_TSPATTRIB_TPM_CALLBACK_ACTIVATEIDENTITY	Get/Set the the address of the callback function to be used

2.3.2.5 Attribute Definitions for a Policy Object

Attribute flags.

TSS_TSPATTRIB_POLICY_CALLBACK_HMAC	Get/Set the address of the callback function to be

	used.
TSS_TSPATTRIB_POLICY_CALLBACK_XOR_ENC	Get/Set the address of the callback function to be used.
TSS_TSPATTRIB_POLICY_CALLBACK_TAKEOWNERSHIP	Get/Set the the address of the callback function to be used.
TSS_TSPATTRIB_POLICY_CALLBACK_CHANGEAUHTASYM	Get/Set the the address of the callback function to be used.
TSS_TSPATTRIB_POLICY_SECRET_LIFETIME	Get/Set the lifetime of a secret.
TSS_TSPATTRIB_POLICY_POPUPSTRING	Set a NULL terminated UNICODE string which is displayed in the TSP policy popup dialog.

Sub-attribute flags

TSS_TSPATTRIB_POLSECRET_LIFETIME_ALWAYS	Secret will not be invalidated.
TSS_TSPATTRIB_POLSECRET_LIFETIME_COUNTER	Secret may be used n-times.
TSS_TSPATTRIB_POLSECRET_LIFETIME_TIMER	Secret will be valid for n seconds.

2.3.2.6 Attribute Definitions for a Key Object

Attribute flags.

TSS_TSPATTRIB_KEY_REGISTER	Set/Get the persistent storage the key is registered in
TSS_TSPATTRIB_KEY_BLOB	Get/Set a key blob
TSS_TSPATTRIB_KEY_INFO	Get key information
TSS_TSPATTRIB_KEY_UUID	Get TSS_UUID structure containing the UUID the key is assigned to.
TSS_TSPATTRIB_KEY_PCR	Get PCR information the key is sealed to
TSS_TSPATTRIB_RSAKEY_INFO	Get exponent/modulus info from a RSA key

Attribute Values

TSS_TSPATTRIB_KEYREGISTER_USER	Key is registered automatically in the persistent storage of TSP.
TSS_TSPATTRIB_KEYREGISTER_SYTEM	Key is registered automatically in persistent storage of TCS.
TSS_TSPATTRIB_KEYREGISTER_NO	Key is not registered in persistent storage.
TSS_TSPATTRIB_KEYBLOB_BLOB	Key information as a key blob.
TSS_TSPATTRIB_KEYBLOB_PUBLIC_KEY	Public key information as public key blob.
TSS_TSPATTRIB_KEYBLOB_PRIVATE_KEY	Encrypted private key information as private key blob.
TSS_TSPATTRIB_KEYINFO_SIZE	Key size in bits
TSS_TSPATTRIB_KEYINFO_USAGE	Key usage info

TSS_TSPATTRIB_KEYINFO_KEYFLAGS	Key flags
TSS_TSPATTRIB_KEYINFO_AUTHUSAGE	Key auth usage info
TSS_TSPATTRIB_KEYINFO_ALGORITHM	Key algorithm ID
TSS_TSPATTRIB_KEYINFO_SIGSCHEME	Key sig scheme
TSS_TSPATTRIB_KEYINFO_ENCSCHEME	key enc scheme
TSS_TSPATTRIB_KEYINFO_MIGRATABLE	if true then key is migratable
TSS_TSPATTRIB_KEYINFO_REDIRECTED	key is redirected
TSS_TSPATTRIB_KEYINFO_VOLATILE	if true key is volatile
TSS_TSPATTRIB_KEYINFO_AUTHDATAUSAGE	if true authorization is required
TSS_TSPATTRIB_KEYINFO_VERSION	version info as TSS version struct
TSS_TSPATTRIB_KEYINFO_RSA_EXPONENT	Exponent of the key
TSS_TSPATTRIB_KEYINFO_RSA_MODULUS	Modulus of the key
TSS_TSPATTRIB_KEYINFO_RSA_PRIMES	Primes of the key
TSS_TSPATTRIB_KEYINFO_RSA_KEYSIZE	Size of the key
TSS_TSPATTRIB_KEYPCR_DIGEST_ATCREATION	Get composite digest value of the PCR values, at the time when the sealing was performed.
TSS_TSPATTRIB_KEYPCR_DIGEST_ATRELEASE	This is the digest of the PCR value to verify when revealing sealed data.
TSS_TSPATTRIB_KEYPCR_SELECTION	This is the selction of PCRs to which the key is bound.

2.3.2.7 Attribute Definitions for a Data Object

Attribute flags.

TSS_TSPATTRIB_ENCDATA_BLOB	Get/Set a data blob
TSS_TSPATTRIB_ENCDATA_PCR	Get PCR information the data is sealed to

Sub-attribute flags

TSS_TSPATTRIB_ENCDATABLOB_BLOB	Data blob that represents the encrypted data depending on its type (seal or bind).
TSS_TSPATTRIB_ENCDATAPCR_DIGEST_ATCREATION	Get composite digest value of the PCR values, at the time when the sealing was performed.

2.3.2.8 Policy Definitions for Secret Mode

Start of informative comment:

Definition of policy mode flags that can be used with the method Tspi_Policy_SetSecret( ).

End of informative comment.

The defined secret modes are based on the data type TSS_FLAG.

Secret Mode	Description
TSS_SECRET_MODE_NONE	No authorization will be processed; different from secret of 20 bytes of 0x00.
TSS_SECRET_MODE_SHA1	Secret string will not be touched by TSS SP and MUST be size of 20 bytes.

Copyright TCG 2003

TSS_SECRET_MODE_PLAIN	Secret string will be hashed using SHA1.
TSS_SECRET_MODE_POPUP	TSS SP will ask for a secret. The provided pass phrase MUST be represented as a UNICODE string and MUST be hashed using SHA1 to get the authorization secret.
TSS_SECRET_MODE_CALLBACK	Application has to provide a call back function.

2.3.2.9 Policy Definition for Secret Lifetime

Start of informative comment:

Definition of secret lifetime flags that can be used with the method Tspi_SetAttribUint32( ) and Tspi_GetAttribUint32( ) addressing a policy object.

End of informative comment.

The defined secret modes are based on the data type TSS_FLAG.

TSS_SECRET_LIFETIME_ALWAYS	Secret will not be invalidated.
TSS_SECRET_LIFETIME_COUNTER	Secret may be used n-times.
TSS_SECRET_LIFETIME_TIMER	Secret will be valid for n seconds.

2.3.2.10 TPM Status Flags Definitions

Start of informative comment:

This flags are used to set the TPM status by calling the methods Tspi_TPM_SetStatus( ) and Tspi_TPM_GetStatus( ).

End of informative comment.

Flag	Description
TSS_TPMSTATUS_DISABLEOWNERCLEAR	Permanently disable the TPM owner authorized clearing of TPM ownership. The method Tspi_TPM_ClearOwner( ) with fForcedClear = FALSE is not available any longer.
TSS_TPMSTATUS_DISABLEFORCECLEAR	Prevent temporarily (until next power on) a forced clear of the TPM ownership. The method Tspi_TPM_ClearOwner( ) with fForcedClear = TRUE is temporarily not available.
TSS_TPMSTATUS_OWNERSETDISABLE	fTpmState = TRUE: Disable the TPM. Owner authorization is required.
TSS_TPMSTATUS_PHYSICALDISABLE	fTpmState = TRUE: Disable the TPM. Proof of physical access is required.
TSS_TPMSTATUS_PHYSICALSETDEACTIVATED	fTpmState = TRUE: Deactivate the TPM. Proof of physical access is required
TSS_TPMSTATUS_SETTEMPDEACTIVATED	Temporarily deactivate (until next power on) the TPM.
TSS_TPMSTATUS_SETOWNERINSTALL	fTpmState = TRUE: Set the ability

	to take TPM ownwership utilizing the method Tspi_TPM_TakeOwnership( ).
TSS_TPMSTATUS_DISABLEPUBEKREAD	Permanently disable the ability to read the endorsement public key without required TPM owner authorizition. The method Tspi_TPM_GetPubEndorsementKey ( ) with fOwnerAuthorized = FALSE is not available any longer.
TSS_TPMSTATUS_DISABLED	Query whether TPM is disabled or enabled.
TSS_TPMSTATUS_DEACTIVATED	Query whether the TPM is deactivated or activated.
TSS_TPMSTATUS_ALLOWMAINTENANCE	Query whether the TPM owner may create a maintenance archive utilizing the method Tspi_TPM_CreateMaintenanceArchive( ) or not.
TSS_TPMSTATUS_PHYSPRES_LIFTIMELOCK	Query whether both physicalPresenceHWEnable and physicalPresenceCMDEnable flags are locked and cannot be changed for the life of the TPM.
TSS_TPMSTATUS_PHYSPRES_HWENABLE	Query whether the TPM hardware signal <physical presence> is enabled to provide proof of physical presence.
TSS_TPMSTATUS_PHYSPRES_CMDENABLE	Query whether the TPM command TSC_PhysicalPresence is enabled to provide proof of physical presence.
TSS_TPMSTATUS_CEKP_USED	Query whether the endorsement key pair was created using the methode Tspi_TPM_CreateEndorsementKey( ) or it was created using a manufacturers process.
TSS_TPMSTATUS_PHYSPRESENCE	Query whether a TPM owner is present indicated by the TPM command TSC_PhysicalPresence.
TSS_TPMSTATUS_PHYSPRES_LOCK	Query whether changes to the physicalPresence flag are permitted.
TSS_TPMSTATUS_POSTINITIALISE	Indicates that the TPM is between the TPM_Init state and the execution of the TPM_Startup command.
TSS_TPMSTATUS_TPMPOST	Sets the TPM to force a full sefttest before allowing commands to be performed.
TSS_TPMSTATUS_TPMPOSTLOCK	Locks the state of the TSS_TPMSTATUS_TPMPOST flag for the lifetime of the TPM

Remarks:

Please see the manual of your TCG system, to set the physical access state.

If the TPM status is set to *DISABLED* only the following TSPI methods will execute. All other methods will return the TPM error TCPA_DISABLED.

- Tspi_TPM_GetCapability
- Tspi_TPM_PcrExtend
- Tspi_TPM_SetStatus using the flag TSS_TPMSTATUS_OWNERSETDISABLE
- Tspi_TPM_SetStatus using the flag TSS_TPMSTATUS_PHYSICALDISABLE
- Tspi_TPM_SelfTestFull
- Tspi_TPM_GetTestResult

If the TPM status is set to *DEACTIVATED* only the following TSPI methods will execute. All other methods will return the TPM error TCPA_DEACTIVATED.

- Tspi_TPM_GetCapability
- Tspi_TPM_TakeOwnership
- Tspi_TPM_SetStatus using the flag TSS_TPMSTATUS_OWNERSETDISABLE
- Tspi_TPM_SetStatus using the flag TSS_TPMSTATUS_PHYSICALDISABLE
- Tspi_TPM_SetStatus using the flag TSS_TPMSTATUS_PHYSICALSETDEACTIVATED
- Tspi_TPM_SelfTestFull
- Tspi_TPM_GetTestResult

2.3.2.11 Algorithm ID Definitions

Start of informative comment:

This table defines the types of algorithms which may be supported.

End of informative comment.

The defined algorithm IDs are based on the data type TSS_ALGORITHM_ID.

Algorithm ID	Description
TSS_ALG_RSA	The RSA algorithm.
TSS_ALG_DES	The DES algorithm.
TSS_ALG_3DES	The 3DES algorithm.
TSS_ALG_SHA	The SHA1 algorithm.
TSS_ALG_HMAC	The RFC 2104 HMAC algorithm.
TSS_ALG_AES	The AES algorithm.

Remarks:

The TPM must support the algorithms TSS_ALG_RSA, TSS_ALG_SHA, TSS_ALG_HMAC.

2.3.2.12 Capability Flag Definitions

Start of informative comment:

Flags indicating a capability to be queried

End of informative comment.

The defined capability flags are based on the data type TSS_FLAG.

TPM Capabilities:

Capabiltiy Area	Description
TSS_TPMCAP_ORD	Queries whether an ordinal is supported

Capabiltiy Area	Description
TSS_TPMCAP_ALG	Queries whether an algorithm is supported.
TSS_TPMCAP_FLAG	Queries the state of a flag
TSS_TPMCAP_PROPERTY	Determines a physical property of the TPM.
TSS_TPMCAP_VERSION	Queries the current TPM version.

TSS Core Service Capabilities:

Capabiltiy Area	Description
TSS_TCSCAP_ALG	Queries whether an algorithm is supported.
TSS_TCSCAP_VERSION	Queries the current TCS version.
TSS_TCSCAP_MANUFACTURER	Queries TCS manufacturer information.
TSS_TCSCAP_CACHING	Queries the support of key and authorization caching.
TSS_TCSCAP_PERSSTORAGE	Queries the support of a persistant storage

TSS Service Provider Capabilities:

Capabiltiy Area	Description
TSS_TSPCAP_ALG	Queries whether an algorithm is supported.
TSS_TSPCAP_VERSION	Queries the current TSP version.
TSS_TSPCAP_PERSSTORAGE	Queries the support of a persistant storage
TSS_TSPCAP_COLLATE_ALG	Queries whether the algorith is supported by the currently registered Tspicb_CollateIdentity function.

2.3.2.13 Sub-Capability Flag Definitions

Start of informative comment:

Sub-Flags indicating a capability to be queried dependent on the capability flag

End of informative comment.

The defined sub-capability flags for capability TSS_TPMCAP_PROPERTY.

TPM Sub-Capabilities:

SubCap Area	Response
TSS_TPMCAP_PROP_PCR	UINT32 value. Returns the number of PCR registers supported by the TPM
TSS_TPMCAP_PROP_DIR	UINT32 value. Returns the number of DIR registers supported by the TPM.
TSS_TPMCAP_PROP_MANUFACTURER	UINT32 value. Returns the Identifier of the TPM manufacturer.
TSS_TPMCAP_PROP_SLOTS	UINT32 value. Returns the maximum number of 2048 bit RSA keys that the TPM is capable of loading. This MAY vary with time and circumstances.

TSS Core Service Sub Capabilities:

The defined sub-capability flags for capability TSS_TCSCAP_MANUFACTURER

SubCap Area	Description
TSS_TCSCAP_PROP_MANUFACTUR	Returns an Unicode string of the TCS

SubCap Area	Description
ER_STR	manufacturer. The contents of this string is determined by the manufacturer and is subject to change in subsequent releases of the TCS.
TSS_TCSCAP_PROP_MANUFACTURER_ID	Returns the UINT32 value which identifies the TCS manufacturer as specified in the main specification by: Capability: TPM_CAP_PROPERTY; Sub-Capability: TPM_CAP_PROP_MANUFACTURER

The defined sub-capability flags for capability TSS_TCSCAP_CACHING

SubCap Area	Description
TSS_TCSCAP_PROP_KEYCACHE	TSS_BOOL value. Indicates support of key caching
TSS_TCSCAP_PROP_AUTHCACHE	TSS_BOOL value. Indicates support of authorization session caching

TSS Service Provider Sub Capabilities:

SubCap Area	Description
[None yet defined]	

2.3.2.14 Persistent Storage Flag Definitions

Start of informative comment:

Definition of flags indicating the persistent storage to be used within the method Tspi_Context_RegisterKey( ).

End of informative comment.

The defined persistent storage flags are based on the data type TSS_FLAG.

Persistent Storage Type	Description
TSS_PS_TYPE_USER	Key is registered persistantly in the user storage database.
TSS_PS_TYPE_SYSTEM	Key is registered persistantly in the system storage database.

2.3.2.15 Migration Scheme Definitions

Start of informative comment:

The scheme indicates how the migration of a key should be done

End of informative comment.

The defined migration scheme flags are based on the data type TSS_MIGRATE_SCHEME.

Migration Scheme	Description
TSS_MS_MIGRATE	A public key that can be used for migrating a key utilizing Tspi_Key_CreateMigrationBlob followed by Tspi_Key_ConvertMigrationBlob.
TSS_MS_REWRAP	A public key that can be used for migrating a key by just rewrapping this key utilizing Tspi_Key_CreateMigrationBlob.

Copyright TCG 2003

TSS_MS_MAINT	A public key that can be used for the maintenance commands.

2.3.2.16 Key Usage Definitions

Start of informative comment:

This table defines the types of keys that are possible.

Each key has a setting defining the encryption and signature scheme to use. The selection of a key usage value limits the choices of encryption and signature schemes.

End of informative comment.

The defined algorithm IDs are based on the data type TSS_KEY_USAGE_ID.

Key Usage	Description
TSS_KEYUSAGE_BIND	The key can be used for binding and unbinding operations only.
TSS_KEYUSAGE_IDENTITY	The key is used for operations that require a TPM identity, only.
TSS_KEYUSAGE_LEGACY	The key can perform signing and binding operations.
TSS_KEYUSAGE_SIGN	The [private] key is used for signing operations, only. This means that it MUST be a leaf of the Protected Storage key hierarchy
TSS_KEYUSAGE_STORAGE	The key is used to wrap and unwrap other keys in the Protected Storage hierarchy, only.
TSS_KEYUSAGE_AUTHCHANGE	The key is used to change authorization

Remarks:

2.3.2.17 Key Size Definitions

Start of informative comment:

This table defines the key sizes returned by Tspi_GetAttribUint32(TSS_TSPATTRIB_KEY_INFO, TSS_TSPATTRIB_KEYINFO_SIZE).

End of informative comment.

The defined key sizes as returned by Tspi_GetAttribUint32(TSS_TSPATTRIB_KEY_INFO, TSS_TSPATTRIB_KEYINFO_SIZE).

Key Size	Description
TSS_KEY_SIZEVAL_512BIT	key size is 512 bit
TSS_KEY_SIZEVAL_1024BIT	key size is 1024 bit
TSS_KEY_SIZEVAL_2048BIT	key size is 2048 bit
TSS_KEY_SIZEVAL_4096IT	key size is 4096 bit
TSS_KEY_SIZEVAL_8192BIT	key size is 8192 bit
TSS_KEY_SIZEVAL_16384BIT	key size is 16384 bit

Remarks:

2.3.2.18 Key Type Flags

Start of informative comment:

This table defines the key type returned by Tspi_GetAttribUint32(TSS_TSPATTRIB_KEY_INFO, TSS_TSPATTRIB_KEYINFO_MIGRATABLE), Tspi_GetAttribUint32(TSS_TSPATTRIB_KEY_INFO, TSS_TSPATTRIB_KEYINFO_REDIRECTED), or Tspi_GetAttribUint32(TSS_TSPATTRIB_KEY_INFO, TSS_TSPATTRIB_KEYINFO_VOLATILE)

End of informative comment.

This table defines the key type returned by:

Tspi_GetAttribUint32(TSS_TSPATTRIB_KEY_INFO, TSS_TSPATTRIB_KEYINFO_MIGRATABLE), Tspi_GetAttribUint32(TSS_TSPATTRIB_KEY_INFO, TSS_TSPATTRIB_KEYINFO_REDIRECTED), Tspi_GetAttribUint32(TSS_TSPATTRIB_KEY_INFO, TSS_TSPATTRIB_KEYINFO_VOLATILE)

Key Type Flag	Description
TSS_KEYFLAG_REDIRECTION	set to 1 if redirection key
TSS_KEYFLAG_MIGRATABLE	set to 1 if mirgratable key
TSS_KEYFLAG_VOLATILEKEY	set to 1 if volatile key

Remarks:

2.3.2.19 Key Authorization

Start of informative comment:

This table defines the key sizes returned by Tspi_GetAttribUint32(TSS_TSPATTRIB_KEY_INFO, TSS_TSPATTRIB_KEYINFO_AUTHDATAUSAGE).

End of informative comment.

This table defines the key sizes returned by Tspi_GetAttribUint32(TSS_TSPATTRIB_KEY_INFO, TSS_TSPATTRIB_KEYINFO_AUTHDATAUSAGE).

Key Authorization	Description
TSS_KEYAUTH_AUTH_NEVER	Key never requires authorization
TSS_KEYAUTH_AUTH_ALWAYS	Key always requires authorization

Remarks:

2.3.2.20 Key Encryption Scheme Definitions

Start of informative comment:

The TPM performs the encryption or decryption in accordance with the specification of the encryption scheme to be used for a key.

End of informative comment.

The defined encryption scheme IDs are based on the data type TSS_KEY_ENC_SCHEME.

Algorithm ID	Description
TSS_ES_NONE	No encryption scheme is set.
TSS_ES_RSAESPKCSV15	The encryption is performed using the scheme RSA_ES_PKCSV15 defined in [PKCS #1v2.0: 8.1].
TSS_ES_RSAESOAEP_SHA1_MGF1	The encryption and decryption is performed

Copyright TCG 2003

	using the scheme RSA_ES_OAEP defined in [PKCS #1v2.0: 8.1] using SHA1 as the hash algorithm for the encoding operation.

Remarks:

The TPM checks that the encryption scheme defined for use with the key is a valid scheme for the key type, as follows:

Key algorithm	Approved schemes
TSS_ALG_RSA	TSS_ES_NONE
	TSS_ES_RSAESPKCSv15
	TSS_ES_RSAESOAEP_SHA1_MGF1

2.3.2.21 Key Signature Scheme Definitions

Start of informative comment:

The TPM performs the digital signatures in accordance with the specification of the signature scheme to be used for a key.

End of informative comment.

The defined signature scheme IDs are based on the data type TSS_KEY_SIG_SCHEME.

Algorithm ID	Description
TSS_SS_NONE	
TSS_SS_RSASSAPKCS1V15_SHA1	The signature is be performed using the scheme RSASSA-PKCS1-v1.5 defined in [PKCS #1v2.0: 8.1] using SHA1 as the hash algorithm for the encoding operation.
TSS_SS_RSASSAPKCS1V15_DER	The signature is performed using the scheme RSASSA-PKCS1-v1.5 defined in [PKCS #1v2.0: 8.1]. The caller must properly format the area to sign using the DER rules. The provided area maximum size is k-11 octets

Remarks:

The TPM checks that the signature scheme defined for use with the key is a valid scheme for the key type, as follows:

Key algorithm	Approved schemes
TSS_ALG_RSA	TSS_ES_NONE
	TSS_SS_RSASSAPKCS1V15_SHA1
	TSS_SS_RSASSAPKCS1V15_DER

2.3.2.22 Event Type Definitions

Start of informative comment:

Flags indicating the type of event/supporting information.

End of informative comment.

The defined event type flags are based on the data type TSS_FLAG.

Event Type	Comments
TSS_EV_CODE_CERT	The event is in response to loading a firmware

	or software component for which a VE certificate was available. *rgbEvent* points to the VE certificate that shipped with the platform firmware or software (or discovered by other means). *PcrValue* is the digest of the firmware, software or other code loaded. Certificates are much too large to put into the log in the Pre-OS environment. Validation of Certificates is unlikely in the Pre-OS environment. The event points to a TSS_EVENT_CERT structure.
TSS_EV_CODE_NOCERT	The event is in response to loading a firmware or other software component, but no VE certificate was found. *ulEventLenght* is 0 and *rgbEvent* is unused. However, *PcrValue* is the digest of the firmware discovered. Absence of a VE certificate does not indicate lack of trust; it merely indicates that a VE certificate was not available at this point in boot. Upper-level software may be able to obtain such certificates.
TSS_EV_XML_CONFIG	The event describes the platform configuration. The supporting information is a platform or firmware-defined XML data structure that indicates security-relevant hardware configuration information. The event logged to the PCR is the SHA-1 digest of the XML data structure, and the firmware guarantees that the configuration stated in the data structure is in effect when the firmware relinquishes control to the next module in boot. Size is the size in bytes of the XML data structure, and *rgbEvent* points to the data structure itself. The information may include size of physical memory, number of processors, chipset configuration, buses discovered and processor/bus frequencies. Firmware vendors are free to define the XML reporting structure and select those parameters that are important for their platforms.
TSS_EV_NO_ACTION	The action was not performed. The corresponding DIGEST structure must be 0x1 (a single binary digit in the LSB of the DIGEST structure), and this value is logged to the PCR. A supporting data structure may be supplied containing information that describes why the event did not occur. If such supporting information is supplied, it should be well-formed XML. However, this supporting information is not required.
TSS_EV_SEPARATOR	A list of actions was complete. This event must be used if more than one event can be logged to the TPM and upper-level software needs to be informed that logging was completed.

Copyright TCG 2003

TSS_EV_ACTION	A logged event. This is a zero terminated UNICODE string with the content defined by the Platform Specific specifications.
TSS_EV_PLATFORM_SPECIFIC	Implementation specification defined data.
Reserved up to 2^{16}-1	TCG-reserved event types
User-definable $2^{16} - (2^{32} - 1)$	Undefined and free for general-purpose use

For the parameters *rgbEvent* etc. mentioned in the table refer to 2.5.2

Remarks:

Additional event types may be defined for TCG usage in specific computing platforms (for example, the PC).

2.3.2.23 Well Known Secret

Start of informative comment:

This is simply a “helper” define for those applications were the well know secret is defined as all zeros. Note, there is no required value for this field.

End of informative comment.

This value is only used for convenience for a “well known secret” for authentication data. The actual value of its definition is not mandated.

Well Known Secret	Description
TSS_WELL_KNOWN_SECRET	A value used for convenience as a “well known value”

Remarks:

2.4 Return Codes

Start of informative comment:

Return codes contain information divided into components: OS error code, TSS layer field and TSS error code. The OS error code is specific to the OS and the services that the TSS may rely upon. The contents of this field is OS-specific and details outside the scope of this specification and applications should only assume they are valid if they are non-zero.

Each layer within the TSS should make a “best effort” to correct errors from lower layers. If the “best effort” fails, the layer must return the lower layer error that is relevant to the requested function and operation. For example, if a key is not loaded it is expected the TCS key manager will attempt to make room for a key and load it, all transparently to the layers above it. However, if the TCS key manager cannot load the needed key it must return the TPM error based on the last error received that is relevant to the currently requested function and object. To further refine the example, if the TPM does not have sufficient space to load a key needed for a TPM_Sign function. The TCS could just attempt to load the key are receive a TCPA_NOSPACE error but that would violate the “best effort” requirement unless the TPM did not implement the TPM_SaveKeyContext in which case the TCS would in fact return TCPA_NOSPACE error. If the TPM allowed saving the key context the TCS would be expected to perform this function to make room for the key. If an error occurred while attempting to load the key because the unwrapping key was not loaded the TCS would be expected to attempt to load the unwrapping key. If the unwrapping key could not be loaded the TCS would return TCPA_NOSPACE error code because that error, not the failure to load the parent’s key, is the most relevant to the requested function and the object.

The coding system should specify room for platform specific requirements and extensions. It should also provide an opportunity to integrate vendor specific return codes in each layer and allow the possibility to specific common return codes over all layers.

End of informative comment.

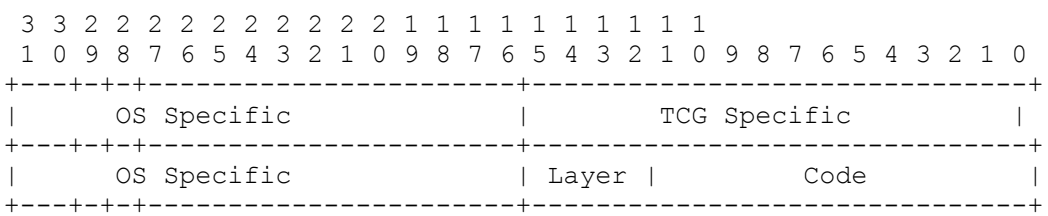
There are three components of each return code: OS code, TSS layer, TSS code.

- The OS code contains information regarding the OS-specific error. This details of this code are outside the scope of this specification and this field is not required. This field may not be used for anything other than the platform defined fields.
- The TSS layer field identifies the layer originating the error.
- The TSS code identifies the specific cause or condition of the error.

Each layer within the TSS SHOULD make a “best effort” to correct errors from lower layers. If the “best effort” fails, the layer must return the lower layer error that is relevant to the requested function and operation.

2.4.1 Return Codes Scheme

A TSS_RESULT is defined as a 32-bit value as follows:



OS specific error information:

The bits #16 to #31 are specific to the operating system. More information please find below.

TSS specific error information:

The bits #0 up to #15 are TSS specific and subdivided in two parts:

- Layer information and
- Error code information

Platform Specific errors:

Any platform specific error return codes must not be returned by any TSS SW stack component; this means that a TSS SW stack component only returns error codes applying to the above described rules and must hide any platform specific error return codes and therefore must map these error codes to TSS specific error codes. (e.g. internal TSP error TSS_E_INTERNAL_ERROR)

Layer Information:

Bit #12 to Bit #15 are specifying the layer of the TSS SW stack returning the error.

Each TSS SW stack component (TDDL, TCS and TSP) must return an error code either with the layer nibble set to its own layer information or with the TPM layer information. The latter will be returned if an error was encountered by the TPM.

The layer code MUST be chosen from the following list.

Layer	Value	Description
TPM	0x0	Error returned by TPM
TDDL	0x1	Error returned by TDDL
TCS	0x2	Error returned by TCS
TSP	0x3	Error returned by TSP

Error Code Information:

Bit #0 to Bit #11 are reporting the appropriate TSS specific error code.

2.4.2 Common Return Code Defines

With the following table the error codes (Bit #0 to Bit #11 reporting the appropriate TCG specific error code) common to all TSS Layers are listed.

Type	Definition
TSS_SUCCESS	Success
TSS_E_FAIL	Non-specific failure
TSS_E_BAD_PARAMETER	One or more parameter is bad.
TSS_E_INTERNAL_ERROR	An internal SW error has been detected.
TSS_E_NOTIMPL	Not implemented.
TSS_E_PS_KEY_NOTFOUND	The key cannot be found in the persistent storage database.
TSS_E_KEY_ALREADY_REGISTERED	Key could not be registered because UUID has already registered.
TSS_E_CANCELED	The action was canceled by requestor.
TSS_E_TIMEOUT	The operation has timed out.
TSS_E_OUTOFMEMORY	Ran out of memory.
TSS_E_TPM_UNEXPECTED	TPM returns with success but TSP/TCS notice that something is wrong.
TSS_E_COMM_FAILURE	A communications error with the TPM has been detected.

TSS_E_TPM_UNSUPPORTED_FEATURE	The TPM does not support the requested feature.

2.4.3 Common Return Code Rules

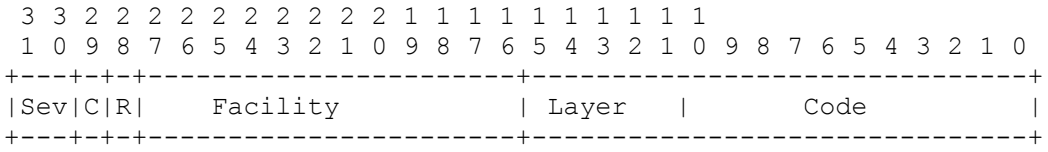
The above return codes may be returned by any function or method from any TSS layer. Other error MUST be explicitly stated within the respective function or specified in the layer-specific return code rules sections below.

2.4.4 OS Specific Error Information:

2.4.4.1 Windows Operating System:

Note: the Sev, S, C, N, R, and Facility fields are defined by the Windows environment.

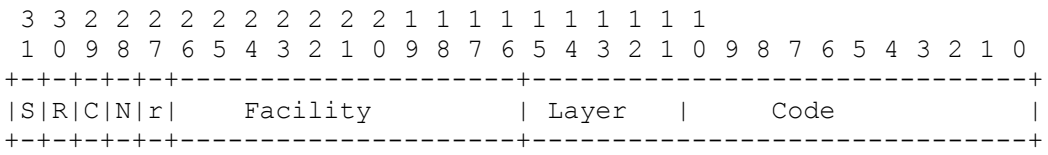
CAPI Error Codes:



Type	Description
Sev	Severity Code. 00 – Success 01 – Informational 10 – Warning 11 = Error
C	Customer Code Flag. (1)
R	Reserved Bit. (0)
Facility	Facility Code. (0x028)
Layer	TSS layer information
Code	TSS error code value

COM Error Codes:

HRESULTs are 32 bit values layed out as follows:



Type	Description
S (Severity)	Indicates Success or Fail. 0 - Success. 1 - Fail (COERROR).
R	Reserved portion of the facility code, corresponds to NT's second severity bit. (1)
C	Reserved portion of the facility code, corresponds to NT's C field. (1)
N	Reserved portion of the facility code. Used to indicate a mapped NT status value. (0)

R	Reserved portion of the facility code. Reserved for internal use. Used to indicate HRESULT values that are not status values, but are instead message IDs for display strings. (0)
Facility	Facility Code. (0x028)
Layer	TSS layer information
Code	TSS error code value

2.4.4.2 Linux

This section is undefined.

2.4.5 ErrorSample

Start of Example:

Samples usage of error codes.

```
int main(int argc, char* argv[])
{
	TSS_RESULT dwTssRetCode = TSS_E_UNKNOWN_ERROR;
	/* pseudo code section to get the TPM object at the TSP global context handle */
	do
	{
		TSS_HTPM	hTPM = NULL;
		BYTE	*pRandomData = NULL;
		// hContext is the global TSP-Context handle
		dwTssRetCode = Tspi_Context_GetTPMObject(hContext, &hTPM);

		// test the return code of the GetTPMObject function (TSS-ReturnCode)
		if (dwTssRetCode == TSS_E_BAD_PARAMETER)
		{
			// bad parameter for the GetTPMObject call
			// at the TSP interface
			// e.g. hContext not set
			break;
		}
		else
			if (dwTssRetCode != TSS_SUCCESS);
				break;
		// call the GetRandom method at the TPM object of the current context
		dwTssRetCode = Tspi_TPM_GetRandom(hTPM, 64, &pRandomData);

		if (dwTssRetCode == TPM_E_DISABLED_CMD)    (TPM-ReturnCode)
		{
			// GetRandom command not enabled at the TPM device
			// start enable sequence for the device
		}
		else
			if (dwTssRetCode != TSS_SUCCESS);
				break;
	} while(FALSE);
```

```
    // free local resources and...
    return dwTssRetCode;
}
```

End of example:

2.5 Structures

2.5.1 TSS_VERSION

Start of informative comment:

This structure allows the TSS Service Provider to communicate with outside entities as to the version of the TPM, TSS Core Service or TSS ServiceProvider.

End of informative comment.

Definition:

```
typedef struct tdTSS_VERSION
{
   BYTE  bMajor;
   BYTE  bMinor;
   BYTE  bRevMajor;
   BYTE  bRevMinor;
} TSS_VERSION;
```

Parameters:

bMajor
This SHALL be the major version indicator for this implementation of the TSS specification. For version 1 this must be 0x01

bMinor
This SHALL be the minor version indicator for this implementation of the TSS specification. For version 1 this must be 0x01.

bRevMajor
This SHALL be the major value of the TSS vendor's implementation. The value of this is left to the TSS vendor to determine.

bRevMinor
This SHALL be the minor value of the TSS vendor's implementation. The value of this is left to the TSS vendor to determine.

Remarks:

The version points to the version of the specification that defines the structure or a service.
If the validity of a structure depends on conformity to a version of the specification and/or to a version of the TSS, that structure includes the current instance of TSS_VERSION

Copyright TCG 2003

2.5.2 TSS_PCR_EVENT

Start of informative comment:

This structure provides information about an individual PCR extend event.

End of informative comment.

Definition:

```
typedef struct tdTSS_PCR_EVENT
{
   TSS_VERSION    versionInfo;
   UINT32         ulPcrIndex;
   TSS_EVENTTYPE  eventType;
   UINT32         ulPcrValueLength;
   #ifdef __midl
   [size_is(ulPcrValueLength)]
   #endif
   BYTE* rgbPcrValue;
   UINT32         ulEventLength;
   #ifdef __midl
   [size_is(ulEventLength)]
   #endif
   BYTE* rgbEvent;
} TSS_PCR_EVENT;
```

Parameters:

versionInfo
Version data.

ulPcrIndex
Index of the PCR this event belongs to.

eventType
Flag indicating the type of the event (see section 2.3.2.16 for definition).

ulPcrValueLength
The length (in bytes) of the *rgbPcrValue* parameter

rgbPcrValue
Pointer to memory containing the value extended into the TPM by Tspi_TPM_PcrExtend. This SHALL be the SHA-1 hash of the Length, PCRIndex, *Event, and EventType values. The hash of *Event SHALL be the data that *Event points to, not the *Event pointer.

ulEventLength
The length (in bytes) of the *rgbEvent* parameter

rgbEvent
Pointer to the event information data.

Remarks:

Copyright TCG 2003

2.5.3 TSS_EVENT_CERT

Start of informative comment:

Certificate structure to use for events of type *TSS_EV_CODE_CERT*.

End of informative comment.

Definition

```
typedef struct tdTSS_EVENT_CERT
{
   TSS_VERSION versionInfo;
   UINT32      ulCertificateHashLength
   BYTE*       rgbCertificateHash;
   UINT32      ulEntityDigestLength
   #ifdef __midl
         [size_is(ulEntityDigestLength)]
   #endif
   BYTE*       rgbentityDigest;
   TSS_BOOL    fDigestChecked;
   TSS_BOOL    fDigestVerified;
   UINT32      ulIssuerLength;
   #ifdef __midl
               [size_is(ulIssuerLength)]
   #endif
   BYTE*       rgbIssuer;
) TSS_EVENT_CERT;
```

Parameters

versionInfo
Version data.

ulCertificateHashLength
The length (in bytes) of the *rgbCertificatHash* parameter

rgbCertificateHash
Pointer to memory containing the hash value of the entire VE certificate

ulEntityDigestLength
The length (in bytes) of the *rgbEntityDigest* parameter

rgbEntityDigest
Pointer to memory containing the actual digest value of the entity

fDigestChecked
TRUE if the entity logging this event checked the measured value against the digest value in the certificate.
FALSE if no checking was attempted.

fDigestVerified
Only valid when *fDigestChecked* is TRUE.

TRUE if measured value matches digest value in certificate, FALSE otherwise.

ulIssuerLength
The length (in bytes) of the *rgbIssuer* parameter

rgbIssuer
Pointer to actual issuer certificate.

Copyright TCG 2003

2.5.4 TSS_UUID

Start of informative comment:

This structure provides information about an UUID identifier that is unique across both space and time. These UUIDs are used to register keys in the persistent storage of the TSS Key Manager. This is specified in accordance to IEEE 802.

End of informative comment.

Definition:

```
typedef struct tdTSS_UUID
{
   UINT32   ulTimeLow;
   UINT16   usTimeMid;
   UINT16   usTimeHigh;
   BYTE     bClockSeqHigh;
   BYTE     bClockSeqLow;
   BYTE     rgbNode[6];
} TSS_UUID;
```

Parameters

ulTimeLow
The low field of the timestamp.

usTimeMid
The middle field of the timestamp.

usTimeHigh
The high field of the timestamp multiplexed with the version number.

bClockSeqHigh
The high field of the clock sequence multiplexed with the variant.

bClockSeqLow
The low field of the clock sequence.

rgbNode
The spatially unique node identifier.

Remarks

Copyright TCG 2003

2.5.5 TSS_KM_KEYINFO

Start of informative comment:

The TSS_KM_KEYINFO structure provides information about a key registered in the TSS Persistent Storage.

End of informative comment.

Definition

```
typedef struct tdTSS_KM_KEYINFO
{
   TSS_VERSION    versionInfo;
   TSS_UUID       keyUUID;
   TSS_UUID       parentKeyUUID;
   BYTE           bAuthDataUsage;
   TSS_BOOL       fIsLoaded;          // TRUE: actually loaded in TPM
   UINT32         ulVendorDataLength; // may be 0
   #ifdef __midl
                  [size_is(ulVendorDataLength)]
   #endif
   BYTE* rgbVendorData;       // may be NULL
} TSS_KM_KEYINFO;
```

Parameters

versionInfo
Version data.

keyUUID
The UUID the key is registered in the persistent storage of the TSS Key Manager.

parentKeyUUID
The UUID the parent key which wraps the key addressed by *keyUUID* is registered in the persistent storage of the TSS Key Manger.

bAuthDataUsage
Flag indicating whether key usage requires authorization or not. Currently the values 0x00 and 0x01 are defined. The value 0x00 means usage of the key without authorization is permitted. The value 0x01 means that on each usage of the key the authorization must be performed. All other values are reserved for future use.

fIsLoaded
Flag indicating the key is loaded into the TPM.
TRUE: Key is loaded into the TPM.
FALSE: Key is not loaded into the TPM.

ulVendorDataLength
Supplies the length (in bytes) of the *rgbVendorData* parameter.
Set to 0 if this data is not of interest.

rgbVendorData
Pointer to vendor specific data.
Set to NULL if data is not of interest.

Copyright TCG 2003

2.5.6 TSS_VALIDATION

Start of informative comment:

The TSS_VALIDATION structure provides the ability to verify signatures and validation digests built over certain TPM command parameters. These parameters are returned as a byte stream and are defined within the TCPA 1.1b Main Specification. The caller must provide some random data (external Data value) as input, which is included in the signature/digest calculation.

The following functions are using this structure:

Tspi_TPM_CertifySelfTest,
Tspi_TPM_GetCapabilitySigned,
Tspi_TPM_LoadMaintenancePubKey,
Tspi_TPM_CheckMaintenancePubKey,
Tspi_Key_CertifyKey,
Tspi_TPM_CreateEndorsementKey,
Tspi_TPM_Quote

If the validation of the signature/digest should be done by the TSS Service Provider itself, a NULL pointer must be passed to these methods. In this case the TSS Service Provider generates its own random data to be included in the signature/digest (external Data value).

End of informative comment.

Definition

```
typedef struct tdTSS_VALIDATION
{
   TSS_VERSION versionInfo;
   UINT32      ulExternalDataLength,  // in
   #ifdef __midl
   [size_is(ulExternalDataLength)]
   #endif
   BYTE* rgbExternalData;  // in
   UINT32      ulDataLength;  // out
   #ifdef __midl
   [size_is(ulDataLength)]
   #endif
   BYTE* rgbData;    // out
   UINT32      ulValidationLength;  // out
   #ifdef __midl
   [size_is(ulValidationLength)]
   #endif
   BYTE* rgbValdationData; // out
} TSS_VALIDATION;
```

Parameters

versionInfo
Version data.

ulExternalData
The length (in bytes) of the *rgbExternalData* parameter

rgbExternalData
Pointer to memory containing the random data used to
- proof the signature for the caller of the signing method.
- avoid replay attacks.

ulDataLength

Copyright TCG 2003

Supplies the length (in bytes) of the *rgbData* parameter.

rgbData
Pointer to the data which was used to calculate the validation.

ulValidationLength
Supplies the length (in bytes) of the *rgbValidationData* parameter.

rgbValidationData
Pointer to the validation data.

2.6 Persistent Storage

Start of informative comment:

Any data can be rendered confidential through encryption and protection of the key used for encryption.

Similarly, any signing authority can be protected if the signing key is protected. A service that protects keys is therefore useful, and sometimes essential. Similarly, it is useful, and sometimes essential; to provide a service that protects authorization data.

The TCG Software Stack enables such a service because a TPM can act as a portal to keep arbitrary amounts of data and keys confidential. “Protected Storage” is a set of commands provided by the TPM to enable virtual secure storage space.

The Subsystem is required to offer persistent storage as a service to functions outside the TPM. This enables applications to provide functions such as User association, key archive, and key restoration, and enables the efficient migration of Subsystem information from one platform to another within a heterogeneous PC environment.

For a user application the persistent storage looks like a data archive, therefore the main function set is associated to such function sets.

End of informative comment.

2.7 Key Management

Start of informative comment:

The Key Management Services of TSS allow definition of a persistent key hierarchy.

The Key Management Services interface was designed to allow a flexible key structure such that an instance like an IT department of an enterprise may define a deep key hierarchy, a shallow hierarchy, roaming keys, migration base keys, etc.

All keys, which should be internally managed by the Key Management Services of TSS must be registered in the persistent storage database of TCS (*System Persistent Storage*) or TSP (*User Persistent Storage*). Each key registered in one of these databases will be referenced by its UUID and called a persistent key from this specification’s point of view.

Some registered keys have a defined fixed UUID by which they can be referenced on all systems providing the same registered key hierarchy. These UUIDs do not provide any information to identify the system the key is registered on.

An application can also load keys not registered in the TCS database. These keys are loaded utilizing the Tcsi by providing a key blob as defined by TCPA_KEY. These keys are called temporary keys from this specification’s point of view.

Using the Key Management Services provided by TSS will simplify the whole mechanism of loading a key into the TPM from a calling context’s point of view. The application must only address a key to be loaded by its well known UUID and the Key Management Services will do all the required loading of

Copyright TCG 2003

the underlying parent keys depending on the registered key hierarchy, which may be totally hidden from the application's scope.

The key hierarchy can be defined by some instance like for example the IT department of an enterprise and the TCG-aware applications may not need to know this key hierarchy at all.

Keys once registered in PS will stay registered in PS until they are unregistered. The PS will stay valid across boots.

NOTE: This specification uses the UUID structure to define fixed values for predefined key identifiers.

End of informative comment.

Keys once registered in Persistent Storage (PS) will stay registered in PS until they are unregistered. The PS will stay valid across platform resets.

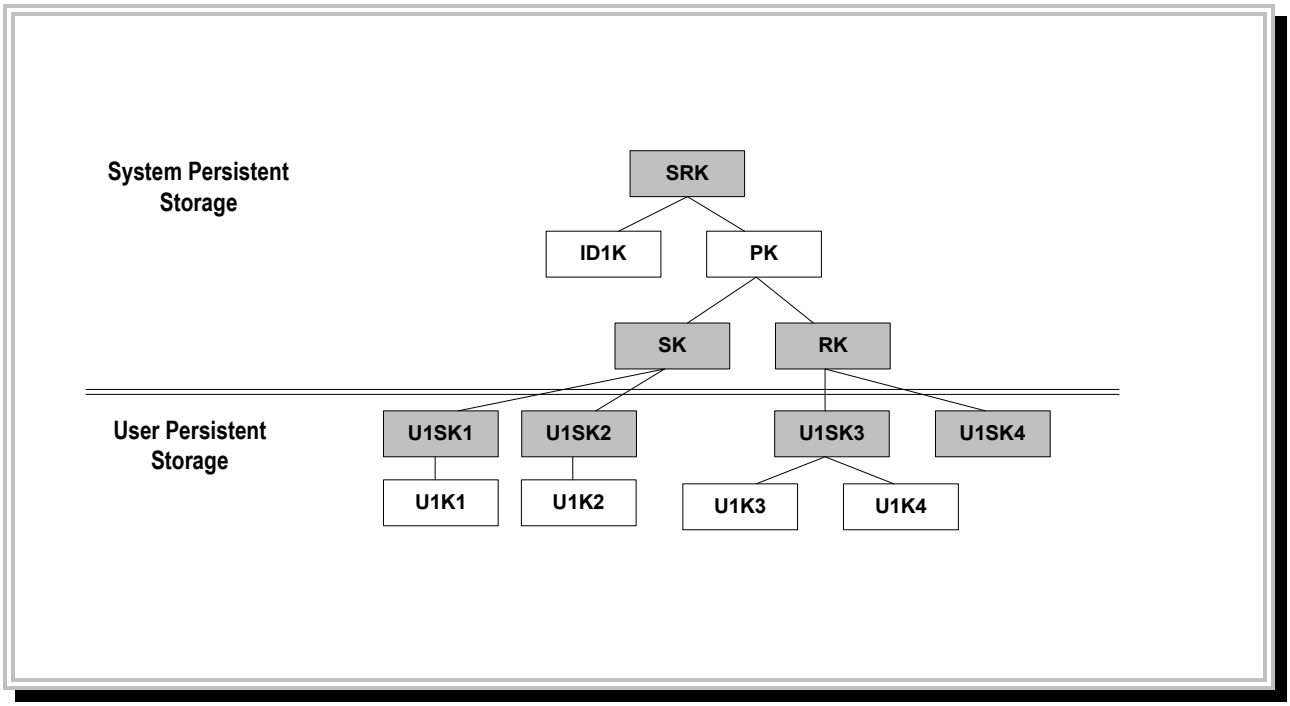


Figure 2-1 Load Key Flow Diagram

The grayed keys in the key hierarchy diagram above are mandatory storage keys and are addressed by fixed UUIDs, they have the same attributes (e.g. migratable, auth) and are stored either in the persistent storage of TCS or the persistent storage of TSP on all platforms. Keys stored in the user specific persistent storage of TSP can be addressed by the same UUID for each user but of course the UUID will still reference a different user storage key.

The following table lists the definition of the keys shown in figure 3.1:

Key	UUID	PS Type	Migratable	Authorization	Description
SRK	Fixed by TCG	System	No	No	Storage Root Key.
PK		System	No	No	Platform specific key.
RK	Fixed	System	Yes	No	Roaming Key.

Copyright TCG 2003

	by TCG				
SK	Fixed by TCG	System	No	No	System specific storage key.
ID1K		System	No	Yes	Identity key #1.
U1SK1	Fixed by TCG	User	No	No	Storage key #1 of User #1.
U1SK2	Fixed by TCG	User	No	Yes	Storage key #2 of User #1.
U1SK3	Fixed by TCG	User	Yes	No	Storage key #3 of User #1.
U1SK4	Fixed by TCG	User	Yes	Yes	Storage key #4 of User #1.
U1K1		User	No	Yes/No	Leaf Key #1 of user #1.
U1K2		User	No	Yes/No	Leaf Key #2 of user #1.
U1K3		User	Yes	Yes/No	Leaf Key #3 of user #1.
U1K4		User	Yes	Yes/No	Leaf Key #4 of user #1.

2.7.1 TSS Load Key Command Flow

Start of informative comment:

This chapter describes the flow of the Tspi Load Key commands in different scenarios.

End of informative comment.

TSP Definitions:

Type	Description
KS (TSP Key Storage)	TSP Storage of Keys. These keys are typically associated with an application.
HMG (HMAC Generator)	HMAC and SHA1 generator. Takes the relevant parameters and generates the authorization data using HMAC and SHA1 operations. The BSG uses "TCG Specific Knowledge" to build the authorization data, for example, it must add the command ordinal to the HMAC calculation.

TCS Definitions:

Type	Description
KCS (TCS Key and Credential Storage)	TCS Storage of Keys and Credentials. These keys and credentials are typically related with the platform. Therefore the keys cannot be roaming keys.
KCM (Key Cache Manager)	Handles key-caching whenever required. The Key cache manager typically uses TPM_SaveKeyContext and TPM_LoadKeyContext for the key caching.
KCMS (Key Cache Manager Storage)	The storage of the KCM.
PBG (Parameter Block Generator)	The Parameter Block Generator uses "TCG Specific Knowledge" to concatenate its input parameters and other parameters (ordinal, tag, etc.) to a TPM Parameter

	Block command.

General Definitions:

Type	Description
BS Key	"Byte Stream" format structure of a TCG Key.

2.7.2 TSS Load Key Flow Diagram

Start of informative comment:

Load Key Flow Description as described the following diagram.

Case 1: Tspi_LoadKeyByBlob

Case 2: Tspi_LoadKeyByUUID, Key registered in KS,
2.1 Parent Key Authorization is Not Required.
2.2 Parent Key Authorization is Required.

Case 3: Tspi_LoadKeyByUUID, Key registered in KCS
3.1 Parent Key Authorization is Not Required.
3.2 Parent Key Authorization is Required.

End of informative comment.

Copyright TCG 2003

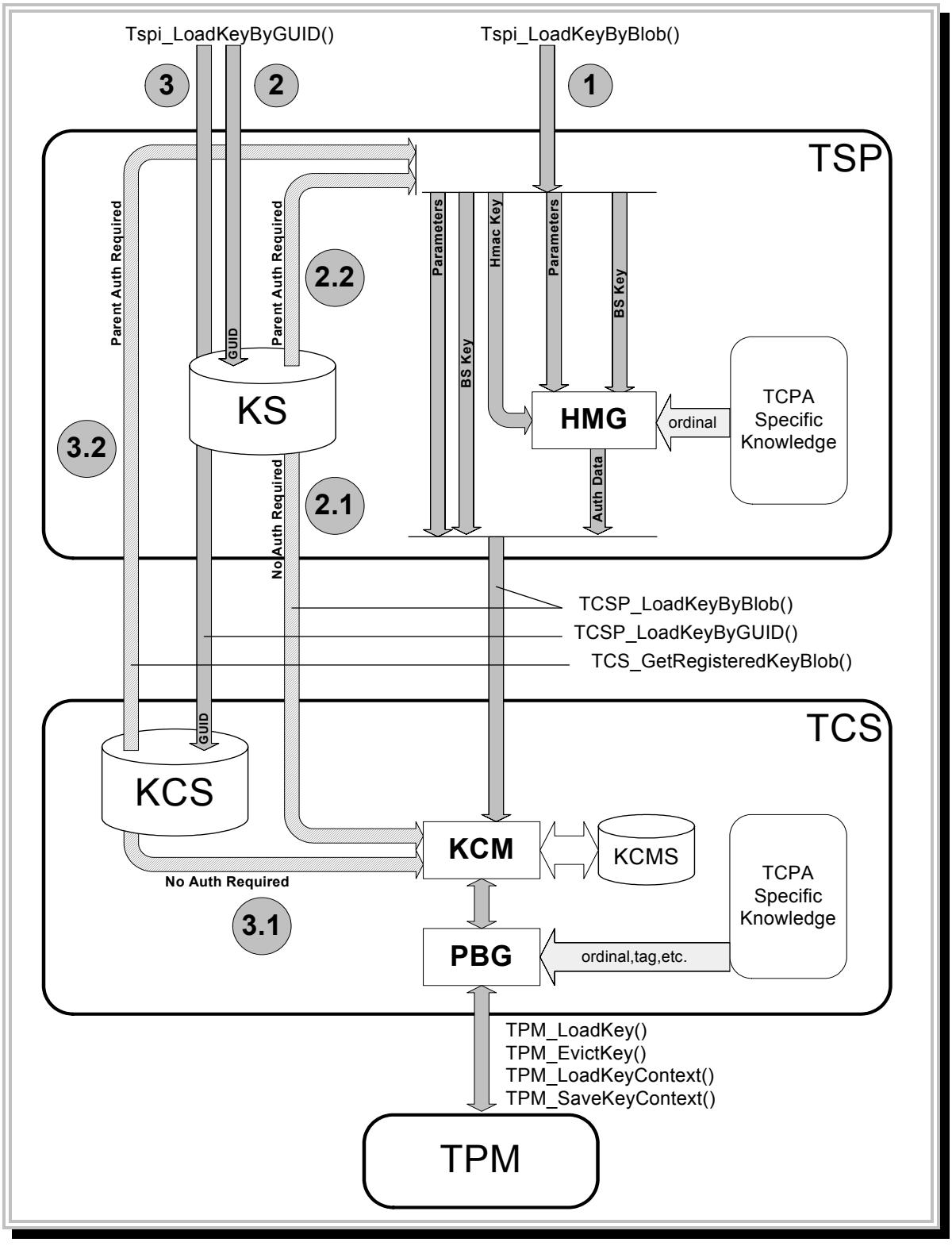


Figure 2-2 Load Key Flow Diagram

2.7.3 Key Handles

Start of informative comment:

To further explain the relationship between the various key handles the following illustration is presented:

First defining a common labeling convention:

- TPM - KeyHandle

  UINT32 to address a key loaded in the TPM. Created and maintained by TPM

- TCS - KeyHandle

  TCS_KEY_HANDLE to address a key object created and maintained by the TCS.

  Handle will internally be mapped to the appropriate TPM KeyHandle.

- TSP - KeyHandle

  TSS_HKEY to address a key object created and maintained by the TSP.

  Handle will internally be mapped to the appropriate TCS KeyHandle.

Using CreateWrapKey: as a rough sequence from the application (app) through the TSP, the TCS and down to the TPM

- (App -> TSP) Load parent wrapping key: Tspi_Context_LoadKeyByBlob. TSP returns key handle (Psp).
- (App -> TSP) Create a key object and initialize object parameters according to the key to be created. TSP returns a key handle (Nsp).
- (App -> TSP) Create a policy object and assign the policy to key object addressed by (Nsp).
- (App -> TSP) Call Tspi_Key_CreateKey
- (TSP internal) Take KeyHandle(Psp) and get appropriate TCS Keyhandle(Pcs).
- (TSP internal) Establish an OSAP session using the policy assigned to the wrapping parent key (secret) and the key object representing the wrapping parent key (TCS KeyHandle (Pcs) of parent key).
- (TSP internal) Encrypt the secrets of the key to be created and compute the authorization. The new secrets are provided by the policy object assigned to the key object addressed by (Nsp).
- (TSP -> TCS) Call Tcsip_CreateWrapKey using the appropriate TCS KeyHandle (Pcs) of the already loaded wrapping parent key
- (TCS internal) Take TCS KeyHandle(Pcs) and get appropriate TPM Keyhandle(Ptpm).
- (TCS internal) Create TPM byte stream for TPM_CreateWrapKey
- (TCS internal) Call TPM

Note:

May be the KCM must reload the parent wrapping key before calling TPM_CreateWrapKey. Using the Load/Save Key Context and Auth Context commands of the TPM this should be no problem.

End of informative comment.

Copyright TCG 2003

2.8 Portable Data

Start of informative comment:

The Key Management Services and the Data Management Services of TSS allow exporting/importing information from/into the TSS utilizing functionalities provided by TSPI.

The format of the data blobs is designed to allow transporting the information independent of a platform using ASN.1 BER encoding.

End of informative comment.

Definition:

```
TssBlobType ::= ENUMERATED
{
   Key-Blob (1),            -- TCPA_KEY as returned from TPM
   PubKey-Blob (2),         -- TCPA_PUBKEY as returned from TPM
   MigKey-Blob (3),         -- Blob as returned from TPM
                                  (TPM_CreateMigrationBlob)
   SealedData-Blob (4),     -- TCPA_STORED_DATA as returned from TPM
   BoundData-Blob (5),      -- TCPA_BOUND_DATA as returned from TPM
   Migticket-Blob (6),      -- TCPA migration data as returned from TPM
   PrivateKey-Blob (7),     -- Encrypted private TCG key blob returned from TPM
   PrivateKey-MOD1-Blob (8)-- The mod1 of the private key to be wrapped
                               by the TSS
}

TssBlobType ::= INTEGER

TssBlob ::= SEQUENCE
{
    StructVersion INTEGER, -- Version of this structure; at the moment 1
    BlobType      TssBlobType,   -- Type of Blob; see enum
    BlobLength    INTEGER,       -- Length of Blob
    Blob          OCTET STRING   -- Blob as returned from TPM
}
```

3. TCG Service Provider (TSP)

3.1 Theory of Operation

3.1.1 Functional Overview

The TSS Service Provider module provides an API making a set of TCG functionalities accessible for TCG-aware application software. It is through this TSP that an application can access data or services on a specific TPM.

The following classes of services are implemented within existing TPMs:

- Integrity Collection and Reporting Services
- Protected Storage Services
- Cryptographic Services
- Credential Services

The standardization of that API to these services enables development and maintenance of TCG-aware application software with no or at least less specific expertise in TPM internals.

Authorization Session Handling

The TSS Service Provider (TSP) hides the management of TCG related authorization sessions from the calling application. There is no requirement for the application to initialize any OIAP or OSAP authorization session. The TSP initializes a required authorization session and handles all internal data of that session.

3.1.2 Interface Design

Although the TSPI is defined as a C interface, this API uses an object-oriented approach. All TSPI functions deal with one or more object handle parameters addressing a certain instances of a class. Callers perform actions on objects utilizing public methods. Attributes are accessed by calling set or get object attribute methods.

3.1.2.1 Classes

The TSPI defines the following classes:

- Context class
- Policy class
- TPM class
- Key class
- Encrypted Data class (sealed or bound data)
- PCR Composite class
- Hash class

Context class

The context contains information about the TSP-Object's execution environment, such as the identity of the object and the transaction/communication with other TSS-Software modules (e.g. TSS-Core-Service). A context object in the TSP environment is similar in concept to the process context that an operating system maintains for an executing program.

Copyright TCG 2003

Policy class

The policy class infrastructure of the TSP can be used to configure policy settings and behaviors for the different user applications. The application can use the TSP-Policy infrastructure to provide specialized secret handling (e.g. CallBack, Lifetime, …) for the authorization.

TPM class

One purpose of the TPM class is to represent the owner for a TCG subsystem (TPM). The owner of a TPM is comparable with an administrator in the PC environment. For that reason there exists only one instance of the TPM class per context. This object is automatically associated with one policy object; which must be used to handle the owner authentication data. On the other hand it provides some basic control and reporting functionality.

Key class

The key class type defined by the TSS service provider represents an entry into the TCG key handling and functionality. Each instance of a key object represents a specific key node, that is part of the TSS key path (hierarchy). A key object, which needs authentication, can be assigned to a policy object that controls the secret management.

Encrypted Data class (Seal and Bind)

This class can be used to join externally (e. g. user, application) generated data to a TCG-aware system (bound to PCR or Platform). For the authentication process this class can be assigned to a policy object.

PCR-Composite class

The contents of the platform configuration register (PCR) of a TCG system can be used to establish a confidence level for this system. This class provides a comfortable way to deal with PCR values (e.g. select, read, write). An object handle of such a class is used from all TSP functions that need PCR information in its parameter list.

Hash class

A hash value represents a unique value corresponding to a particular set of bytes. This class provides a cryptographically secure way to use these functions for digital signature operations.

3.1.2.2 Object Relationship

Working objects are subdivided into authorized and non-authorized working objects. Non-authorized working objects are the PCR composite objects and hash objects. Authorized working objects are the TPM object, key objects and encrypted data objects.

Copyright TCG 2003

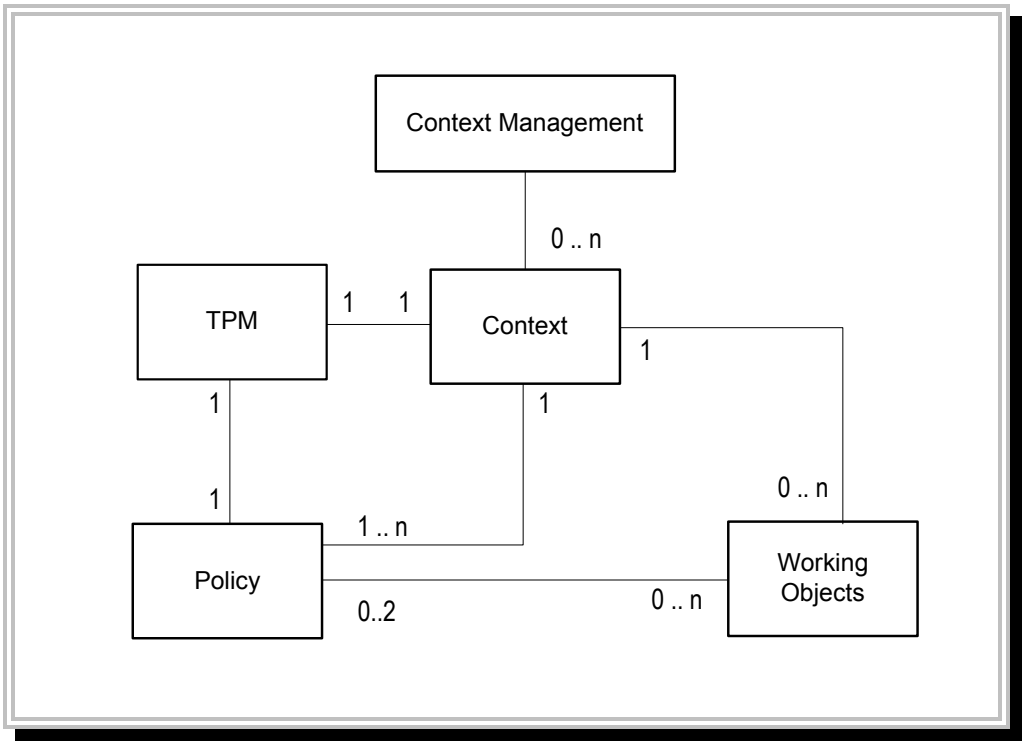


Figure 3-1 Object Relationship

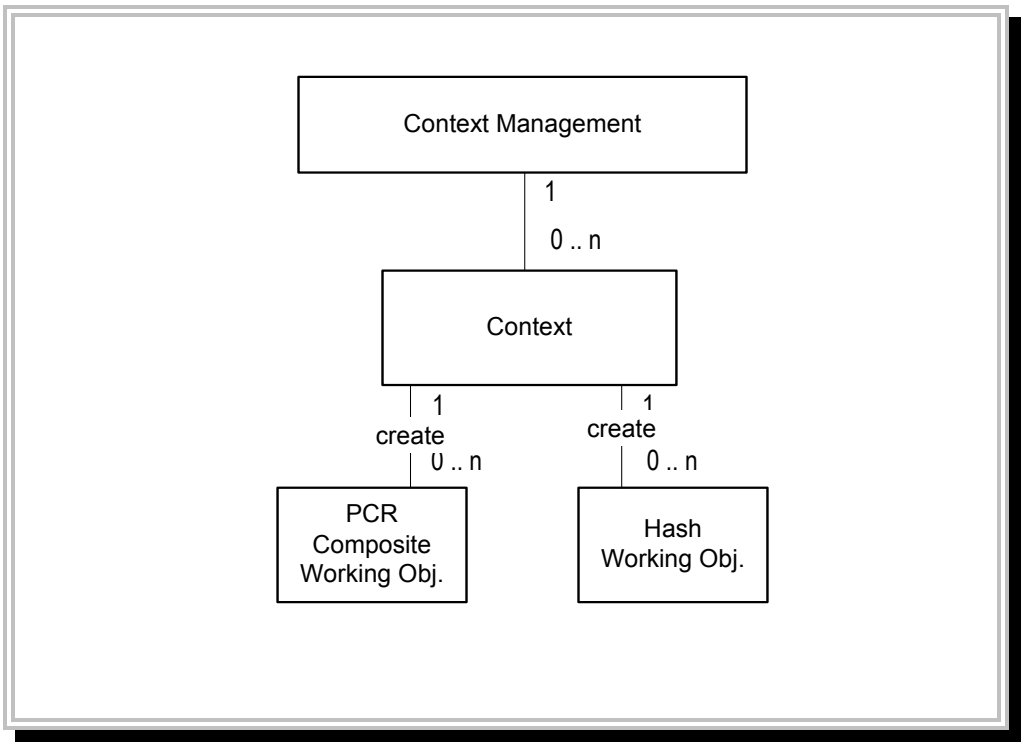


Figure 3-2 Non authorized working object relationship

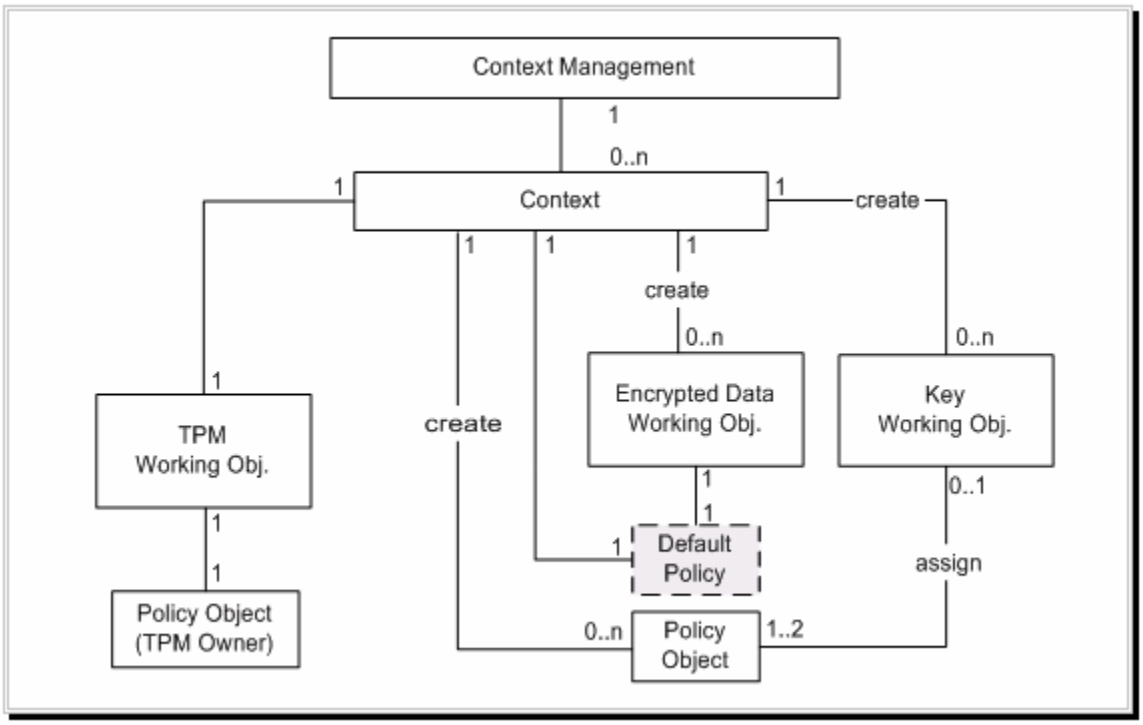


Figure 3-3 Authorized working object relationship

The calling application (the user) may have to supply authorization data only once for each policy it wants to utilize. A policy may be assigned to several objects like key objects, encrypted data objects or a TPM object utilizing the Tspi_Policy_AssignToObject( ) method. Each of these objects will utilize its assigned policy object to process authorized TPM commands using internal functions of the policy object.

On creation of a context a default policy is created and each new created object is automatically assigned to this default policy. The default policy for each working object exists as long as no assign command sets a new policy object to the working object. The TPM object has a separate policy object that represent the owner of the TPM. Assigning one or more working objects to a policy object is done by internal policy and working object functions.

3.1.3 Authorization Data Handling

The TSP provides policy objects helping the calling application handling and caching secrets for authorized objects. The following objects are authorized objects: TPM, Key and encrypted data. The TSP also knows when to use the secret of the object (for OIAP) or a session secret derived from the object secret (for OSAP). An authorized object is assigned to exactly one policy object but a policy object may be assigned to 0..n authorized objects to facilitate the same secret for various authorized objects. If an authorized object is not explicitly assigned to a certain policy object, the TSP automatically assigns this authorized object to a default policy object.

There is no requirement for the application to provide a secret when calling a function where authorization is necessary. The TSP asks the user to enter a secret in a dialog box for this purpose. This service is not restrictive and can be modified by an application on a policy object base. Depending on the platform the service provider should use non-paged memory for secrets provided to it. Before freeing this memory it should also zero the memory area used for this.

The default mode for a policy object is set to TSS_SECRET_MODE_POPUP, but can also be set to TSS_SECRET_MODE_PLAIN, TSS_SECRET_MODE_SHA1 or TSS_SECRET_MODE_CALLBACK. If the mode is set to TSS_SECRET_MODE_CALLBACK, the application is responsible to register a callback function during initialization of a policy object. If the callback is not registered the command will fail.

Mode:

TSS_SECRET_MODE_POPUP

The TSP displays a dialog to enter pass phrase. The pass phrase provided by the user is handled as a null terminated UNICODE string and will be hashed using SHA1 to get the authorization secret. Once a pass phrase was provided the TSP may cache the resulting authorization secret in the appropriate policy object (depending on the policy object settings) and the dialog will not pop up again.

Examples:

- If an identity key is to be used for quoting a system the dialog is asking for the secret required to use that identity key.
- If a secret has to be changed the dialog is asking for the old and new secret.

TSS_SECRET_MODE_PLAIN or TSS_SECRET_MODE_SHA1

The application can set a secret per policy base. The TSP caches and uses the secret on processing authorized commands on assigned authorized objects.

The secret can be set as plaintext or as digest (SHA1) by the application.

Type	Definition
TSS_SECRET_MODE_SHA1	The TSP will only accept an array of 20 bytes and will not touch this data at all. The data will be handled as the authorization secret.
TSS_SECRET_MODE_PLAIN	The TSP will accept any byte array and will calculate a hash using SHA1 to get the authorization secret.

TSS_SECRET_MODE_CALLBACK

This mode can be used by an application in the following situations:

- An application doesn't want to reveal the secret.

Copyright TCG 2003

- The secret is collected by another mechanism like a biometric device.
- The secret is protected by another security token like a smart card.

The TSP will call one of the application provided callback functions. The first one is used to calculate the HMAC for the authorization data required for TPM command authorization. The second one is used for XOR encryption of a new or changed secret. The third one is used to encrypt a secret with a public key.

All necessary parameters are included in the callback function to perform the above actions.

3.1.3.1 Secrets Handled by Service Provider

Secret Caching

The concept of TCG defines that keys may require authorization for their use. The secret for a key requiring authorization has to be presented for each usage of such a key. For usability the TSS Service Provider supports a mechanism allowing that the secret has to be presented only once to the TSP for a specified duration. For all subsequent calls requiring this secret, the Service Provider takes the secret from its internal secret cache.

Caching of secrets rises up security concerns.

Generally an application must trust that the loaded TSP is the authentic TSP (e.g. evaluated by a Conformance Entity) it wants to load. For this purpose it must utilize any appropriate means e.g. the TCG means to rely on a software stack.

If the application cannot rely on the authenticity of the loaded TSP it must not use it. Instead it has to call the TSS Core Service interface directly.

Lifetime of a cached secret

The cache with it's secret is destroyed when:

- the secret is changed,
- the secret is flushed,
- the secret use counter runs down,
- the secret gets timed out or,
- the policy object is closed.

3.1.3.2 Secrets Handled by Application

An application may trust the TSP since it is authentic and trusted, but there are several circumstances where an application doesn't want to reveal the secret or cannot provide the secret to the TSS Service Provider. Usage of a smart card containing the secret could be one example for such a scenario.

For this purpose the TSPI provides a callback function mechanism.

3.1.4 Implementation Considerations

The main focus of the TSS Service Provider is to abstract implementation details at TPM level and expose TPM functionality in a way that TCG-aware application software can access it easily. In particular, this will enable application developers to write TCG-aware applications without requiring much TPM intimate knowledge.

The interfaces exposed by the TSP are developed within the context of a given platform being consistent with API standards of that platform. It is possible to implement the interfaces defined in this specification in a way that is suitable for use with procedural programming languages such as C as well as with object-oriented languages such as C++. This may entail some modification of the naming

conventions, parameter types, and so on. As long as the implementation is functionally equivalent, it is consistent with the intent of this specification.

A TSP is built upon the services exposed by the TSS Core Service. This provides TSP developers with a set of functionality they can use to simplify the development and maintenance of their software.

3.1.5 User Interface Elements

The user interface (UI) provided by the TSS Service Provider should be implemented according to the published guidelines of the target environment.

In particular, the TSP implements a UI for authorization data. The TSP is the best place to implement authorization data management, because it encapsulates knowledge about authorization protocols, authorization session, authorization data length and so on.

3.1.6 Runtime Considerations

The TSS Service provider is implemented as a user-mode application module running in the context of and with the same privileges as the calling TCG-aware application. In a Windows system the TSP will typically be an in-process COM sever or a DLL.

3.2 TSPI-specific Return Code Defines

The TSPI common return codes are returned as TSS error code and are defined in section 2.4.2 Common Return Code Defines. Below are TSP-specific return codes.

Type	Definition
TSS_E_INVALID_OBJECT_TYPE	Object type not valid for this operation.
TSS_E_INVALID_OBJECT_INIT_FLAG	Invalid object initialization flag
TSS_E_INVALID_HANDLE	Invalid object handle
TSS_E_NO_CONNECTION	TCS connection has not been established, but is required.
TSS_E_CONNECTION_FAILED	Establishing a connection to Core Service failed
TSS_E_CONNECTION_BROKEN	Communication with Core Service has been established but has since failed.
TSS_E_HASH_INVALID_ALG	Invalid hash algorithm.
TSS_E_HASH_INVALID_LENGTH	Hash length is inconsistent with hash algorithm.
TSS_E_HASH_NO_DATA	Hash object has no internal hash value.
TSS_E_SILENT_CONTEXT	Context is silent and but requires user input.
TSS_E_INVALID_ATTRIB_FLAG	Flag value for attrib-functions inconsistent.
TSS_E_INVALID_ATTRIB_SUBFLAG	Subflag value for attrib-functions inconsistent.
TSS_E_INVALID_ATTRIB_DATA	Data for attrib-functions invalid.
TSS_E_NO_PCRS_SET	No PCR register are selected or set.
TSS_E_KEY_NOT_LOADED	The addressed key is currently not loaded.
TSS_E_KEY_NOT_SET	No key information is currently available.
TSS_E_VALIDATION_FAILED	Internal validation of data failed.
TSS_E_TSP_AUTHREQUIRED	Authorization is required.
TSS_E_TSP_AUTH2REQUIRED	Multiple authorization is required.

Copyright TCG 2003

TSS_E_TSP_AUTHFAIL	Authorization failed.
TSS_E_TSP_AUTH2FAIL	Multiple authorization failed.
TSS_E_KEY_NO_MIGRATION_POLICY	There's no migration policy object set for the addressed key.
TSS_E_POLICY_NO_SECRET	No secret information is currently available for the addressed policy object.
TSS_E_INVALID_OBJ_ACCESS	The operation failed due to an invalid object status.
TSS_E_INVALID_ENCSCHEME	Invalid encryption scheme.
TSS_E_INVALID_SIGSCHEME	Invalid signature scheme.
TSS_E_ENC_INVALID_LENGTH	Invalid length of data to be encrypted.
TSS_E_ENC_NO_DATA	No data to encrypt.
TSS_E_ENC_INVALID_TYPE	Invalid encryption type.
TSS_E_INVALID_KEYUSAGE	Invalid key usage.
TSS_E_VERIFICATION_FAILED	Verification of signature failed.
TSS_E_HASH_NO_IDENTIFIER	The hash algorithm identifier is not set.

3.3 Interface Description

3.3.1 Syntax

The syntax used in describing the TSS Service Provider is based on the common procedural language constructs. Data types are described in terms of common C.

3.3.2 Calling Conventions regarding Memory Management

The TSP allocates memory for out parameters and provides a function to free the memory previously allocated by the TSP on a context object base. The calling application MUST free memory allocated by the TSS Service Provider. The caller of the TSP functions is responsible for calling Tspi_Context_FreeMemory for each call that produced allocation of memory.

Example

```
// prototyping
TSS_RESULT Func(
   UINT32* pulLength,      // out
   BYTE** prgbData);       // out

// C++ sample
TSS_RESULT     Result = TSS_SUCCESS;
UINT32   ulLength = 0L;
BYTE* prgbData = NULL;

// Init TSS_HCONTEXT hContext via a TSPI function

Result = Func(&ulLength, &prgbData);

// if (TSS_SUCCEEDED(Result)) …
// work with prgbData

// afterwards cause the TSP to free pData
Tspi_Context_FreeMemory(hContext, prgbData);
```


Copyright TCG 2003

3.3.3 Classes and Methods

3.3.3.1 Common Methods Definition

3.3.3.1.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32bit attribute of the object.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_SetAttribUint32
(
   TSS_HOBJECT hObject,    // in
   TSS_FLAG    attribFlag, // in
   TSS_FLAG    subFlag,    // in
   UINT32      ulAttrib    // in
);
```

Parameters

hObject
Handle of the object where the attribute is to be set.

attribFlag
Flag indicating the attribute to set (see table Defined Attributes).

subFlag
Sub flag indicating the attribute to set (see table Defined Attributes).

ulAttrib
Value which is to be set for the specified attribute (see table Defined Attributes).

Defined Attributes

See table Defined Attributes of appropriate Class Definition section

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INVALID_ATTRIB_FLAG
TSS_E_INVALID_ATTRIB_SUBFLAG
TSS_E_INVALID_ATTRIB_DATA
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

Copyright TCG 2003

3.3.3.1.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32bit attribute of the object

End of informative comment.

Definition:

```
TSS_RESULT Tspi_GetAttribUint32
(
   TSS_HOBJECT    hObject,    // in
   TSS_FLAG       attribFlag, // in
   TSS_FLAG       subFlag,    // in
   UINT32*        pulAttrib   // out
);
```

Parameters

hObject
Handle of the object to retrieve the attribute.

attribFlag
Flag indicating the attribute to query (see table Defined Attributes).

subFlag
Sub flag indicating the attribute to query (see table Defined Attributes).

pulAttrib
Receives the value of the specified attribute (see table Defined Attributes).

Defined Attributes

See table Defined Attributes of the appropriate class definition section

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INVALID_ATTRIB_FLAG
TSS_E_INVALID_ATTRIB_SUBFLAG
TSS_E_INVALID_ATTRIB_DATA
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

Copyright TCG 2003

3.3.3.1.3 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32bit attribute of the object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_SetAttribData
(
   TSS_HOBJECT    hObject,          // in
   TSS_FLAG       attribFlag,       // in
   TSS_FLAG       subFlag,          // in
   UINT32         ulAttribDataSize, // in
   BYTE*          rgbAttribData     // in
);
```

Parameters

hObject
Handle of the object where the attribute is to be set.

attribFlag
Flag indicating the attribute to set (see table Defined Attributes).

subFlag
Sub flag indicating the attribute to set (see table Defined Attributes).

ulAttribDataSize
Supplies the length (in bytes) of the *rgbAttribData* parameter.

rgbAttribData
Pointer to the actual data which is to be set for the specified attribute (see table Defined Attributes).

Defined Attributes

See table Defined Attributes of appropriate Class Definition section

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INVALID_ATTRIB_FLAG
TSS_E_INVALID_ATTRIB_SUBFLAG
TSS_E_INVALID_ATTRIB_DATA
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

Copyright TCG 2003

3.3.3.1.4 Tspi_GetAttribData

Start of informative comment:

This method gets a non 32bit attribute of the object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_GetAttribData
(
   TSS_HOBJECT    hObject,           // in
   TSS_FLAG       attribFlag,        // in
   TSS_FLAG       subFlag,           // in
   UINT32*        pulAttribDataSize, // out
   BYTE**         prgbAttribData     // out
);
```

Parameters

hObject
Handle of the object where to retrieve the attribute.

attribFlag
Flag indicating the attribute to query (see table Defined Attributes).

subFlag
Sub flag indicating the attribute to query (see table Defined Attributes).

pulAttribDataSize
Receives the length (in bytes) of the *prgbAttribData* parameter.

prgbAttribData
On successful completion of the command, this parameter points to a buffer containing to the actual data of the specified attribute (see table Defined Attributes).

Defined Attributes

See table Defined Attributes of appropriate Class Definition section

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INVALID_ATTRIB_FLAG
TSS_E_INVALID_ATTRIB_SUBFLAG
TSS_E_INVALID_ATTRIB_DATA
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_GetAttribData method allocates a memory block for the requested attribute data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

Copyright TCG 2003

3.3.3.1.5 Tspi_ChangeAuth

Start of informative comment:

This method changes the authorization data (secret) of an entity (object) and assigns the object to the policy object. All classes using secrets provide this method for changing their authorization data.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_ChangeAuth
(
   TSS_HOBJECT hObjectToChange,  // in
   TSS_HOBJECT hParentObject,    // in
   TSS_HPOLICY hNewPolicy        // in
);
```

Parameters

hObjectToChange
Handle of the object the authorization data should be changed.

hParentObject
Handle of the parent object wrapping the object addressed by *hObjectToChange*.

hNewPolicy
Handle of the policy object providing the new authorization data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

This command requires the parent object to unwrap the old authorization data and to create a shared secret, which is utilized to encrypt the new authorization data ensuring a secure authorization data transmission to the TPM. On successful completion of the command the object addressed by hObjectToChange is bound to the policy object addressed by hNewPolicy.

Special considerations for TPM Owner and SRK secrets

Owner authorization is required to change the Owner and SRK authorizations.

To change the TPM owner authorization: the ObjectToChange handle is the TPM Object handle and the parentObject will be NULL (0x00000000).

To change the SRK authorization: the ObjectToChange is the SRK Object handle and parentObject handle is the TPM Object handle.

Copyright TCG 2003

3.3.3.1.6 Tspi_ChangeAuthAsym

Start of informative comment:

This method changes the authorization data (secret) of an entity (object) utilizing the asymmetric change protocol and assigns the object to the policy object. All classes using secrets provide this method for changing their authorization data.

This method changes the authorization data of an object ensuring that the parent of the object does not get knowledge of the new secret.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_ChangeAuthAsym
(
   TSS_HOBJECT    hObjectToChange,  // in
   TSS_HOBJECT    hParentObject,    // in
   TSS_HKEY       hIdentKey,        // in
   TSS_HPOLICY    hNewPolicy        // in
);
```

Parameters

hObjectToChange
Handle of the object the authorization data should be changed.

hParentObject
Handle of the parent object wrapping the object addressed by *hObjectToChange*.

hIdentKey
Handle of the identity key object required to proof the internally created temporary key.

hNewPolicy
Handle of the policy object providing the new authorization data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

The asymmetric change protocol requires creating a temporary asymmetric key pair. The creation of this key pair according to the rules of TCG is internally proofed utilizing the identity key object. On successful completion of the command the object addressed by hObjectToChange is bound to the policy object addressed by hNewPolicy.

Special considerations for TPM Owner and SRK secrets

Owner authorization is required to change the Owner and SRK authorizations.

To change the TPM owner authorization: the ObjectToChange handle is the TPM Object handle and the parentObject will be NULL (0x00000000).

To change the SRK authorization: the ObjectToChange is the SRK Object handle and parentObject handle is the TPM Object handle.

3.3.3.1.7 Tspi_GetPolicyObject

Start of informative comment:

This method returns a policy object currently assigned to a working object

End of informative comment.

Definition:

```
TSS_RESULT Tspi_GetPolicyObject
(
   TSS_HOBJECT    hObject,    // in
   TSS_FLAG       policyType, // in
   TSS_HPOLICY*   phPolicy    // out
);
```

Parameters

hObject
Handle of the object.

policyTypet
Flag indicating the policy type of interest. (see table Defined Attributes)

phPolicy
Receives the handle to the assigned policy object.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

In most cases a usage policy object is of interest (TSS_POLICY_USAGE). A few key object functions use a migration policy object (TSS_POLICY_MIGRATION).

Copyright TCG 2003

3.3.3.2 Tspi_Context Class Definition

Start of informative comment:

The Tspi_Context class represents a context of a connection to the TSS Core Service running on the local or a remote TCG system.

The focus of the Context object is:

- to provide a connection to a TSS Core Service. There might be multiple connections to the same or different core services.
- to provide functions for resource management and freeing of memory
- to create working objects.
- to establish a default policy for working objects as well as a policy object for the TPM object representing the TPM owner.
- to provide functionality to access the persistent storage database.

End of informative comment.

3.3.3.2.1 Tspi_Context_Create

Start of informative comment:

This method returns a handle to a new context object. The context handle is used in various functions to assign resources to it.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_Create
(
   TSS_HCONTEXT*  phContext   // out
);
```

Parameters

phContext
Receives the handle to the created context object.

Return Values

TSS_SUCCESS
TSS_E_INTERNAL_ERROR

Remarks

3.3.3.2.2 Tspi_Context_Close

Start of informative comment:

This method destroys a context and releases all assigned resources.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_Close
(
   TSS_HCONTEXT   hContext    // in
);
```

Parameters

hContext
Handle of the context object which is to be closed.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

3.3.3.2.3 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32bit attribute of the context object.

End of informative comment.

Definition:

See section 3.3.3.1.1 for definition.

Parameters

See section 3.3.3.1.1 for description.

Defined Attributes

Flag	SubFlag	Attribute Value	Description
TSS_TSPATTRIB_CONTEXT_SILENT_MODE		TSS_TSPATTRIB_CONTEXT_NOT_SILENT	TSP dialogs are shown (Default).
		TSS_TSPATTRIB_CONTEXT_SILENT	TSP dialogs are not shown.

Return Values

See section 3.3.3.1.1 for description.

Remarks

Copyright TCG 2003

3.3.3.2.4 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32bit attribute of the context object

End of informative comment.

Definition:

See section 3.3.3.1.2 for definition.

Parameters

See section 3.3.3.1.2 for description.

Defined Attributes

Flag	SubFlag	Attribute Value	Description
TSS_TSPATTRIB_CONTEXT_SILENT_MODE		TSS_TSPATTRIB_CONTEXT_NOT_SILENT	TSP dialogs are shown (Default).
		TSS_TSPATTRIB_CONTEXT_SILENT	TSP dialogs are not shown.

Return Values

See section 3.3.3.1.2 for description.

Remarks

Copyright TCG 2003

3.3.3.2.5 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32bit attribute of the context object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 3.3.3.1.3 for definition.

Parameters

See section 3.3.3.1.3 for description.

No Attributes Defined yet

Return Values

See section 3.3.3.1.3 for description.

Remarks

3.3.3.2.6 Tspi_GetAttribData

Start of informative comment:

This method gets a non 32bit attribute of the context object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 3.3.3.1.4 for definition.

Parameters

See section 3.3.3.1.4 for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_CONTEXT_MACHINE_NAME	0	Machine name of the TSS given as a null terminated UNICODE string.

Return Values

See section 3.3.3.1.4 for description.

Remarks

3.3.3.2.7 Tspi_Context_Connect

Start of informative comment:

This method establishes a connection to a local or remote TSS system.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_Connect
(
   TSS_HCONTEXT   hContext,      // in
   UNICODE*    wszDestination // in
);
```

Parameters

hContext
Handle of the context object

wszDestination
Pointer to a null terminated UNICODE string specifying the remote system which is to be connected.
If NULL, the context object is bound to the local system.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_NO_CONNECTION
TSS_E_INTERNAL_ERROR

Remarks

3.3.3.2.8 Tspi_Context_FreeMemory

Start of informative comment:

This method frees memory allocated by TSS Service Provider on a context base.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_FreeMemory
(
   TSS_HCONTEXT   hContext,   // in
   BYTE*          rgbMemory   // in
);
```

Parameters

hContext
Handle of the context object

rgbMemory
Pointer to the memory block to be freed.
If NULL, all allocated memory blocks bound to the context are freed.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

Copyright TCG 2003

3.3.3.2.9 Tspi_Context_GetDefaultPolicy

Start of informative comment:

This method provides the default policy object of a context.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_GetDefaultPolicy
(
   TSS_HCONTEXT   hContext,   // in
   TSS_HPOLICY*   phPolicy    // out
);
```

Parameters

hContext
Handle of the context object

phPolicy
Receives the handle of the default policy object bound to the context.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

3.3.3.2.10 Tspi_Context_CreateObject

Start of informative comment:

This method creates and initializes an empty object of the specified type and returns a handle addressing that object. The object is bound to an already opened context.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_CreateObject
(
   TSS_HCONTEXT   hContext,   // in
   TSS_FLAG       objectType, // in
   TSS_FLAG       initFlags,  // in
   TSS_HOBJECT*   phObject    // out
);
```

Parameters

hContext
Handle of the context object

objectType
Flag indicating the object type to create (see section 2.3.2.1).

initFlags
Flag indicating the default attributes of the object (see section 2.3.2.2).

phObject
Receives the handle of the created object.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INVALID_OBJECT_TYPE
TSS_E_INVALID_OBJECT_INIT_FLAG
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TSS_E_ENC_INVALID_TYPE
TSS_E_HASH_INVALID_ALG

Remarks

Copyright TCG 2003

3.3.3.2.11 Tspi_Context_CloseObject

Start of informative comment:

This method destroys the object associated with the object handle. All allocated resources (e.g. objects) associated within the object are also released.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_CloseObject
(
   TSS_HCONTEXT   hContext,   // in
   TSS_HOBJECT    hObject     // in
);
```

Parameters

hContext
Handle of the context object

hObject
Handle of object to be closed.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

3.3.3.2.12 Tspi_Context_GetCapability

Start of informative comment:

This method provides the capabilities of the TSS Core Service or TSS Service Provider.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_GetCapability
(
   TSS_HCONTEXT   hContext,           // in
   TSS_FLAG       capArea,            // in
   UINT32         ulSubCapLength,     // in
   BYTE*          rgbSubCap,          // in
   UINT32*        pulRespDataLength,  // out
   BYTE**         prgbRespData        // out
);
```

Parameters

hContext
Handle of the context object

capArea
Flag indicating the attribute to query (see table Defined Attributes).

ulSubCapLength
The length (in bytes) of the *rgbSubCap* parameter.

rgbSubCap
Data indicating the attribute to query (see table Defined Attributes).

pulRespDataLength
Receives the length (in bytes) of the *prgbRespData* parameter.

prgbRespData
On successful completion of the command, this parameter points to a buffer containing the actual data of the specified capability (see table Defined Attributes).

Defined Attributes

Capability Area	SubCap Area	Response
TSS_TCSCAP_ALG	TSS_ALG_XX: A value of TSS Algorithm ID as defined in 2.3.2.11	Boolean value. TRUE indicates that the TCS supports the algorithm, FALSE indicates that the TCS does not support the algorithm.
TSS_TCSCAP_VERSION		Returns the TSS_VERSION structure that identifies the version of the TCS.
TSS_TCSCAP_CACHING	TSS_TCSCAP_PROP_KEYCACHE	Boolean value. TRUE indicates that the TCS supports key caching, FALSE indicates that the TCS does not support key caching.
TSS_TCSCAP_CACHING	TSS_TCSCAP_PROP_AUTHCACH	Boolean value. TRUE

	E	indicates that the TCS supports authorization session caching, FALSE indicates that the TCS does not support authorization session caching.
TSS_TCSCAP_PERSSTORAGE		Boolean value. TRUE indicates that the TCS supports persistent storage, FALSE indicates that the TCS does not support persisten storage.
Tbd	tbd	Returnes the default key length, supported algorithms with the supported key lengths.
TSS_TSPCAP_ALG	TSS_ALG_XX: A value of TSS Algorithm ID as defined in 0	Boolean value. TRUE indicates that the TSP supports the algorithm, FALSE indicates that the TSP does not support the algorithm.
TSS_TSPCAP_VERSION		Returns the TSS_VERSION structure that identifies the version of the TSP.
TSS_TSPCAP_PERSSTORAGE		Boolean value. TRUE indicates that the TSP supports persistent storage, FALSE indicates that the TSP does not support persisten storage.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_Context_GetCapability method allocates a memory block for the requested capability data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

Copyright TCG 2003

3.3.3.2.13 Tspi_Context_GetTPMObject

Start of informative comment:

This method retrieves the TPM object of a context. Only one instance of this object exists for a given context and implicitly represents a TPM owner.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_GetTpmObject
(
   TSS_HCONTEXT   hContext,   // in
   TSS_HTPM*      phTPM       // out
);
```

Parameters

hContext
Handle of the context object

phTPM
Receives the handle of the TPM object bound to the context.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

Copyright TCG 2003

3.3.3.2.14 Tspi_Context_LoadKeyByBlob

Start of informative comment:

This method creates a key object based on the information got by the key blob and loads the key into the TPM which unwraps the key blob utilizing the key addressed by hUnwrappingKey. The key blob addressed by hUnwrappingKey must have been loaded previously into the TPM. The function returns a handle to the created key object by hKey.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_LoadKeyByBlob
(
   TSS_HCONTEXT   hContext,         // in
   TSS_HKEY       hUnwrappingKey,   // in
   UINT32         ulBlobLength,     // in
   BYTE*          rgbBlobData,      // in
   TSS_HKEY*      phKey             // out
) ;
```

Parameters

hContext
Handle of the context object

hUnwrappingKey
Handle of the key object addressing the key which should be used to the key information provided by the *rgbBlobData* parameter.

ulBlobLength
The length (in bytes) of the *rgbBlobData* parameter.

rgbBlobData
The wrapped key blob to load.

phKey
Receives the handle of the key object representing the loaded key.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

3.3.3.2.15 Tspi_Context_LoadKeyByUUID

Start of informative comment:

This method creates a key object based on the information got from the key manager using the UUID and loads the key into the TPM. The persistent storage provides all information to load the parent keys required to load the key associated with the given UUID.

There are some subtle cases that need to be considered when using this command with parent keys that may require authorization.

If none of the registered keys require authorization, the application can use Tspi_Context_LoadKeyByUUID() as specified below without error or alerts.

If one of the registered keys require authorization, and the application knows the registered key stack, it must get the keys from key database by Tspi_Context_GetKeyByUUID(),assign a policy to the key and loads the key by Tspi_Key_LoadKey().

If one of the registered keys require authorization, and the application doesn't know the key stack, it must first retrieve the key stack information from the key database by calling Tspi_Context_GetRegisteredKeysByUUID()and can then continue as in the above paragraph.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_LoadKeyByUUID
(
   TSS_HCONTEXT   hContext,               // in
   TSS_FLAG       persistentStorageType,  // in
   TSS_UUID       uuidData,               // in
   TSS_HKEY*      phKey                   // out
);
```

Parameters

hContext
Handle of the context object

persistentStorageType
Flag indicating the persistent storage (see section 2.3.2.14) the key is registered in.

uuidData
The UUID of the key by which the key was registered in the persistent storage (TSP or connected TCS).

phKey
Receives the handle of the key object representing the loaded key.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_PS_KEY_NOTFOUND
TSS_E_INTERNAL_ERROR

Remarks

3.3.3.2.16 Tspi_Context_RegisterKey

Start of informative comment:

This method registers a key in the TSS Persistent Storage database.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_RegisterKey
(
   TSS_HCONTEXT   hContext,                    // in
   TSS_HKEY       hKey,                        // in
   TSS_FLAG       persistentStorageType,       // in
   TSS_UUID       uuidKey,                     // in
   TSS_FLAG       persistentStorageTypeParent, // in
   TSS_UUID       uuidParentKey                // in
);
```

Parameters

hContext
Handle of the context object

hKey
Handle of the key object addressing the key to be registered.

persistentStorageType
Flag indicating the persistent storage (see section 2.3.2.14) the key is registered in.

uuidKey
The UUID by which the key is registered in the persistent storage (TSP or connected TCS).

persistentStorageTypeParent
Flag indicating the persistent storage (see section 2.3.2.14) the parent key is registered in.

uuidParentKey
The UUID by which the parent key was registered in the persistent storage (TSP or connected TCS).

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_PS_KEY_NOTFOUND
TSS_E_INTERNAL_ERROR

Remarks

The required key information must be set in the key object by Tspi_SetAttribData( ) before this method is called.

A registered key contains all information required for loading the key into the TPM plus additional information about its parent key.

3.3.3.2.17 Tspi_Context_UnregisterKey

Start of informative comment:

This method unregisters a key from the persistent storage database.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_UnregisterKey
(
   TSS_HCONTEXT   hContext,               // in
   TSS_FLAG       persistentStorageType,  // in
   TSS_UUID       uuidKey,                // in
   TSS_HKEY*      phkey                   // out
);
```

Parameters

hContext
Handle of the context object

persistentStorageType
Flag indicating the persistent storage (see section 2.3.2.14) the key is registered in.

uuidKey
The UUID of the key to be removed from the persistent storage (TSP or connected TCS).

phKey
Receives the handle of a key object containing the info from the archive.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_PS_KEY_NOTFOUND
TSS_E_INTERNAL_ERROR

Remarks

3.3.3.2.18 Tspi_Context_GetKeyByUUID

Start of informative comment:

This method searches the persistent storage for a registered key using the provided UUID and creates a key object initialized according to the found data. On successful completion of the method a handle to the created new key object is returned.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_GetKeyByUUID
(
   TSS_HCONTEXT   hContext,               // in
   TSS_FLAG       persistentStorageType,  // in
   TSS_UUID       uuidData,               // in
   TSS_HKEY*      phKey                   // out
);
```

Parameters

hContext
Handle of the context object

persistentStorageType
Flag indicating the persistent storage (see section 2.3.2.14) the key is registered in.

uuidData
The UUID of the key by which the key was registered in the persistent storage (TSP or connected TCS)

phKey
Receives the handle of the key object representing the key.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_PS_KEY_NOTFOUND
TSS_E_INTERNAL_ERROR

Remarks

3.3.3.2.19 Tspi_Context_GetKeyByPublicInfo

Start of informative comment:

This method searches the persistent storage for a registered key using the provided public key information and creates a key object initialized according to the found data. On successful completion of the method a handle to the created new key object is returned.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_GetKeyByPublicInfo
(
   TSS_HCONTEXT      hContext,               // in
   TSS_FLAG          persistentStorageType,  // in
   TSS_ALGORITHM_ID  algID,                  // in
   UINT32            ulPublicInfoLength,     // in
   BYTE*             rgbPublicInfo,          // in
   TSS_HKEY*         phKey                   // out
);
```

Parameters

hContext
Handle of the context object

persistentStorageType
Flag indicating the persistent storage (see section 2.3.2.14) the key is registered in.

algID
This parameter indicates the algorithm of the requested key.

ulPublicInfoLength
The length of the public key info provided at the parameter *rgbPublicInfo*.

rgbPublicInfo
The public key info provided to identify the key to be looked for at the persistent storage. In case *algID* equals to TSS_ALG_RSA this parameter contains the modulus of the public RSA key.

hKey
Receives the handle of the key object representing the key. In case the key hasn't been found, this value will be NULL.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_PS_KEY_NOTFOUND
TSS_E_INTERNAL_ERROR

Remarks

If the key identified by the public key info was not found at the persistent storage the method will return TSS_E_PS_KEY_NOTFOUND.

Copyright TCG 2003

3.3.3.2.20 Tspi_Context_GetRegisteredKeysByUUID

Start of informative comment:

This method gets an array of TSS_KM_KEYINFO structures. This information reflects the registered key hierarchy. The keys stored in the persistent storage are totally independent from either the context provided in the function call or the context, which was provided while processing the key registration.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_GetRegisteredKeysByUUID
(
   TSS_HCONTEXT      hContext,               // in
   TSS_FLAG          persistentStorageType,  // in
   TSS_UUID*         pUuidData               // in
   UINT32*           pulKeyHierarchySize,    // out
   TSS_KM_KEYINFO**  ppKeyHierarchy          // out
);
```

Parameters

hContext
Handle of the context object

persistentStorageType
Flag indicating the persistent storage (see section 0) the key is registered in.

pUuidData
The UUID the key was registered in the persistent storage (TSP or connected TCS).
If no key UUID is provided (KeyUUID == NULL), the returned array of TSS_KM_KEYINFO structure contains data reflecting the whole key hierarchy starting with root key.
If a certain key UUID is provided, the returned array of TSS_KM_KEYINFO structures only contains data reflecting the path of the key hierarchy regarding that key. The first array entry is the key addressed by the given UUID followed by its parent key up to the root key.

pulKeyHierarchySize
Receives the length (number of array entries) of the ppK*eyHierarchy* parameter.

ppKeyHierarchy
On successful completion of the command, this parameter points to a buffer containing the actual key hierarchy data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_Context_GetRegisteredKeysByUUID method allocates a memory block for the requested key hierarchy data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

This array will return information of the whole registered key hierarchy independent from any context.

Copyright TCG 2003

If no keys have been registered *punKeyHierarchySize = 0 and *ppKeyHierarchy = NULL and the function returns with TSS_SUCCESS.

Copyright TCG 2003

3.3.3.3 TSPI_Policy Class Definition

Start of informative comment:

The Tspi_Policy class represents information authorization data (secrets), authorization data handling and the assigned authorized objects like key objects or encrypted data objects.

Authorization data and secret will be used synonymously.

Secret Lifetime

If an application uses the mode TSS_SECRET_LIFETIME_COUNTER or TSS_SECRET_LIFETIME_TIMER, the application has to be aware that during a command processing the secret may be invalidated because of a time out or because the counter runs out.

TSPI Default Policy

Each context has its own default policy object that is automatically assigned to a new key or encrypted data object after its creation. If this policy object is not appropriate, a different policy object can be assigned with the function Tspi_Policy_AssignToObject(…).

The default policy object has following settings after initialization:

Secret mode = TSS_SECRET_MODE_POPUP

Secret lifetime mode = SECRET_LIFTIME_ALWAYS

End of informative comment.

3.3.3.3.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32bit attribute of the policy object.

End of informative comment.

Definition:

See section 3.3.3.1.1 for definition.

Parameters

See section 3.3.3.1.1 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_POLICY_CALLBACK_HMAC	Applicat-ion provided data.	Address of callback or NULL (disable)	
TSS_TSPATTRIB_POLICY_CALLBACK_XOR_ENC	Applicat-ion provided data.	Address of callback or NULL (disable)	
TSS_TSPATTRIB_POLICY_CALLBACK_TAKEOWNERSHIP	Applicat-ion provided data.	Address of callback or NULL (disable)	
TSS_TSPATTRIB_POLICY_CALLBACK_CHANGEAUHTASYM	Applicat-ion provided data.	Address of callback or NULL (disable)	

TSS_TSPATTRIB_POLICY_SECRET_LIFETIME	TSS_TSPATTRIB_POLSECRET_LIFETIME_ALWAYS	Not important	Secret will not be invalidated.
	TSS_TSPATTRIB_POLSECRET_LIFETIME_COUNTER	Counter value	Secret may be used n-times.
	TSS_TSPATTRIB_POLSECRET_LIFETIME_TIMER	Time value in seconds	Secret will be valid for n seconds.

Return Values

See section 3.3.3.1.1 for description.

Remarks

The lifetime is decreased as soon as the secret is set Tspi_Policy_SetSecret( ) method.. After invalidation of a secret a further call to Tspi_Policy_SetSecret( ) will start the processing again.

3.3.3.3.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32bit attribute of the policy object

End of informative comment.

Definition:

See section 3.3.3.1.2 for definition.

Parameters

See section 3.3.3.1.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_POLICY_CALLBACK_HMAC	Application provided data.	Address of callback or NULL (disable)	
TSS_TSPATTRIB_POLICY_CALLBACK_XOR_ENC	Application provided data.	Address of callback or NULL (disable)	
TSS_TSPATTRIB_POLICY_CALLBACK_TAKEOWNERSHIP	Application provided data.	Address of callback or NULL (disable)	
TSS_TSPATTRIB_POLICY_CALLBACK_CHANGEAUHTASYM	Application provided data.	Address of callback or NULL (disable)	
TSS_TSPATTRIB_POLICY_SECRET_LIFETIME	TSS_TSPATTRIB_POLSECRET_LIFETIME_ALWAYS	TRUE if the flag is set in the policy object, FALSE if not.	Secret will not be invalidated.
	TSS_TSPATTRIB_POLSECRET_LIFETIME_COUNTER	Counter value	Secret may be used n-times.
	TSS_TSPATTRIB_POLSECRET_LIFETIME_TIMER	Time value in seconds	Secret will be valid for n seconds.

Return Values

See section 3.3.3.1.2 for description.

The lifetime is decreased as soon as the secret is set Tspi_Policy_SetSecret( ) method.. After invalidation of a secret a further call to Tspi_Policy_SetSecret( ) will start the processing again.

3.3.3.3.3 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32bit attribute of the policy object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 3.3.3.1.3 for definition.

Parameters

See section 3.3.3.1.3 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_POLICY_POPUPSTRING	0	POPUPSTRING	Text for popup

Return Values

See section 3.3.3.1.3 for description.

Remarks

3.3.3.3.4 Tspi_GetAttribData

Start of informative comment:

This method gets a non 32bit attribute of the policy object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 3.3.3.1.4 for definition.

Parameters

See section 3.3.3.1.4 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_POLICY_POPUPSTRING	0	POPUPSTRING	Text for popup

Return Values

See section 3.3.3.1.4 for description.

Remarks

3.3.3.3.5 Tspi_Policy_SetSecret

Start of informative comment:

This method sets the authorization data of a policy object and defines the handling of its retrieving.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Policy_SetSecret
(
   TSS_HPOLICY hPolicy,          // in
   TSS_FLAG    secretMode,       // in
   UINT32      ulSecretLength,   // in
   BYTE*       rgbSecret         // in
);
```

Parameters

hPolicy
Handle of the policy object

secretMode
Flag indicating the policy secret mode to set (see section 2.3.2.3).

ulSecretLength
The length (in bytes) of the *rgbSecret* parameter.

rgbSecret
The secret data blob.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

If the secret mode does not require any authorization data, the parameter *ulSecretLength* can be 0 and the parameter *rgbSecret* can be NULL.

Copyright TCG 2003

3.3.3.3.6 Tspi_Policy_FlushSecret

Start of informative comment:

The function flushes a cached secret.

End of informative comment.

Definition

```
TSS_RESULT Tspi_Policy_FlushSecret
(
   TSS_HPOLICY hPolicy  // in
);
```

Parameters

hPolicy
Handle of the policy object

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

3.3.3.3.7 Tspi_Policy_AssignToObject

Start of informative comment:

This method assigns an object (working object) like TPM object, key object, encrypted data object to a certain policy. Each of these working objects will utilize its assigned policy object to process an authorized TPM command.

By default each new initialized working object is assigned to the default policy, which is automatically created when a context object is created. When a working object is assigned to a policy the reference to the working object is added to the list of assigned objects stored in the policy object and the reference to the policy object is stored in the working object by internal object functions.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Policy_AssignToObject
(
   TSS_HPOLICY    hPolicy, // in
   TSS_HOBJECT    hObject  // in
);
```

Parameters

hPolicy
Handle of the policy object

hObject
Handle of the object to be assigned

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

Each TPM object, key object or data object can be assigned to one policy object of type TSS_POLICY_USAGE..

A key object or data object additionally can be assigned to one policy object of type TSS_POLICY_MIGRATION

The required policy object type must be set in the policy object by Tspi_SetAttribData( ) or on creation of the policy object.

Copyright TCG 2003

3.3.3.4 Tspi_TPM Class Definition

Start of informative comment:

Owner:

The Owner of the TPM has the right to perform special operations. The process of taking ownership is the procedure whereby the Owner inserts a shared secret into the TPM. For all future operations, knowledge of the shared secret is proof of Ownership. When the Owner wishes to perform one of the special operations then the Owner must use the authorization protocol to prove knowledge of the shared secret.

Identity:

A TPM may have multiple identities. Each identity may have attestation from exactly one Privacy CA.

To create a TSS identity that is recognized by the Privacy CA, the TPM must contain a private endorsement key. For this purpose there must be available: the endorsement credential, the platform credential, the conformance credential, and the public key of the Privacy CA. The process of obtaining evidence of TPM identity has three main phases: Create a new identity, contact a Privacy CA and activate this identity.

Credentials:

A Subsystem or its associated platform may store certificates. These are not essential for the Subsystem itself, but are useful because of their operational advantages when replying to integrity challenges. An Integrity Challenger requires the data in these certificates in order to judge the validity of integrity metrics measured in the platform. So, receiving the data from the TSS relieves the Challenger of the need to fetch it independently.

End of informative comment.

3.3.3.4.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32bit attribute of the TPM object.

End of informative comment.

Definition:

See section 3.3.3.1.1 for definition.

Parameters

See section 3.3.3.1.1 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_TPM_CALLBACK_COLLATEIDENTITY	Application provided data.	Address of callback or NULL (disable	
TSS_TSPATTRIB_TPM_CALLBACK_ACTIVATEIDENTITY	Application provided data.	Address of callback or NULL (disable	

Return Values

See section 3.3.3.1.1 for description.

Remarks

Copyright TCG 2003

3.3.3.4.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32bit attribute of the TPM object

End of informative comment.

Definition:

See section 3.3.3.1.2 for definition.

Parameters

See section 3.3.3.1.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_TPM_CALLBACK_COLLATEIDENTITY	Application provided data.	Address of callback or NULL (disable	
TSS_TSPATTRIB_TPM_CALLBACK_ACTIVATEIDENTITY	Application provided data.	Address of callback or NULL (disable	

Return Values

See section 3.3.3.1.2 for description.

Remarks

3.3.3.4.3 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32bit attribute of the TPM object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 3.3.3.1.3 for definition.

Parameters

See section 3.3.3.1.3 for description.

No Attributes Defined yet

Return Values

See section 3.3.3.1.3 for description.

Remarks

Copyright TCG 2003

3.3.3.4.4 Tspi_GetAttribData

Start of informative comment:

This method gets a non 32bit attribute of the TPM object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 3.3.3.1.4 for definition.

Parameters

See section 3.3.3.1.4 for description.

No Attributes Defined yet

Return Values

See section 3.3.3.1.4 for description.

Remarks

Copyright TCG 2003

3.3.3.4.5 Tspi_TPM_CreateEndorsementKey

Start of informative comment:

This method creates the endorsement key.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_CreateEndorsementKey
(
   TSS_HTPM          hTPM,           // in
   TSS_HKEY          hKey,           // in
   TSS_VALIDATION*   pValidationData // in, out
);
```

Parameters

hTPM
Handle of the TPM object

hKey
Handle of the key object specifying the attributes of the endorsement key to create.

pValidationData
Validation data structure
[IN] Provide externalData information required to compute the signature.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The key information required for creating the endorsement key must be set in the key object by Tspi_SetAttribData( ) before this method is called.

On return the public endorsement key (PUBEK) can be retrieved by Tspi_GetAttribData( ) from the key object.

Copyright TCG 2003

3.3.3.4.6 Tspi_TPM_GetPubEndorsementKey

Start of informative comment:

The function gets the public endorsement key.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetPubEndorsementKey
(
   TSS_HTPM          hTPM,                // in
   TSS_BOOL          fOwnerAuthorized,    // in
   TSS_VALIDATION*   pValidationData,     // in, out
   TSS_HKEY*         phEndorsementPubKey  // out
);
```

Parameters

hTPM
Handle of the TPM object

fOwnerAuthorized
If TRUE, the TPM owner secret must be provided to get the endorsement public key.
If FALSE, no TPM owner secret must be provided to get the endorsement public key.

pValidationData
Validation data structure
[IN] Provide externalData information required to compute the signature.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

phEndorsementPubKey
Receives a handle to a key object representing the endorsement public key.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The public key information of the endorsement key can be retrieved by calling *Tspi_GetAttribData*.

Copyright TCG 2003

3.3.3.4.7 Tspi_TPM_TakeOwnership

Start of informative comment:

This method takes owner ship of the TPM. The process of taking ownership is the procedure whereby the owner inserts a shared secret into the TPM. The Owner of the TPM has the right to perform special operations.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_TakeOwnership
(
   TSS_HTPM    hTPM,                // in
   TSS_HKEY    hKeySRK,             // in
   TSS_HKEY    hEndorsementPubKey   // in
);
```

Parameters

hTPM
Handle of the TPM object

hKeySRK
Handle to the key object representing the SRK (Storage Root Key).

hEndorsementPubKey
Handle to the key object representing the endorsement public key required for encrypting the secret of SRK and the TPM owner secret.
If NULL, the TSP internally queries the TPM for that endorsement public key.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

Copyright TCG 2003

3.3.3.4.8 Tspi_TPM_CollateIdentityRequest

Start of informative comment:

This method creates an identity key, binds it to the label and returns a certificate request package. The privacy CA requires this certificate request to attest the identity key.

Only the Owner of the TPM has the privilege of creating a TPM identity key.

The symmetric session key is required to provide confidentiality of the “TCPA_IDENTITY_REQ” data structure, which should be sent to the Privacy CA chosen by the owner.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_CollateIdentityRequest
(
   TSS_HTPM          hTPM,                     // in
   TSS_HKEY          hKeySRK,                  // in
   TSS_HKEY          hCAPubKey                 // in
   UINT32            ulIdentityLabelLength,    // in
   BYTE*             rgbIdentityLabelData,     // in
   TSS_HKEY          hIdentityKey,             // in
   TSS_ALGORITHM_ID  algID                     // in
   UINT32*           pulTCPAIdentityReqLength, // out
   BYTE**            prgbTCPAIdentityReq       // out
);
```

Parameters

hTPM
Handle of the TPM object

hKeySRK
Handle to the key object representing the SRK (Storage Root Key).

hCAPubKey
Handle to the key object representing the public key of the CA which signs the certificate of the created identity key.

ulIdentityLabelLength
Supplies the length (in bytes) of the *rgbIdentityLabelData* parameter.

rgbIdentityLabelData
Pointer to a memory block containing the identity label, which should be a UNICODE string.

hIdentityKey
Handle of the identity key object

algID
The type of symmetric algorithm to use as required by the Enhanced CA as defined in Algorithm ID Definitions 2.3.2.11.

pulTCPAIdentityReqLength
Receives the length (in bytes) of the *prgbTCPAIdentityReq* parameter.

prgbTCPAIdentityReq
Pointer to the memory block containing the certificate request structure TCPA_IDENTITY_REQ.

Copyright TCG 2003

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_CollateIdentityRequest method allocates a memory block for the requested certificate request data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

The key information required for creating the identity key must be set in the key object by Tspi_SetAttribData( ) before this method is called.

This method assembles all data necessary to request attestation for a Trusted Platform Module identity and exports this data by the output parameter prgbTCPAIdentityReq.

The structure “proof” (of type TCPA_IDENTITY_PROOF) contains fields that a privacy-CA requires in order to decide whether to attest to the TPM identity described by “proof”.

Executing this method the TSS Service Provider performs two encryptions. The first is to symmetrically encrypt the information and the second is to encrypt the symmetric encryption key with an asymmetric algorithm. The symmetric key is a random nonce and the asymmetric key is the public key of the CA that will provide the identity credential.

For reasons of interoperability, publicCAKey SHOULD indicate TSS_ALG_RSA (RSA) with a key length of 2048 bits. CASymKey SHOULD be TSS_ALG_3DES (3DES in CBC mode and PKCS padding as defined in RFC 1423).

Copyright TCG 2003

3.3.3.4.9 Tspi_TPM_ActivateIdentity

Start of informative comment:

This method proofs the credential to be the credential of the identity key and returns the decrypted credential created by the privacy CA for that identity.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_ActivateIdentity
(
   TSS_HTPM    hTPM,                          // in
   TSS_HKEY    hIdentKey,                     // in
   UINT32      ulAsymCAContentsBlobLength,    // in
   BYTE*       rgbAsymCAContentsBlob,         // in
   UINT32      ulSymCAAttestationBlobLength, // in
   BYTE*       rgbSymCAAttestationBlob,       // in
   UINT32*     pulCredentialLength,           // out
   BYTE**      prgbCredential                 // out
);
```

Parameters

hTPM
Handle of the TPM object

hIdentityKey
Handle of the identity key object

ulAsymCAContentsBlobLength
Supplies the length (in bytes) of the *rgbAsymCAContentsBlob* parameter.

rgbAsymCAContentsBlob
Pointer to a memory block containing the encrypted ASYM_CA_CONTENTS data structure got from the privacy CA.

ulSymCAAttestationBlobLength
Supplies the length (in bytes) of the *rgbSymCAAttestationBlob* parameter.

rgbSymCAAttestationBlob
Pointer to a memory block containing the encrypted SYM_CA_ATTESTATION data structure got from the privacy CA.

pulCredetialLength
Receives the length (in bytes) of the *prgbCredential* parameter.

prgbCredetial
Pointer to the memory block containing the decrypted credential data structure TCPA_IDENTITY_CREDENTIAL.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_ActivateIdenty method allocates a memory block for the returned credential data. This memory must be released utilizing the Tspi_Context_FreeMemory method. The TSP MUST support 3DES and MAY support other mechanisms.

Copyright TCG 2003

3.3.3.4.10 Tspi_TPM_ClearOwner

Start of informative comment:

This method clears the TPM ownership.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_ClearOwner
(
   TSS_HTPM    hTPM,          // in
   TSS_BOOL    fForcedClear   // in
);
```

Parameters

hTPM
Handle of the TPM object.

fForcedClear
If FALSE, a clear ownership with proof of the TPM owner secret is done.
If TRUE, a forced clear ownership with proof of physical access is done.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

Please see the manual of your TCG system, to set the physical access state.

3.3.3.4.11 Tspi_TPM_SetStatus

Start of informative comment:

This method modifies the TPM status.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_SetStatus
(
   TSS_HTPM    hTPM,       // in
   TSS_FLAG    statusFlag, // in
   TSS_BOOL    fTpmState   // in
);
```

Parameters

hTPM
Handle of the TPM object.

statusFlag
Flag indicating the status to be set (see section 2.3.2.10 for description).

fTpmState
Status value to set.

Defined Attributes:

Flag	fTpmState	Description
TSS_TPMSTATUS_ DISABLEOWNERCLEAR	Ignored	Permanently disable the TPM owner authorized clearing of TPM ownership. The method Tspi_TPM_ClearOwner( ) with fForcedClear = FALSE is not available any longer. Owner authorization is required.
TSS_TPMSTATUS_ DISABLEFORCECLEAR	Ignored	Prevent temporarily (until next power on) a forced clear of the TPM ownership. The method Tspi_TPM_ClearOwner( ) with fForcedClear = TRUE is temporarily not available.
TSS_TPMSTATUS_ OWNERSETDISABLE	TSS_BOOL	fTpmState = TRUE: Disable the TPM. Owner authorization is required.
TSS_TPMSTATUS_ PHYSICALDISABLE	TSS_BOOL	fTpmState = TRUE: Disable the TPM. Proof of physical access is required.
TSS_TPMSTATUS_ PHYSICALSETDEACTIV ATED	TSS_BOOL	fTpmState = TRUE: Deactivate the TPM. Proof of physical access is required
TSS_TPMSTATUS_ SETTEMPDEACTIVATED	Ignored	Temporarily deactivate (until next power on) the TPM.
TSS_TPMSTATUS_ SETOWNERINSTALL	TSS_BOOL	fTpmState = TRUE: Set the ability to take TPM ownwership utilizing the method Tspi_TPM_TakeOwnership( ). Proof of physical access is required.
TSS_TPMSTATUS_ DISABLEPUBEKREAD	Ignored	Permanently disable the ability to read the endorsement public key without required TPM owner authorizition. The

		method Tspi_TPM_GetPubEndorsementKey ( ) with fOwnerAuthorized = FALSE is not available any longer. Owner authorization is required.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

Please see the manual of your TCG system, to set the physical access state.

For information about which functionality is still available if the TPM is disabled or deactivated, see section 2.3.2.10.

3.3.3.4.12 Tspi_TPM_GetStatus

Start of informative comment:

This method queries the TPM status.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetStatus
(
   TSS_HTPM    hTPM,       // in
   TSS_FLAG    statusFlag, // in
   TSS_BOOL*   pfTpmState  // out
);
```

Parameters

hTPM
Handle of the TPM object.

statusFlag
Flag indicating the status to retrieve (see section 2.3.2.10 for description).

pfTpmState
The value referenced by *pfTpmState* contains the queried status value.

Defined Attributes:

Flags below has descriptions that contain specific information for this function. For other flags see the general description of the flags in TPM Status Flags Definitions 2.3.2.10.

Flag	Description
TSS_TPMSTATUS_ DISABLEOWNERCLEAR	*pfTpmState = TRUE: TPM owner authorized clearing of TPM ownership is permanently disabled. The method Tspi_TPM_ClearOwner( ) with fForcedClear = FALSE is not available any longer.
TSS_TPMSTATUS_ DISABLEFORCECLEAR	*pfTpmState = TRUE: A forced clear of the TPM ownership is temporarily (until next power on) prevented. The method Tspi_TPM_ClearOwner( ) with fForcedClear = TRUE is temporarily not available.
TSS_TPMSTATUS_ DISABLED	*pfTpmState = TRUE: TPM is disabledfTpmState = TRUE.
TSS_TPMSTATUS_PHYS ICALSETDEACTIVATED	*pfTpmState = TRUE: TPM is permantenty deactivated.
TSS_TPMSTATUS_SETT EMPDEACTIVATED	*pfTpmState = TRUE: TPM is temporarily deactivated.
TSS_TPMSTATUS_ SETOWNERINSTALL	*pfTpmState = TRUE: The method Tspi_TPM_TakeOwnership( )can be utilized to take TPM ownership.
TSS_TPMSTATUS_ DISABLEPUBEKREAD	*pfTpmState = TRUE: The ability to read the endorsement public key without required TPM owner authorizition is permanently disabled. The method Tspi_TPM_GetPubEndorsementKey ( ) with fOwnerAuthorized = FALSE is not available any longer. Requires Owner authorization
TSS_TPMSTATUS_	*pfTpmState = TRUE: The TPM owner may create a

ALLOWMAINTENANCE	maintenance archive utilizing the method Tspi_TPM_CreateMaintenanceArchive( )
TSS_TPMSTATUS_ PHSYPRES_LIFTIMELOCK	*pfTpmState = TRUE: The state of either physicalPresenceHWEnable or physicalPresenceCMDEnable cannot be changed for the life of the TPM.
TSS_TPMSTATUS_ PHYSPRES_HWENABLE	*pfTpmState = TRUE: The TPM hardware signal physical presence is enabled to proof physical presence.
TSS_TPMSTATUS_ PHYSPRES_CMDENABLE	*pfTpmState = TRUE: The TPM command TSC_PhysicalPresence indicating physical presence is enabled.
TSS_TPMSTATUS_ CEKP_USED	*pfTpmState = TRUE: The endorsement key pair was created using the methode Tspi_TPM_CreateEndorsementKey( ). *pfTpmState = FALSE: The endorsement key pair was created using a manufacturers process.
TSS_TPMSTATUS_ PHSYPRESENCE	*pfTpmState = TRUE: Software indication that a user is physically present.
TSS_TPMSTATUS_ PHYSPRES_LOCK	*pfTpmState = TRUE: Indicates that changes to the physicalPresence flag are not permitted.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

Because reading any TPM flags requires Owner authorization, this command requires Owner authorization for any flags.

For information about which functionality is still available if the TPM is disabled or deactivated, see section 2.3.2.10.

Copyright TCG 2003

3.3.3.4.13 Tspi_TPM_SelfTestFull

Start of informative comment:

This method performs a self-test of each internal TPM function.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_SelfTestFull
(
   TSS_HTPM    hTPM  // in
);
```

Parameters

hTPM
Handle of the TPM object.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

3.3.3.4.14 Tspi_TPM_CertifySelfTest

Start of informative comment:

This method performs a self-test of each internal TPM function and returns an authenticated value (signature) if the test has passed.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_CertifySelfTest
(
   TSS_HTPM          hTPM,            // in
   TSS_HKEY          hKey,            // in
   TSS_VALIDATION*   pValidationData  // in, out
);
```

Parameters

hTPM
Handle of the TPM object.

hKey
Handle of the signature key object.

pValidationData
Validation data structure
[IN] Provide externalData information required to compute the signature.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

Calculation of hash value for the validation data:
SHA1 hash of the concatenated data of <the null terminated string of “Test Passed”>|| <externalData> || <ordinal>.
See the definition of the ordinal in TCPA 1.1b Main Specification

Copyright TCG 2003

3.3.3.4.15 Tspi_TPM_GetTestResult

Start of informative comment:

The method provides manufacturer specific information regarding the results of the self test

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetTestResult
(
   TSS_HTPM    hTPM,                // in
   UINT32*     pulTestResultLength, // out
   BYTE**      prgbTestResult       // out
);
```

Parameters

hTPM
Handle of the TPM object.

pulTestResultLength
Receives the length (in bytes) of the *prgbTestResult* parameter.

prgbTestResult
Pointer to the memory block containing the TPM manufacturer specific information.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The *Tspi_TPM_GetTestResult* method allocates a memory block for the requested data. This memory must be released utilizing the *Tspi_Context_FreeMemory* method.

Calculation of hash value for the validation data:
Concatination of <the null terminated string of “Test Passed”>|| <externalData> || <ordinal>.
See the definition of the ordinal in TCPA 1.1b Main Specification Part 2

Copyright TCG 2003

3.3.3.4.16 Tspi_TPM_GetCapability

Start of informative comment:

This method provides the capabilities of the TPM.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetCapability
(
   TSS_HTPM    hTPM,                // in
   TSS_FLAG    capArea,             // in
   UINT32      ulSubCapLength,      // in
   BYTE*       rgbSubCap,           // in
   UINT32*     pulRespDataLength,   // out
   BYTE**      prgbRespData         // out
);
```

Parameters

hTPM
Handle of the TPM object

capArea
Flag indicating the attribute to query (see table Defined Attributes).

ulSubCapLength
The length (in bytes) of the *rgbSubCap* parameter.

rgbSubCap
Data indicating the attribute to query (see table Defined Attributes).

pulRespDataLength
Receives the length (in bytes) of the *prgbRespData* parameter.

prgbRespData
Receives pointer to the actual data of the specified attribute (see table Defined Attributes).

Defined Attributes

Capability Area	SubCap Area	Response
TSS_TPMCAP_ORD	Value of the ordinal	Boolean value. TRUE indicates that the TPM supports the ordinal. FALSE indicates that the TPM does not support the ordinal.
TSS_TPMCAP_FLAG	Ignored	Bit map of persistent and volatile flags.
TSS_TPMCAP_ALG	TSS_ALG_XX: A value of TSS Algorithm ID as defined in 0	Boolean value. TRUE indicates that the TPM supports the algorithm, FALSE indicates that the TPM does not support the algorithm.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_PCR	UINT32 value.

Copyright TCG 2003

		Returns the number of PCR registers supported by the TPM
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_DIR	UINT32 value. Returns the number of DIR registers supported by the TPM.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MANUFACTURER	UINT32 value. Returns the Identifier of the TPM manufacturer.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_SLOTS	UINT32 value. Returns the maximum number of 2048 bit RSA keys that the TPM is capable of loading. This MAY vary with time and circumstances.
TSS_TPMCAP_VERSION	Ignored	Returns the TSS_VERSION structure that identifies the version of the TPM.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_GetCapability method allocates a memory block for the requested capability data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

Some capability information can only be requested if owner authorization data is provided by the policy object bound to the TPM object.

Information about capArea and rgbSubCap is transmitted to the TPM without any interpretation by TCS. The TPM will return an appropriate error on wrong values.

3.3.3.4.17 Tspi_TPM_GetCapabilitySigned

Start of informative comment:

NOTE: The TPM function TPM_GetCapabilitySigned that actually performs this functions was found to contain a vulnerability that makes its security questionable therefore its use unadvised. Since the final TPM specification contained this function and products have shipped with this function it is exposed at the TPM layer. However, the TSS Working Group has decided that TSS should not require the implementation of this function for any TSS. However, if a TSS provider should decided to include this function the TSS WG recommends the implementation contained here.

This method provides the capabilities of the TPM and returns a signature to proof the TPM as originator of the capability data.

Definition:

```
TSS_RESULT Tspi_TPM_GetCapabilitySigned
(
    TSS_HTPM          hTPM,               // in
    TSS_HTPM          hKey,               // in
    TSS_FLAG          capArea,            // in
    UINT32            ulSubCapLength,     // in
    BYTE*             rgbSubCap,          // in
    TSS_VALIDATION*   pValidationData,    // in, out
    UINT32*           pulRespDataLength,  // out
    BYTE**            prgbRespData        // out
);
```

Parameters

hTPM
Handle of the TPM object

hKey
Handle of the signature key object

capArea
Flag indicating the attribute to query (see table Defined Attributes).

ulSubCapLength
The length (in bytes) of the *rgbSubCap* parameter.

rgbSubCap
Data indicating the attribute to query (see table Defined Attributes).

pValidationData
Validation data structure
[IN] Provide externalData information required to compute the signature.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

pulRespDataLength
Receives the length (in bytes) of the *prgbRespData* parameter.

prgbRespData
Receives pointer to the actual data of the specified attribute (see table Defined Attributes).

Defined Attributes

Capability Area	SubCap Area	Response
TSS_TPMCAP_ALG	TSS_ALG_XX:	Boolean value. TRUE

	A value of TSS Algorithm ID as defined in 0	indicates that the TPM supports the algorithm, FALSE indicates that the TPM does not support the algorithm.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_PCR	UINT32 value. Returns the number of PCR registers supported by the TPM
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_DIR	UINT32 value. Returns the number of DIR registers supported by the TPM.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MANUFACTURER	UINT32 value. Returns the Identifier of the TPM manufacturer.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_SLOTS	UINT32 value. Returns the maximum number of 2048 bit RSA keys that the TPM is capable of loading. This MAY vary with time and circumstances.
TSS_TPMCAP_VERSION	Ignored	Returns the TSS_VERSION structure that identifies the version of the TPM.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_GetCapabilitySigned method allocates a memory block for the requested capability data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

Calculation of hash value for the validation data:
SHA1 hash of the concatenated data of <respData”>|| <externalData>
See the definition of the ordinal in TCPA 1.1b Main Specification

Information about capArea and rgbSubCap is transmitted to the TPM without any interpretation by TCS. The TPM will return an appropriate error on wrong values.

End of informative comment.

3.3.3.4.18 Tspi_TPM_CreateMaintenanceArchive

Start of informative comment:

This method creates the TPM Manufacturer specific maintenance archive data.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_CreateMaintenanceArchive
(
   TSS_HTPM    hTPM,                  // in
   TSS_BOOL    fGenerateRndNumber,    // in
   UINT32*     pulRndNumberLength,    // out
   BYTE**      prgbRndNumber,         // out
   UINT32*     pulArchiveDataLength,  // out
   BYTE**      prgbArchiveData        // out
);
```

Parameters

hTPM
Handle of the TPM object

fGenerateRndNumber
TRUE: a random number is generated by the TPM and returned.
FALSE: a random number is calculated based on the owner secret.

pulRndNumberLength
Receives the length (in bytes) of the *prgbRndNumber* parameter.
0, if *fGenerateRndNumber* is FALSE.

prgbRndNumber
Receives pointer to the random number data Attributes).
NULL, if *fGenerateRndNumber* is FALSE.

pulArchiveDataLength
Receives the length (in bytes) of the *prgbArchiveData* parameter.

prgbArchiveData
Receives pointer to the archive data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_NOTIMPL
TSS_E_INTERNAL_ERROR

Remarks

The *Tspi_TPM_CreatMaintenaceArchive* method allocates memory blocks for the requested output data. This memory must be released utilizing the *Tspi_Context_FreeMemory* method.

3.3.3.4.19 Tspi_TPM_KillMaintenanceFeature

Start of informative comment:

This method disables the functionality of creating a maintenance archive

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_KillMaintenanceFeature
(
   TSS_HTPM    hTPM  // in
);
```

Parameters

hTPM
Handle of the TPM object

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_NOTIMPL
TSS_E_INTERNAL_ERROR

Remarks

After disabling the functionality of creating a maintenance archive, this functionality can only be enabled again by releasing the TPM ownership.

3.3.3.4.20 Tspi_TPM_LoadMaintenancePubKey

Start of informative comment:

This method loads the public maintenance key into the TPM.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_LoadMaintenancePubKey
(
   TSS_HTPM          hTPM,            // in
   TSS_HKEY          hMaintenanceKey, // in
   TSS_VALIDATION*   pValidationData  // in, out
);
```

Parameters

hTPM
Handle of the TPM object

hMaintenanceKey
Handle of the maintenance key object

pValidationData
Validation data structure
[IN] Provide externalData information required to compute the signature.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_NOTIMPL
TSS_E_INTERNAL_ERROR

Remarks

The maintenance public key can only be loaded once. Subsequent calls to *Tspi_TPM_LoadMaintenancePubKey* will fail.

The key information required for loading the maintenance public key must be set in the key object by *Tspi_SetAttribData()* before this method is called.

If pValidationData != NULL: The caller has to proof the digest by its own.
If pValidationData = NULL: The TSS Service Provider proofs the digest got from the TPM internally.

Calculation of hash value for the validation data:
SHA1 hash of the concatenated data of <maintenance public key>|| <externalData>
See the definition of the ordinal in TCPA 1.1b Main Specification

Copyright TCG 2003

3.3.3.4.21 Tspi_TPM_CheckMaintenancePubKey

Start of informative comment:

This method proofs the maintenance public key.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_CheckMaintenancePubKey
(
   TSS_HTPM          hTPM,            // in
   TSS_HKEY          hMaintenanceKey, // in
   TSS_VALIDATION*   pValidationData  // in, out
);
```

Parameters

hTPM
Handle of the TPM object

hMaintenanceKey
Handle of the maintenance key object

pValidationData
Validation data structure
[IN] Provide externalData information required to compute the signature.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_NOTIMPL
TSS_E_INTERNAL_ERROR

Remarks

If hMaintenanceKey = NULL, pValidationData must not be NULL. The caller has to proof the digest by its own.
If hMaintenanceKey != NULL, pValidationData must be NULL. The TSS Service Provider proofs the digest got from the TPM internally. The key information required for proofing the maintenance public key must be set in the key object by *Tspi_SetAttribData()* before this method is called.

Calculation of hash value for the validation data:
SHA1 hash of the concatenated data of <maintenance public key>|| <externalData>
See the definition of the ordinal in TCPA 1.1b Main Specification

Copyright TCG 2003

3.3.3.4.22 Tspi_TPM_GetRandom

Start of informative comment:

This method gets a random number from the TSS Service Provider utilizing the TPM.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetRandom
(
   TSS_HTPM    hTPM,                // in
   UINT32      ulRandomDataLength,  // in
   BYTE**      prgbRandomData       // out
);
```

Parameters

hTPM
Handle of the TPM object

ulRandomDataLength
Number of random bytes requested.

prgbRandomData
Receives a pointer to memory containing the random data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The maximum length of the random number is 4096 Bytes.

The Tspi_TPM_GetRandom method allocates a memory block for the requested random data. This memory must be released utilizing the Tspi_Context_FreeMemory method

3.3.3.4.23 Tspi_TPM_StirRandom

Start of informative comment:

This method adds entropy to the TPM Random Number Generator

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_StirRandom
(
   TSS_HTPM    hTPM,                // in
   UINT32      ulEntropyDataLength, // in
   BYTE*       rgbEntropyData       // in
);
```

Parameters

hTPM
Handle of the TPM object

ulEntropyDataLength
The length (in bytes) of the *rgbEntropyData* parameter.

rgbEntropyData
Pointer to the entropy data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

Copyright TCG 2003

3.3.3.4.24 Tspi_TPM_AuthorizeMigrationTicket

Start of informative comment:

This method provides the migration ticket required for the migration process.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_AuthorizeMigrationTicket
(
   TSS_HTPM              hTPM,                // in
   TSS_HKEY              hMigrationKey,       // in
   TSS_MIGRATE_SCHEME    migrationScheme   ,  // in
   UINT32*               pulMigTicketLength,  // out
   BYTE**                prgbMigTicket        // out
);
```

Parameters

hTPM
Handle of the TPM object

hMigrationKey
Handle of the key object representing the migration key.

migrationScheme
Flag indicating the migration scheme to be used.

pulMigTicketLength
Receives the length (in bytes) of the *prgbMigTicket* parameter.

prgbMigTicket
Receives a pointer to the memory block containing the migration ticket blob.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_AuthorizeMigrationTicket method allocates a memory block for the requested ticket data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

Copyright TCG 2003

3.3.3.4.25 Tspi_TPM_GetEvent

Start of informative comment:

This method provides a PCR event for a given PCR index and event number.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetEvent
(
   TSS_HTPM       hTPM,          // in
   UINT32         ulPcrIndex,    // in
   UINT32         ulEventNumber, // in
   TSS_PCR_EVENT* pPcrEvent      // out
);
```

Parameters

hTPM
Handle of the TPM object

ulPcrIndex
Index of the PCR to request.

ulEventNumber
Index of the event to request.

pPcrEvent
Receives the PCR event data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

3.3.3.4.26 Tspi_TPM_GetEvents

Start of informative comment:

This method provides a specific number of PCR events for a given index.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetEvents
(
   TSS_HTPM          hTPM,            // in
   UINT32            ulPcrIndex,      // in
   UINT32            ulStartNumber,   // in
   UINT32*           pulEventNumber,  // in, out
   TSS_PCR_EVENT**   prgPcrEvents     // out
);
```

Parameters

hTPM
Handle of the TPM object

ulPcrIndex
Index of the PCR to request.

ulStartNumber
Index of the first event to request.

pulEventNumber
[IN] Number of elements to request.
[OUT] Receives number of returned event data structures in prg*PcrEvents* parameter

prgPcrEvents
Receives a pointer to an array of PCR event data.
If NULL, only the number of elements is returned in pulEventNumber parameter.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_GetEvents method allocates a memory block for the requested event data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

3.3.3.4.27 Tspi_TPM_GetEventLog

Start of informative comment:

This method provides the whole event log.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetEventLog
(
   TSS_HTPM          hTPM,            // in
   UINT32*           pulEventNumber,  // out
   TSS_PCR_EVENT**   prgPcrEvents     // out
);
```

Parameters

hTPM
Handle of the TPM object

pulEventNumber
Receives number of returned event data structures in prg*PcrEvents* parameter

prgPcrEvents
Receives a pointer to an array of PCR event data.
If NULL, only the number of elements is returned in pulEventNumber parameter.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_GetEventLog method allocates a memory block for the requested event data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

Copyright TCG 2003

3.3.3.4.28 Tspi_TPM_Quote

Start of informative comment:

This method quotes a TCG system.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_Quote
(
   TSS_HTPM          hTPM,            // in
   TSS_HKEY          hIdentKey,       // in
   TSS_HPCRS         hPcrComposite,   // in
   TSS_VALIDATION*   pValidationData  // in, out
   );
```

Parameters

hTPM
Handle of the TPM object

hIdentKey
Handle of the signature key object

hPcrComposite
Handle of the PCR composite object

pValidationData
Validation data structure
[IN] Provide externalData information required to compute the signature.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The required information about which PCRs should be quoted must be set in the PcrComposite object before calling this method. On return each element of the collection has its PCRValue set.

The returned signature is computed over the TCPA_QUOTE_INFO structure as defined in the TCPA 1.1b Main Specification.

The Tspi_TPM_GetEventLog method allocates a memory block for the requested event data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

3.3.3.4.29 Tspi_TPM_PcrExtend

Start of informative comment:

This method extends a PCR register and writes the PCR event log.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_PcrExtend
(
   TSS_HTPM       hTPM,               // in
   UINT32         ulPcrIndex,         // in
   UINT32         ulPcrDataLength,    // in
   BYTE*          pbPcrData,          // in
   TSS_PCR_EVENT* pPcrEvent,          // in
   UINT32*        pulPcrValueLength,  // out
   BYTE**         prgbPcrValue        // out
);
```

Parameters

hTPM
Handle of the TPM object

ulPcrIndex
Index of the PCR to extend.

ulPcrDataLength
Parameter contains the length of data to be extended.

ulPcrData
Data pointer to the data blob for the PCR extends operation.

pPcrEvent
Pointer to a TSS_PCR_EVENT structure containing the info for an event entry. If this pointer is NULL no event entry is created and the function only executes an extend operation

pulPcrValueLength
Receives the length (in bytes) of the *prgbPcrValue* parameter.

prgbPcrValue
Receives a pointer to the memory block containing the PCR data after the extend operation.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_PcrExtend method allocates a memory block for the prgbPcrValue data.
This memory must be released utilizing the Tspi_Context_FreeMemory method.

3.3.3.4.30 Tspi_TPM_PcrRead

Start of informative comment:

This methods reads a PCR register.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_PcrRead
(
   TSS_HTPM    hTPM,                // in
   UINT32      ulPcrIndex,          // in
   UINT32*     pulPcrValueLength,   // out
   BYTE**      prgbPcrValue         // out
);
```

Parameters

hTPM
Handle of the TPM object

ulPcrIndex
Index of the PCR to read.

pulPcrValueLength
Receives the length (in bytes) of the *prgbPcrValue* parameter.

prgbPcrValue
Receives a pointer to the memory block containing the PCR data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_PcrRead method allocates a memory block for the prgbPcrValue data. This memory must be released utilizing the Tspi_Context_FreeMemory method

3.3.3.4.31 Tspi_TPM_DirWrite

Start of informative comment:

This method writes a Data Integrity Register.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_DirWrite
(
   TSS_HTPM hTPM,              // in
   UINT32   ulDirIndex,        // in
   UINT32   ulDirDataLength,   // in
   BYTE*    rgbDirData         // in
);
```

Parameters

hTPM
Handle of the TPM object

ulDirIndex
Index of the DIR to write.

ulDirDataLength
The length (in bytes) of the *rgbDirData* parameter

rgbDirData
Pointer to memory containing the data to be written to the DIR.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

3.3.3.4.32 Tspi_TPM_DirRead

Start of informative comment:

This method reads a Data Integrity Register.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_DirRead
(
   TSS_HTPM hTPM,              // in
   UINT32   ulDirIndex,        // in
   UINT32*  pulDirDataLength, // out
   BYTE**   prgbDirData        // out
);
```

Parameters

hTPM
Handle of the TPM object

ulDirIndex
Index of the DIR to read.

pulDirDataLength
Receives the length (in bytes) of the *prgbDirData* parameter.

prgbDirData
Receives a pointer to the memory block containing the the DIR data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_DirRead method allocates a memory block for the prgbDirData data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

Copyright TCG 2003

3.3.3.4.33 Tspi_ChangeAuth

Start of informative comment:

This method changes the authorization data (owner secret) of the TPM object and assigns the TPM object to the policy object.

End of informative comment.

Definition:

See section 3.3.3.1.5 for definition.

Parameters

hObjectToChange
Handle of the TPM object.

hParentObject
NULL

hNewPolicy
Handle to the policy object.

Return Values

See section 3.3.3.1.5 for description.

Remarks

3.3.3.4.34 Tspi_GetPolicyObject

Start of informative comment:

This method returns a policy object currently assigned to the TPM object

End of informative comment.

Definition:

See section 3.3.3.1.7 for definition.

Parameters

hObject
Handle of the object.

policyTypet
Flag indicating the policy type of interest. Must be TSS_POLICY_USAGE.

hPolicy
Receives the handle to the assigned policy object.

Return Values

See section 3.3.3.1.7 for description.

Remarks

3.3.3.5 Tspi_Key Class Definition

3.3.3.5.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32bit attribute of the key object.

End of informative comment.

Definition:

See section 3.3.3.1.1 for definition.

Parameters

See section 3.3.3.1.1 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_KEY_REGISTER	0	TSS_TSPATTRIB_KEYREGISTER_USER	Key is registered automatically in PS.
	0	TSS_TSPATTRIB_KEYREGISTER_SYTEM	Key is registered automatically in PS.
	0	TSS_TSPATTRIB_KEYREGISTER_NO	Key is not registered in PS.
TSS_TSPATTRIB_KEY_INFO	TSS_TSPATTRIB_KEYINFO_USAGE	TSS_KEYUSAGE_XX	TSS Key usage value indicating the usage type of the key as defined in 2.3.2.16.
	TSS_TSPATTRIB_KEYINFO_MIGRATABLE	Boolean value.	If TRUE, key is migratable.
	TSS_TSPATTRIB_KEYINFO_REDIRECTED	Boolean value.	If TRUE, key is redirected. Refer to main spec for details.
	TSS_TSPATTRIB_KEYINFO_VOLATILE	Boolean value.	If TRUE, key is volatile.
	TSS_TSPATTRIB_KEYINFO_AUTHDATAUSAGE	Boolean value.	If TRUE, authorization is required to use the key.
	TSS_TSPATTRIB_KEYINFO_ALOGRITHM	TSS_ALG_XX	TSS algorithm ID value indicating the algorithm of the key as defined in 2.3.2.11.
	TSS_TSPATTRIB_KEYINFO_ENCSCHEME	TSS_KEY_ENCSCHEME_XX	TSS encryption scheme value that the key uses to encrypt information as defined in 2.3.2.20
	TSS_TSPATTRIB_KEYINFO_SIGSCHEME	TSS_KEY_SIGSCHEME_XX	TSS signature scheme value that the key uses to perform digital signatures

			as defined in 0
	TSS_TSPATTRIB_KEYINFO_SIZE		The RSA key size in bits.
	TSS_TSPATTRIB_KEYINFO_KEYFLAGS		Contains the TCG key flag info.
	TSS_TSPATTRIB_KEYINFO_AUTHUSAGE		Direct set of the authDataUsage in the TCG-KeyParams.
TSS_TSPATTRIB_RSAKEY_INFO	TSS_TSPATTRIB_KEYINFO_RSA_PRIMES		The number of prime factors used by the RSA key.

Return Values

See section 3.3.3.1.1 for description.

Remarks

3.3.3.5.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32bit attribute of the key object

End of informative comment.

Definition:

See section 3.3.3.1.2 for definition.

Parameters

See section 3.3.3.1.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_KEY_REGISTER	0	TSS_TSPATTRIB_KEYREGISTER_USER	Key is registered automatically in PS.
	0	TSS_TSPATTRIB_KEYREGISTER_SYTEM	Key is registered automatically in PS.
	0	TSS_TSPATTRIB_KEYREGISTER_NO	Key is not registered in PS.
TSS_TSPATTRIB_KEY_INFO	TSS_TSPATTRIB_KEYINFO_USAGE	TSS_KEYUSAGE_XX	TSS Key usage value indicating the usage type of the key as defined in 2.3.2.16.
	TSS_TSPATTRIB_KEYINFO_MIGRATABLE	Boolean value.	If TRUE, key is migratable.
	TSS_TSPATTRIB_KEYINFO_REDIRECTED	Boolean value.	If TRUE, key is redirected. Refer to main spec for details.
	TSS_TSPATTRIB_KEYINFO_VOLATILE	Boolean value.	If TRUE, key is volatile.
	TSS_TSPATTRIB_KEYINFO_AUTHDATAUSAGE	Boolean value.	If TRUE, authorization is required to use the key.
	TSS_TSPATTRIB_KEYINFO_ALOGRITHM	TSS_ALG_XX	TSS algorithm ID value indicating the algorithm of the key as defined in 2.3.2.11.
	TSS_TSPATTRIB_KEYINFO_ENCSCHEME	TSS_KEY_ENCSCHEME_XX	TSS encryption scheme value that the key uses to encrypt information as defined in 2.3.2.20
	TSS_TSPATTRIB_KEYINFO_SIGSCHEME	TSS_KEY_SIGSCHEME_XX	TSS signature scheme value that the key uses to perform digital signatures as defined in 2.3.2.21

	TSS_TSPATTRIB_KEYINFO_KEYFLAGS		Contains the TCG key flag info.
	TSS_TSPATTRIB_KEYINFO_AUTHUSAGE		Returns the content of the authDataUsage.
TSS_TSPATTRIB_RSAKEY_INFO	TSS_TSPATTRIB_KEYINFO_RSA_KEYSIZE		The size of the RSA key in bits
	TSS_TSPATTRIB_KEYINFO_RSA_PRIMES		The number of prime factors used by the RSA key.

Return Values

See section 3.3.3.1.2 for description.

Remarks

3.3.3.5.3 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32bit attribute of the key object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 3.3.3.1.3 for definition.

Parameters

See section 3.3.3.1.3 for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_KEY_BLOB	TSS_TSPATTRIB_KEYBLOB_BLOB	Key information as a key blob.
	TSS_TSPATTRIB_KEYBLOB_PUBLIC_KEY	Public key information as a key blob.
	TSS_TSPATTRIB_KEYBLOB_PRIVATE_KEY	Encrypted private key information as private key blob.

Return Values

See section 3.3.3.1.3 for description.

Remarks

Copyright TCG 2003

3.3.3.5.4 Tspi_GetAttribData

Start of informative comment:

This method gets a non 32bit attribute of the key object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 3.3.3.1.4 for definition.

Parameters

See section 3.3.3.1.4 for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_KEY_BLOB	TSS_TSPATTRIB_KEYBLOB_BLOB	Key information returned as a key blob.
	TSS_TSPATTRIB_KEYBLOB_PUBLIC_KEY	Public key information as public key blob.
	TSS_TSPATTRIB_KEYBLOB_PRIVATE_KEY	Encrypted private key information as private key blob.
TSS_TSPATTRIB_KEY_INFO	TSS_TSPATTRIB_KEYINFO_VERSION	Version info returned as TSS_VERSION structure
TSS_TSPATTRIB_RSAKEY_INFO	TSS_TSPATTRIB_KEYINFO_RSA_EXPONENT	The public exponent of the key.
	TSS_TSPATTRIB_KEYINFO_RSA_MODULUS	The RSA public modulus.
TSS_TSPATTRIB_KEY_UUID	0	TSS_UUID structure containing the UUID the key is assigned to.
TSS_TSPATTRIB_KEY_PCR	TSS_TSPATTRIB_KEYPCR_DIGEST_ATCREATION	Composite digest value of the PCR values, at the time when the sealing was performed.
	TSS_TSPATTRIB_KEYPCR_DIGEST_ATRELEASE	Composite digest value of the PCR values, at the time when the unsealing should be performed.
	TSS_TSPATTRIB_KEYPCR_SELECTION	A bit map that indicates if a PCR is active or not.

Return Values

See section 3.3.3.1.4 for description.

Remarks

3.3.3.5.5 Tspi_Key_LoadKey

Start of informative comment:

The Tspi_Key_LoadKey method loads the key blob of the object into the TPM. The TPM will unwrap the key when it is loaded.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_LoadKey
(
   TSS_HKEY    hKey,          // in
   TSS_HKEY    hUnwrappingKey // in
);
```

Parameters

hKey
Handle of the key object to load.

hUnwrappingKey
Handle of the key which should be used to unwrap the key addressed by *hKey*.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The key information for the key to load is set by calling *Tspi_SetAttribData*. The key blob addressed by *hUnwrappingKey* must have been loaded into the TPM previously.

Copyright TCG 2003

3.3.3.5.6 Tspi_Key_UnloadKey

Start of informative comment:

The Tspi_Key_UnloadKey method unloads the key reference by the key object from the TPM. This call will result in a TPM_EvictKey operation for the specified key.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_UnloadKey
(
   TSS_HKEY    hKey           // in
);
```

Parameters

hKey
Handle of the key object to unload.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_Key_UnloadKey method unloads the key reference by the key object from the TPM. This call will result in a TPM_EvictKey operation for the specified key.

3.3.3.5.7 Tspi_Key_GetPubKey

Start of informative comment:

This method returns the public key of the key object.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_GetPubKey
(
   TSS_HKEY hKey,              // in
   UINT32*  pulPubKeyLength    // out
   BYTE**   prgbPubKey);       // out
);
```

Parameters

hKey
Handle of the key object.

pulPubKeyLength
Receives the length (in bytes) of the *prgbPubKey* parameter.

prgbPubKey
Receives a pointer to the memory block containing the public key blob retrieved for the key object referenced by *hKey*.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

Copyright TCG 2003

3.3.3.5.8 Tspi_Key_CertifyKey

Start of informative comment:

This method signs a public key.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_CertifyKey
(
   TSS_HKEY          hKey,            // in
   TSS_HKEY          hCertifyingKey,  // in
   TSS_VALIDATION*   pValidationData  // in, out
);
```

Parameters

hKey
Handle of the key object where the public key should be signed.

hCertifyingKey
Handle to the certifying key used to sign the key addressed by *hKey*.

pValidationData
Pointer to a structure of the type *TSS_VALIDATION*. After successful completion of the call the member *rgbValidationData* of this structure contains the signature data of the command. The member prgbData of the structure points to a buffer containing a TCPA_CERTIFY_INFO data stream as specified within the TCPA 1.1b Main Specification.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

This method calls the TPM command *TPM_CertifyKey* where the public key information to be signed is addressed by *hKey* and the signing key is addressed by *hCertifyingKey*. Memory allocated by this method for the members of the structure *TSS_VALIDATION* must be deallocated by calling *Tspi_Context_FreeMemory*.

Copyright TCG 2003

3.3.3.5.9 Tspi_Key_CreateKey

Start of informative comment:

The *Tspi_Key_CreateKey* method creates a key pair within the TPM and wraps it with the key addressed by *hWrappingKey*.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_CreateKey
(
   TSS_HKEY    hKey,          // in
   TSS_HKEY    hWrappingKey,  // in
   TSS_HPCRS   hPcrComposite  // in, may be NULL
);
```

Parameters

hKey
Handle of the key object to create.

hWrappingKey
Handle to the key used to wrap the newly created key.

hPcrComposite
Handle to an object of the type *Tspi_PcrComposite*. If the value of the handle doesn't equal to NULL, the newly created key will be bound to the PCR values described with this object.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_KEY_NO_MIGRATION_POLICY
TSS_E_INTERNAL_ERROR

Remarks

This method calls the TPM command *TPM_CreateWrapKey*. If a PCR composite object is provided at the parameter *hPcrComposite* (*hPcrComposite* is not NULL) the created key blob is bound to this PCR values. The key object addressed by *hKey* must contain the key information needed for key creation, previously set with *Tspi_SetAttribXXX()*. On return the object pointed to by *hKey* contains the wrapped key blob, which can be retrieved by calling *GetAttribData()*.

When a migratable key will be created the command requires a migration secret provided with a policy. If there was no secret set for this policy it will be retrieved via pop-up or callback function.

Copyright TCG 2003

3.3.3.5.10 Tspi_Key_WrapKey

Start of informative comment:

This method wraps a key with the key addressed by *hWrappingKey*.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_WrapKey
(
   TSS_HKEY    hKey,          // in
   TSS_HKEY    hWrappingKey,  // in
   TSS_HPCRS   hPcrComposite  // in, may be NULL
);
```

Parameters

hKey
Handle of the key object to create.

hWrappingKey
Handle to the key used to wrap the key addressed by *hKey*.

hPcrComposite
Handle to an object of the type *Tspi_PcrComposite*. If the value of the handle doesn't equal to NULL, the key addressed by *hKey* will be bound to the PCR values described with this object.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

If a PCR composite object is provided at the parameter *hPcrComposite* (*hPcrComposite* is not NULL) the created key blob is bound to this PCR values. The key object addressed by *hKey* must contain the key information required for key creation, previously set with *Tspi_SetAttribXXX()*. On return the object pointed to by *hKey* contains the wrapped key blob, which can be retrieved by calling Tspi_*GetAttribData()*.

3.3.3.5.11 Tspi_Key_CreateMigrationBlob

Start of informative comment:

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_CreateMigrationBlob
(
   TSS_HKEY hKeyToMigrate,          // in
   TSS_HKEY hParentKey,             // in
   UINT32   ulMigTicketLength,      // in
   BYTE*    rgbMigTicket            // in
   UINT32*  pulRandomLength,        // out
   BYTE**   prgbRandom,             // out
   UINT32*  pulMigrationBlobLength, // out
   BYTE**   prgbMigrationBlob       // out
);
```

Parameters

hKeyToMigrate
Handle of the key object to migrate.

hParentKey
Handle to the parent key related to the key addressed by *hKeyToMigrate*.

ulMigTicketLength
The length (in bytes) of the *rgbMigTicket* parameter

rgbMigTicket
Pointer to memory containing the migration ticket (migration public key and its authorization digest). This data previously have been returned by the method *Tspi_TPM_AuthorizeMigrationTicket()*.

pulRandomLength
On successful completion this parameter returns the random data length returned at the parameter *prgbRandom*.

prgbRandom
On successful completion this parameter returns the random data.

pulMigrationBlobLength
On successful completion this parameter returns the length of the migration blob data returned at the parameter *prgbMigrationBlob*.

prgbMigrationBlob
On successful completion this parameter returns the migration data blob.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_KEY_NO_MIGRATION_POLICY
TSS_E_INTERNAL_ERROR

Copyright TCG 2003

Remarks

The function returns a key blob containing an encrypted part, which will be different depending on the migration scheme indicated within the migration ticket previously created by the method *Tspi_TPM_AuthorizeMigrationTicket()*.

Migration scheme: TSS_MS_REWRAP

The returned key blob can be loaded into another TPM without further actions. No random number is returned.

Migration scheme: TSS_MS_MIGRATE

The method returns a random number and a migration blob which must be converted by calling *Tspi_Key_ConvertMigrationBlob()*.

This method calls the TPM command TPM_CreateMigrationBlob().

The Tspi_Key_CreateMigrationBlob method allocates a memory block for the allocated data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

Copyright TCG 2003

3.3.3.5.12 Tspi_Key_ConvertMigrationBlob

Start of informative comment:

This method takes the migration blob built by Tspi_Key_CreateMigrationBlob using the migration scheme TSS_MS_MIGRATE and creates a normal wrapped key. The resulting normal wrapped key blob is stored in the instance associated with hKeyToMigrate and may be retrieved from that instance by Tspi_GetAttribData().

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_ConvertMigrationBlob
(
   TSS_HKEY hKeyToMigrate,          // in
   TSS_HKEY hParentKey,             // in
   UINT32   ulRandomLength,         // in
   BYTE*    rgbRandom,              // in
   UINT32   ulMigrationBlobLength,  // in
   BYTE*    rgbMigrationBlob        // in
);
```

Parameters

hKeyToMigrate
Handle of the key object to convert.

hParentKey
Handle to the parent key related to the key addressed by *hKeyToMigrate*.

ulRandomLength
Length of random data provided at the parameter *rgbRandom*.

rgbRandom
Random data as returned together with the migration blob by the method *Tspi_Key_CreateMigrationBlob()*.

ulMigrationBlobLength
Length of the migration blob data provided at the parameter *rgbMigrationBlob*.

rgbMigrationBlob
Migration blob data as returned by a previously called method *Tspi_Key_CreateMigrationBlob()*.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

Copyright TCG 2003

3.3.3.5.13 Tspi_ChangeAuth

Start of informative comment:

This method changes the authorization data (owner secret) of the key object and assigns the key object to the policy object.

End of informative comment.

Definition:

See section 3.3.3.1.5 for definition.

Parameters

hObjectToChange
Handle of the key object.

hParentObject
Handle to the parent object related with the object addressed by *hObjectToChange*.

hNewPolicy
Handle to the new policy object.

Return Values

See section 3.3.3.1.5 for description.

Remarks

The TSP's key manager SHALL replace key blobs associated with *hObjectToChange with the new key blobs that contain the new authorization value.*

Changing the owner secret of the SRK (Storage Root Key) the parameter *hObjectToChange* must refer to the SRK object and the parameter *hParentObject* must refer to the TPM object

Copyright TCG 2003

3.3.3.5.14 Tspi_ChangeAuthAsym

Start of informative comment:

This method changes the authorization data (secret) of the key object utilizing the asymmetric change protocol and assigns the key object to new policy object.

This method changes the authorization data of the key object ensuring that the parent of the key object does not get knowledge of the new secret.

End of informative comment.

Definition:

See section 3.3.3.1.6 for definition.

Parameters

See section 3.3.3.1.6 for description.

Return Values

See section 3.3.3.1.6 for description.

Remarks

Copyright TCG 2003

3.3.3.5.15 Tspi_GetPolicyObject

Start of informative comment:

This method returns a policy object currently assigned to the key object

End of informative comment.

Definition:

See section 3.3.3.1.7 for definition.

Parameters

See section 3.3.3.1.7 for description.

Return Values

See section 3.3.3.1.7 for description.

Remarks

See section 3.3.3.1.7 for description.

3.3.3.6 Tspi_Hash Class Definition

3.3.3.6.1 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32bit attribute of the data object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 3.3.3.1.3 for definition.

Parameters

See section 3.3.3.1.3 for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_HASH_IDENTIFIER	0	Sets the length and data for the hash algorithm identifier (Hash object created with type is HASH_ALG_OTHER)

Return Values

See section 3.3.3.1.3 for description.

Remarks

3.3.3.6.2 Tspi_Hash_Sign

Start of informative comment:

This method signs the hash data of the object with the provided signing key. Note, that while the parameter hHash implies this is a hash value, it is, in fact, just an opaque value assigned by the caller that created the TSS_HHASH object.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Hash_Sign
(
   TSS_HHASH   hHash,               // in
   TSS_HKEY    hKey,                // in
   UINT32*     pulSignatureLength,  // out
   BYTE**      prgbSignature        // out
);
```

Parameters

hHash
Handle to the hash object instance which hash value should be signed.

hKey
Handle to the key object which should be used for the signature.

pulSignatureLength

On successful completion this parameter indicates the length of the signature data returned at the parameter *prgbSignature*.

prgbSignature
On successful completion this parameter points to the signature data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_HASH_INVALID_LENGTH
TSS_E_HASH_NO_DATA
TSS_E_HASH_NO_IDENTIFIER
TSS_E_INTERNAL_ERROR

Remarks

The data to be signed must be set at the hash instance associated with *hHash* by calling *Tspi_Hash_SetHashValue()* or *Tspi_Hash_UpdateHash()*.

The Tspi_Hash_Sign method allocates a memory block for the prgbSignature data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

3.3.3.6.3 Tspi_Hash_VerifySignature

Start of informative comment:

This method verifies the hash value of the hash object with a given signature

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Hash_VerifySignature
(
   TSS_HHASH   hHash,               // in
   TSS_HKEY    hKey,                // in
   UINT32      ulSignatureLength,   // in
   BYTE*       rgbSignature         // in
);
```

Parameters

hHash
Handle to the hash object instance which hash value should be verified.

hKey
Handle to the key object which should be used for the signature verification.

ulSignatureLength
This parameter indicates the length of the signature data provided at the parameter *rgbSignature*.

rgbSignature
This parameter points to the signature data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_HASH_INVALID_LENGTH
TSS_E_HASH_NO_DATA
TSS_E_INVALID_SIGSCHEME
TSS_E_INTERNAL_ERROR

Remarks

The data to be verified must be set at the hash instance associated with *hHash* by calling *Tspi_Hash_SetHashValue()* or *Tspi_Hash_UpdateHash()*.

Copyright TCG 2003

3.3.3.6.4 Tspi_Hash_SetHashValue

Start of informative comment:

This method sets the hash value of the hash object.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Hash_SetHashValue
(
   TSS_HHASH   hHash,               // in
   UINT32      ulHashValueLength,   // in
   BYTE*       rgbHashValue         // in
);
```

Parameters

hHash
Handle to the hash object instance which hash value should be set.

ulHashValueLength
This parameter indicates the length of the hash value data provided at the parameter *rgbHashValue*.

rgbHashValue
This parameter points to the hash value data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_HASH_INVALID_LENGTH
TSS_E_HASH_NO_DATA
TSS_E_INTERNAL_ERROR

Remarks

If the object was created with the flag TSS_HASH_OTHER then the hash algorithm identifier has to be set by calling *Tspi_SetAttribData()* to perform the sign operation.

Copyright TCG 2003

3.3.3.6.5 Tspi_Hash_GetHashValue

Start of informative comment:

This method retunes the hash value of the hash object.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Hash_GetHashValue
(
   TSS_HHASH   hHash,               // in
   UINT32*     pulHashValueLength,  // out
   BYTE**      prgbHashValue        // out
);
```

Parameters

hHash
Handle to the hash object instance which hash value should be returned.

pulHashValueLength
On successful completion this parameter indicates the length of the hash data returned at the parameter *prgbHashValue*.

prgbSignature
On successful completion this parameter points to the hash data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_HASH_INVALID_LENGTH
TSS_E_HASH_NO_DATA
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_Hash_GetHashValue method allocates a memory block for the prgbHashValue data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

3.3.3.6.6 Tspi_Hash_UpdateHashValue

Start of informative comment:

This method updates the hash object with new data.

Supported Hash Algorithm:

- SHA1

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Hash_UpdateHashValue
(
   TSS_HHASH   hHash,          // in
   UINT32      ulDataLength,   // in
   BYTE*       rgbData         // in
);
```

Parameters

hHash
Handle to the hash object instance which hash value should updated.

ulDataLength
This parameter indicates the length of the data provided at the parameter *rgbData*.

rgbData
This parameter points to the data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_HASH_INVALID_LENGTH
TSS_E_HASH_NO_DATA
TSS_E_INTERNAL_ERROR

Remarks

The object can't be modified after *Tspi_Hash_SetHashValue()*, *Tspi_Hash_GetHashValue()*, *Tspi_Hash_Sign()* or *Tspi_Hash_VerifySignature()* have been called on it. If the object was created with the flag TSS_HASH_OTHER then this method will return an error.

Copyright TCG 2003

3.3.3.7 Tspi_Data Class Definition

3.3.3.7.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32bit attribute of the data object.

End of informative comment.

Definition:

See section 3.3.3.1.1 for definition.

Parameters

See section 3.3.3.1.1 for description.

No Attributes Defined yet

Return Values

See section 3.3.3.1.1 for description.

Remarks

Copyright TCG 2003

3.3.3.7.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32bit attribute of the data object

End of informative comment.

Definition:

See section 3.3.3.1.2 for definition.

Parameters

See section 3.3.3.1.2 for description.

No Attributes Defined yet

Return Values

See section 3.3.3.1.2 for description.

Remarks

3.3.3.7.3 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32bit attribute of the data object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 3.3.3.1.3 for definition.

Parameters

See section 3.3.3.1.3 for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_ENCDATA_BLOB	TSS_TSPATTRIB_ENCDATABLOB_BLOB	Data blob that represents the encrypted data depending on its type (seal, bind or legacy).

Return Values

See section 3.3.3.1.3 for description.

Remarks

3.3.3.7.4 Tspi_GetAttribData

Start of informative comment:

This method gets a non 32bit attribute of the data object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 3.3.3.1.4 for definition.

Parameters

See section 3.3.3.1.4 for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_ENCDATA_BLOB	TSS_TSPATTRIB_ENCDATABLOB_BLOB	Data blob that represents the encrypted data depending on its type (seal, bind or legacy).
TSS_TSPATTRIB_ENCDATA_PCR	TSS_TSPATTRIB_ENCDATAPCR_DIGEST_ATCREATION	Composite digest value of the PCR values, at the time when the sealing was performed.
	TSS_TSPATTRIB_ENCDATAPCR _DIGEST_ATRELEASE	Composite digest value of the PCR values, at the time when the unsealing should be performed.
	TSS_TSPATTRIB_ENCDATAPCR _SELECTION	A bit map that indicates if a PCR is active or not.

Return Values

See section 3.3.3.1.4 for description.

Remarks

Copyright TCG 2003

3.3.3.7.5 Tspi_Data_Bind

Start of informative comment:

This method encrypts a data blob in a manner that is decryptable by Tspi_Data_Unbind. The data blob is encrypted using a public key operation with the key addressed by the given encryption key object.

To bind data that is larger than the RSA public key modulus it is the responsibility of the caller to perform the blocking and subsequent combination of data.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Data_Bind
(
   TSS_HENCDATA   hEncData,      // in
   TSS_HKEY       hEncKey,       // in
   UINT32         ulDataLength,  // in
   BYTE*          rgbDataToBind  // in
);
```

Parameters

hEncData
Handle of the data object which contains the encrypted data on successful completion of the command.

hEncKey
Handle to the key object addressing the public key which is used to encrypt the data.

ulDataLength
The length (in bytes) of the *rgbDataToBind* parameter.

rgbDataToBind
Pointer to memory containing the data to be encrypted.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INVALID_ENCSCHEME
TSS_E_ENC_INVALID_LENGTH
TSS_E_ENC_NO_DATA
TSS_E_ENC_INVALID_TYPE
TSS_E_INTERNAL_ERROR

Remarks

The bound data blob is stored in the data object addressed by hEncData and can be exported from that object by GetAttribData( ). The caller gets this exported encrypted data blob according the rules of TCG in order to ensure interoperability between different TCG systems.

The Caller of the Tspi_Data_Bind method should perform validations that the public key presented to it is from a valid TPM.

Copyright TCG 2003

3.3.3.7.6 Tspi_Data_Unbind

Start of informative comment:

This method takes the encrypted data blob that was exported from the data object used in the Tspi_Data_Bind command and decrypts it

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Data_Unbind
(
   TSS_HENCDATA   hEncData,               // in
   TSS_HKEY       hKey                    // in
   UINT32*        pulUnboundDataLength,   // out
   BYTE**         prgbUnboundData         // out
);
```

Parameters

hEncData
Handle of the data object which addresses the encrypted data.

hKey
Handle of the key object addressing the private key which is used to decrypt the data.

pulDataLength
Receives the length (in bytes) of the *prgbUnboundData* parameter.

prgbUnboundData
On successful completion of the command, this parameter points to a buffer containing the plaintext data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_ENC_INVALID_LENGTH
TSS_E_ENC_NO_DATA
TSS_E_ENC_INVALID_TYPE
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_Data_Unbind method allocates a memory block for the decrypted data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

The encrypted data blob must be imported to the object addressed by hEncData by SetAttribData( ) before calling this method.

This method operates on a block-by-block basis, and has no notion of any relation between one block and another.

3.3.3.7.7 Tspi_Data_Seal

Start of informative comment:

This method encrypts a data blob in a manner that is only decryptable by Tspi_Data_Unseal on the same system. The data blob is encrypted using a public key operation with the nonmigratable key addressed by the given encryption key object.

Additionally the Tspi_Data_Seal operation allows software to explicitly state the future “trusted” configuration that the platform must be in for the encrypted data to be revealed and implicitly includes the relevant Platform Configuration Register (PCR) values when the Tspi_Data_Seal operation was performed. Which PCR registers are going to be part of the seal operation is specified by the PCR composite object addressed by hPcrComposite.

If the Tspi_Data_Unseal operation succeeds, proof of the platform configuration that was in effect when the Tspi_Data_Seal operation was performed is returned to the caller, as well as the secret data. This proof may, or may not, be of interest. If the SEALed secret is used to authenticate the platform to a third party, a caller is normally unconcerned about the state of the platform when the secret was SEALed, and the proof may be of no interest. On the other hand, if the SEALed secret is used to authenticate a third party to the platform, a caller is normally concerned about the state of the platform when the secret was SEALed. Then the proof is of interest.

For example, if SEAL is used to store a secret key for a future configuration (probably to prove that the platform is a particular platform that is in a particular configuration), the only requirement is that that key can be used only when the platform is in that future configuration. Then there is no interest in the platform configuration when the secret key was SEALed. An example of this case is when SEAL is used to store a network authentication key.

On the other hand, suppose an OS contains an encrypted database of users allowed to log on to the platform. The OS uses a SEALED blob to store the encryption key for the user-database. However, the nature of SEAL is that *any* SW stack can SEAL a blob for any other software stack. Hence the OS can be attacked by a second OS replacing both the SEALED-blob encryption key, *and* the user database itself, allowing untrusted parties access to the services of the OS. To thwart such attacks, SEALED blobs include the *past* SW configuration. Hence, if the OS is concerned about such attacks, it may check to see whether the past configuration is one that is known to be trusted.

To seal data that is larger than the RSA public key modulus it is the responsibility of the caller to perform the blocking and subsequent combination of data.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Data_Seal
(
   TSS_HENCDATA   hEncData,      // in
   TSS_HKEY       hEncKey,       // in
   UINT32         ulDataLength,  // in
   BYTE*          rgbDataToSeal, // in
   TSS_HPCRS      hPcrComposite  // in
);
```

Parameters

hEncData
Handle of the data object which contains the sealed data on successful completion of the command.

hEncKey
Handle to the key object addressing the nonmigratable key which is used to encrypt the data.

Copyright TCG 2003

ulDataLength
The length (in bytes) of the *rgbDataToSeal* parameter.

rgbDataToSeal
Pointer to memory containing the data to be encrypted.

hPcrComposite
Handle of the PCR Composite object specifying the PCRs which are part of the sealed data blob.
Set to NULL, if the encrypted data should only be bound to the system and PCRs are not of interest.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_ENC_INVALID_LENGTH
TSS_E_ENC_NO_DATA
TSS_E_ENC_INVALID_TYPE
TSS_E_INTERNAL_ERROR

Remarks

The sealed data blob is stored in the data object addressed by hEncData and can be exported from that object by GetAttribData( ). The caller gets this exported encrypted data blob according the rules of TCG.

The information about the used PCRs must be set in the PCR composite object addressed by hPcrComposite before calling this methode. This value MUST be set to NULL, if PCR values are not of interest.

3.3.3.7.8 Tspi_Data_Unseal

Start of informative comment:

This method reveals data encrypted by Tspi_Data_Seal only if it was encrypted on the same platform and the current configuration (as defined by the named PCR contents of the encrypted data blob) is the one named as qualified to decrypt it. This is internally proofed and guaranteed by the TPM.

If the Tspi_Data_Unseal operation succeeds, proof of the platform configuration that was in effect when the Tspi_Data_Seal operation was performed is returned to the caller, as well as the secret data. This proof may, or may not, be of interest. If the SEALed secret is used to authenticate the platform to a third party, a caller is normally unconcerned about the state of the platform when the secret was SEALed, and the proof may be of no interest. On the other hand, if the SEALed secret is used to authenticate a third party to the platform, a caller is normally concerned about the state of the platform when the secret was SEALed. Then the proof is of interest.

For example, if SEAL is used to store a secret key for a future configuration (probably to prove that the platform is a particular platform that is in a particular configuration), the only requirement is that that key can be used only when the platform is in that future configuration. Then there is no interest in the platform configuration when the secret key was SEALed. An example of this case is when SEAL is used to store a network authentication key.

On the other hand, suppose an OS contains an encrypted database of users allowed to log on to the platform. The OS uses a SEALED blob to store the encryption key for the user-database. However, the nature of SEAL is that *any* SW stack can SEAL a blob for any other software stack. Hence the OS can be attacked by a second OS replacing both the SEALED-blob encryption key, *and* the user database itself, allowing untrusted parties access to the services of the OS. To thwart such attacks, SEALED blobs include the *past* SW configuration. Hence, if the OS is concerned about such attacks, it may check to see whether the past configuration is one that is known to be trusted.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Data_Unseal
(
   TSS_HENCDATA   hEncData,               // in
   TSS_HKEY       hKey,                   // in
   UINT32*        pulUnsealedDataLength,  // out
   BYTE**         prgbUnsealedData        // out
);
```

Parameters

hEncData
Handle of the data object which contains the sealed data.

hKey
Handle to the key object addressing the nonmigratable key which is used to decrypt the data.

pulUnsealedDataLength
The length (in bytes) of the *prgbUnsealedData* parameter.

prgbUnsealedData
On successful completion of the command, this parameter points to a buffer containing the plaintext data.

Copyright TCG 2003

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_ENC_INVALID_LENGTH
TSS_E_ENC_NO_DATA
TSS_E_ENC_INVALID_TYPE
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_Data_Unseal method allocates a memory block for the decrypted data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

The sealed data blob must be imported to the object addressed by hEncData utilizing SetAttribData( ) before calling this method.

The platform configuration status at the time when the Tspi_Data_Seal method has sealed the data can be retrieved from the object addressed by hEncData utilizing GetAttribData( )after this methode was called.

This method operates on a block-by-block basis, and has no notion of any relation between one block and another.

Copyright TCG 2003

3.3.3.7.9 Tspi_ChangeAuth

Start of informative comment:

This method changes the authorization data (secret) of the data object and assigns the data object to the policy object.

End of informative comment.

Definition:

See section 3.3.3.1.5 for definition.

Parameters

See section 3.3.3.1.5 for description.

Return Values

See section 3.3.3.1.5 for description.

Remarks

Copyright TCG 2003

3.3.3.7.10 Tspi_ChangeAuthAsym

Start of informative comment:

This method changes the authorization data (secret) of the data object utilizing the asymmetric change protocol and assigns the data object to the policy object.

This method changes the authorization data of the data object ensuring that the parent of the data object does not get knowledge of the new secret.

End of informative comment.

Definition:

See section 3.3.3.1.6 for definition.

Parameters

See section 3.3.3.1.6 for description.

Return Values

See section 3.3.3.1.6 for description.

Remarks

Copyright TCG 2003

3.3.3.7.11 Tspi_GetPolicyObject

Start of informative comment:

This method returns a policy object currently assigned to the data object

End of informative comment.

Definition:

See section 3.3.3.1.7 for definition.

Parameters

See section 3.3.3.1.7 for description.

Return Values

See section 3.3.3.1.7 for description.

Remarks

See section 3.3.3.1.7 for description.

Copyright TCG 2003

3.3.3.8 Tspi_PcrComposite Class Definition

3.3.3.8.1 Tspi_PcrComposite_SelectPcrIndex

Start of informative comment:

This method selects a PCR index inside a PCR composite object. The PCR composite object must be created with the function Tspi_Context_CreateObject(). An exampled for the usage is the selection of PCR registers before calling Tspi_TPM_Quote().

End of informative comment.

Definition:

```
TSS_RESULT Tspi_PcrComposite_SelectPcrIndex
(
   TSS_HPCRS   hPcrComposite, // in
   UINT32      ulPcrIndex     // in
);
```

Parameters

hPcrComposite
Handle to the PCR composite object instance where the index should be selected.

ulPcrIndex
This parameter indicates the index of the PCR to select.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The PCR composite object must have been created by the method *Tspi_Context_CreateObject()*. An example for the usage of this method is the selection of PCR registers prior to calling Tspi_TPM_Quote(). Multiple PCRs with different indexes can be selected by calling the method multiple times on the same PCR composite object.

3.3.3.8.2 Tspi_PcrComosite_SetPcrValue

Start of informative comment:

This method sets the digest for a given PCR index inside the PCR composite object.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_PcrComposite_SetPcrValue
(
   TSS_HPCRS   hPcrComposite,    // in
   UINT32      ulPcrIndex,       // in
   UINT32      ulPcrValueLength, // in
   BYTE*       rgbPcrValue       // in
);
```

Parameters

hPcrComposite
Handle to the PCR composite object instance where a PCR value should be set.

ulPcrIndex
This parameter indicates the index of the PCR to set.

ulPcrValueLength
The length (in bytes) of the *rgbPcrValue* parameter

rgbPcrValue
Pointer to memory containing the actual value which should be set for the PCR indicated by *ulPcrIndex*.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

An exampled for the usage is the preparation of a PCR composite object before calling *Tspi_Key_CreateKey()*. The PCR composite object must have been created by the method *Tspi_Context_CreateObject()*.Multiple PCRs with different indexes can be set by calling this method multiple times on the same PCR composite object.

3.3.3.8.3 Tspi_PcrComposite_GetPcrValue

Start of informative comment:

This method returns the digest value of a given PCR index inside a PCR composite object.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_PcrComposite_GetPcrValue
(
   TSS_HPCRS   hPcrComposite,       // in
   UINT32      ulPcrIndex,          // in
   UINT32*     pulPcrValueLength,   // out
   BYTE**      prgbPcrValue         // out
);
```

Parameters

hPcrComposite
Handle to the PCR composite object instance from which the PCR value should be returned.

ulPcrIndex
This parameter indicates the index of the PCR to read.

pulPcrValueLength
Receives the length (in bytes) of the *prgbPcrValue* parameter.

prgbPcrValue
After successful completion this parameter receives a pointer to the memory block containing the PCR value of the PCR indicated by *ulPcrIndex*.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

An example for the usage of this method is for retrieving the value of a PCR after a *Tspi_TPM_Quote()* call. Multiple PCRs vales for different indexes can be retrieved by calling this method multiple times on a PCR composite object.

The Tspi_PcrComposite_GetPcrValue method allocates a memory block for the prgbPcrValue data.This memory must be released utilizing the Tspi_Context_FreeMemory method.

3.3.4 Callback Function Definitions

3.3.4.1 Tspicb_CallbackHMACAuth

Start of informative comment:

This method is called each time when authorized TPM commands are called and the callback mechanism is set in the assigned policy object. In functions where there is only one object with a usage policy, this usage policy object will have the necessary attribute for the callback. In some cases, the TSP may internally use an OSAP session, as required by the function. Section Tspicb_CallbackXorEnc 3.3.4.2 defines which functions *require* OSAP and also defines which policy must hold the callback pointer for the xor encrypt. When one of these functions is called, then the same policy object will hold the pointer for this function as well if desired.

When a change of authorization is being done, then it is possible that this function may be registered to two different policy objects. Before a change of authorization, the callback, when required, must be registered to the usage policy of the existing object. For the verify, the callback should be registered to the *new* usage policy if a callback is required.

When the parameter ReturnOrVerify is TRUE, the callback must calculate the HMAC data, if FALSE the callback must verify the HMAC data returned from the TPM.

The two pointers rgbNonceEvenOSAP, rgbNonceOddOSAP are only valid, if the service provider uses an internally OSAP session. In this case the shared secret must be used for the HMAC.

End of informative comment.

Definition:

```
TSS_RESULT Tspicb_CallbackHMACAuth
(
   PVOID          lpAppData         // in
   TSS_HOBJECT    hAuthorizedObject // in
   TSS_BOOL       ReturnOrVerify    // in
   UINT32         ulPendingFunction // in
   TSS_BOOL       ContinueUse       // in
   UINT32         ulSizeNonces      // in
   BYTE*          rgbNonceEven      // in
   BYTE*          rgbNonceOdd,      // in
   BYTE*          rgbNonceEvenOSAP  // in
   BYTE*          rgbNonceOddOSAP   // in
   UINT32         ulSizeDigestHmac  // in
   BYTE*          rgbParamDigest    // in
   BYTE*          rgbHmacData       // in, out
);
```

Parameters

lpAppData
Pointer to application provided data as provided on registration of callback function

hAuthorizedObject
Handle to the object authorization is required

ReturnOrVerify
Flag indicating authorization or verification is required.

(CalculateHMACData)

TRUE: the callback must calculate the HMAC data

Copyright TCG 2003

FALSE: callback must verify the HMAC data returned from the TPM.

ulPendingFunction
Ordinal number of TPM command for which the HMAC must be calculated.

ContinueUse
The continue use flag for the authorization session. Required to calculate or verify the rgbHmacData

ulSizeNonces
The size of the nonces rgbNonceEven, rgbNonceOdd, rgbNonceEvenOSAP and rgbNonceOddOSAP

rgbNonceEven
Even nonce previously generated by TPM to cover inputs. Required to calculate or verify the rgbHmacData

rgbNonceOdd
Nonce generated by TSP associated with the authorization session. Required to calculate or verify the rgbHmacData

rgbNonceEvenOSAP
Nonce generated by TPM and associated with shared secret. Required to calculate the shared secret for the OSAP session.

rgbNonceOddOSAP
The nonce generated by the caller associated with the shared secret. Required to calculate the shared secret for the OSAP session.

ulSizeDigestHmac
The size of the parameter rgbParamDigest and rgbHmacData

rgbParamDigest
SHA1 digest of the TPM function parameters.

If ReturnOrVerify = TRUE, digest of incoming parameters.

If ReturnOrVerify = FALSE, digest of ingoing parameters.

rgbHmacData
The authorization digest for inputs or returned parameters.

If ReturnOrVerify = TRUE, authorization digest required to process the TPM command

If ReturnOrVerify = FALSE, authorization digest returned from the TPM

Example:

```
TSS_RESULT Tspicb_CallbackHMACAuth
(
```


Copyright TCG 2003

```
    PVOID       lpAppData            // in
    TSS_HOBJECT hAuthorizedObject    // in
    TSS_BOOL    ReturnOrVerify       // in
    UINT32      ulPendingFunction    // in
    TSS_BOOL    ContinueUse          // in
    UINT32      ulSizeNonces         // in
    BYTE*       rgbNonceEven         // in
    BYTE*       rgbNonceOdd,         // in
    BYTE*       rgbNonceEvenOSAP     // in
    BYTE*       rgbNonceOddOSAP      // in
    UINT32      ulSizeDigestHmac     // in
    BYTE*       rgbParamDigest       // in
    BYTE*       rgbHmacData          // in, out

{
    // Get secret from user for hAuthorizedObject
    // (e.g. via application dialog)

    BYTE *pbSecret = SHA1(…)
    BYTE *pHmacDataTemp= HMAC(pbSecret, DataToHMAC);

    if (ReturnOrVerify)
    {
          memcpy(rgbHmacData,pHmacDataTemp, ulSizeDigestHmac);
    }
    else
    {
          if (memcmp(rgbHmacData, pHmacDataTemp, ulSizeDigestHmac))
                return TSS_E_FAIL;
    }

    return TSS_SUCCESS;
}
```

3.3.4.2 Tspicb_CallbackXorEnc

Start of informative comment:

This method is called when one of the following TSPI function is called and the callback mechanism is set in the assigned policy object. A flag indicates the purpose of the call. That means, in one case a new secret must be insert and in the other case an existing secret must be changed.

Some functions require OSAP sessions and for others it is optional to use OIAP or OSAP. For the functions that require OSAP, there is potential confusion as to which policy object should hold the callback pointer when desired. This section defines which policy object will hold the callback pointer when desired.

TSPI functions:

Tspi_Data_Seal

The callback is registered to the usage policy of the enc data object.

Tspi_Key_CreateKey (NEW; Usage and migration secret)

The callback is registered to the usage policy of the parent key object

Tspi_TPM_CollateIdentityRequest

The policy object is registered to the usage policy of the TPM object.

Tspi_ChangeAuth

The callback is registered to the usage policy of the object being changed.

Refer to TCPA 1.1b Main Specification for information about the encryption process

End of informative comment.

Definition:

```
TSS_RESULT Tspicb_CallbackXorEnc
(
   PVOID       lpAppData,           // in
   TSS_HOBJECT hOSAPObject,         // in
   TSS_HOBJECT hObject,             // in
   TSS_FLAG    PurposeSecret,       // in
   UINT32      ulSizeNonces,        // in
   BYTE*       rgbNonceEven,        // in
   BYTE*       rgbNonceOdd,         // in
   BYTE*       rgbNonceEvenOSAP,    // in
   BYTE*       rgbNonceOddOSAP,     // in
   UINT32      ulSizeEncAuth,       // in
   BYTE*       rgbEncAuthUsage,     // out
   BYTE*       rgbEncAuthMigration  // out
):
```

Parameters

lpAppData
Pointer to application provided data as provided on registration of callback function

hOSAPObject
Handle to the object authorization is required

hObject
Handle to the object the secret should be set or changed

PurposeSecret
Flag indicating the whether a new secret must be inserted or an existing secret must be changed.

(NewSecret)

TRUE: New secret must be inserted

FALSE: Existing secret must be changed

ulSizeNonces
The size of the nonces rgbNonceEven, rgbNonceOdd, rgbNonceEvenOSAP and rgbNonceOddOSAP

rgbNonceEven
Even nonce previously generated by TPM to cover inputs. Required to calculate or verify the rgbHmacData

rgbNonceOdd
Nonce generated by TSP associated with the authorization session. Required to calculate or verify the rgbHmacData

rgbNonceEvenOSAP
Nonce generated by TPM and associated with shared secret. Required to calculate the shared secret for the OSAP session.

rgbNonceOddOSAP
The nonce generated by the caller associated with the shared secret. Required to calculate the shared secret for the OSAP session.

ulSizeEncAuth
The size of the parameter rgbEncAuthUsage and rgbEncAuthMigration

rgbEncAuthUsage
Encrypted usage secret

rgbEncAuthMigration
Encrypted migration secret

Example:

```
TSS_RESULT Tspicb_CallbackXorEnc(
   PVOID       lpAppData,           // in
   TSS_HOBJECT hOSAPObject,         // in
   TSS_HOBJECT hObject,             // in
   TSS_FLAGS   PurposeSecret,       // in
   UINT32      ulSizeNonces,        // in
   BYTE*       rgbNonceEven,        // in
   BYTE*       rgbNonceOdd,         // in
   BYTE*       rgbNonceEvenOSAP,    // in
   BYTE*       rgbNonceOddOSAP,     // in
   UINT32      ulSizeEncAuth,       // in
   BYTE*       rgbEncAuthUsage,     // out
   BYTE*       rgbEncAuthMigration  // out

{
   // Get secret from user for hOSAPObject (e.g. via application dialog)
   BYTE *pbSecretFromUser= SHA1(…)

   BYTE *pbSessionSecret= HMAC(pbSecretFromUser,
               rgbNonceEvenOSAP,
```

```
                rgbNonceOddOSAP);

    BYTE *pbEncValue= SHA1(pbSessionSecret, rgbNonceEven);

    BYTE *pbEncAuthUsageTemp= XOR(pbSecretToEnc, pbEncValue);
    memcpy(rgbEncAuthUsage, pbEncAuthUsageTemp, ulSizeEncAuth);

    return TSS_SUCCESS;
}
```

3.3.4.3 Tspicb_CallbackTakeOwnership

Start of informative comment:

This method is called when the function Tspip_TPM_TakeOwnership is used and the callback mechanism is set in the assigned policy object of the object.

Both the SRK and TPM secrets are being set in Tspip_TPM_TakeOwnership. Both the SRK and TPM polices need appropriate information to obtain the encrypted secrets before the command is executed. If the SRK authorization is to be encrypted by the calling application, then the policy object of the SRK usage policy will have the callback registered to it. If the TPM authorization must also be encrypted by the calling application, then the usage policy of the TPM object will have the registered callback.

End of informative comment.

It is the application writer's responsibility to display or mask any pop-up windows that may result from this function.

Definition:

```
TSS_RESULT Tspicb_CallbackTakeOwnership
(
   PVOID       lpAppData         // in
   TSS_HOBJECT hObject,          // in
   TSS_HKEY    hObjectPubKey     // in
   UINT32      ulSizeEncAuth,    // in
   BYTE*       rgbEncAuth        // out
);
```

Parameters

lpAppData
Pointer to application provided data as provided on registration of callback function

hObject
Handle to the TPM object

hObjectPubKey
Handle to the key object representing the endorsement public key required for encrypting the secret of SRK and the TPM owner secret.

ulSizeEncAuth
The size of the parameter *rgbEncAuthOwner* and rgbEncAuthSrk

rgbEncAuth
The encrypted authorization. If the callback is registered to the usage policy of the TPM object, then this will be the encrypted owner authorization. If it is registered to the usage policy of the SRK object, then this will be the encrypted SRK usage policy.

Example:

```
TSS_RESULT Tspicb_CallbackTakeOwnership(
   PVOID       lpAppData         // in
   TSS_HOBJECT hObject,          // in
   TSS_HKEY    hObjectPubKey     // in
   UINT32      ulSizeEncAuth,    // in
   BYTE*       rgbEncAuth        // out
{
   // Get secrets (e.g. via application dialog)
```


Copyright TCG 2003

```
    BYTE *pbSecretOwnerToEnc = SHA1(…)
    BYTE *pbSecretSrkToEnc = SHA1(…)

    Tspi_GetAttribData(hObjectPubKey, …, pubKey);

    BYTE *pbEncAuthTemp = Encrypt(pubKey, pbSecretToEnc);

    memcpy(rgbEncAuth, pbEncAuthTemp, ulSizeEncAuth);

    return TSS_SUCCESS;
}
```

3.3.4.4 Tspicb_CallbackChangeAuthAsym

Start of informative comment:

This method is called when the function Tspip_ChangeAuthAsym is used and the callback mechanism is set in the assigned policy object of the object.

The service provider uses the HMAC calculation as parameter for the TPM command TPM_ChangeAuthAsymFinish(…). The HMAC links the old and new authorization values together (see Main Specification).

The usage policy of the object being changed will have this registered callback.

End of informative comment.

Definition:

```
TSS_RESULT CallbackChangeAuthAsym
(
   PVOID       lpAppData         // in
   TSS_HOBJECT hObject,          // in
   TSS_HKEY    hObjectPubKey     // in
   UINT32      ulSizeEncAuth,    // in
   UINT32      ulSizeAuthLink,   // in
   BYTE*       rgbEncAuth,       // out
   BYTE*       rgbAuthLink       // out
);
```

Parameters

lpAppData
Pointer to application provided data as provided on registration of callback function

hObject
Handle to the object the secret should changed

hObjectPubKey
Handle to the key object representing the public key required for encrypting the secret of SRK and the TPM owner secret.

ulSizeEncAuth
The size of the parameter rgbEncAuth

ulSizeAuthLink
The size of the parameter rgbAuthLink

rgbEncAuth
New authorization data encrypted with ephemeral key

rgbAuthLink
HMAC digest that links the old and new authorization values together

Example:

```
TSS_RESULT Tspicb_CallbackChangeAuthAsym
(
```

```
    PVOID       lpAppData         // in
    TSS_HOBJECT hObject,          // in
    TSS_HKEY    hObjectPubKey     // in
    UINT32      ulSizeEncAuth,    // in
    UINT32      ulSizeAuthLink,   // in
    BYTE*       rgbEncAuth,       // out
    BYTE*       rgbAuthLink       // out
{
    // Get old and new secret (e.g. via application dialog)
    BYTE *pbOldSecret = SHA1(…)
    BYTE *pbNewSecret = SHA1(…)

    Tspi_GetAttribData(hObjectPubKey, …, pubKey);

    BYTE* pbEncAuthTemp = Encrypt(pubKey, pbNewSecret);
    memcpy(rgbEncAuth, pbEncAuthTemp, ulSizeEncAuth);

    BYTE* pbAuthLinkTemp = HMAC(pbOldSecret, pbNewSecret);
    memcpy(rgbAuthLink, pbAuthLinkTemp, ulSizeAuthLink);

    return TSS_SUCCESS;
    }
```

3.3.4.5 Tspicb_CollateIdentity

Start of informative comment:

This method is called when the function Tspi_CollateIdentity is used. The intent is to allow application writers to use any symmetric key algorithm they choose when encrypting the identity request packet.

This callback is registered as an attribute of the TPM object within the calling context.

End of informative comment.

It is the application writer's responsibility to display or mask any pop-up windows that may result from this function. This callback MAY be provided by default by the TSP. A call to

Definition:

```
TSS_RESULT Tspicb_CollateIdentity
(
   PVOID              lpAppData,                     // in
   UINT32             ulTCPAPlainIdentityProofLength, // in
   BYTE*              rgbTCPAPlainIdentityProof,     // in
   TSS_ALGORITHM_ID   algID,                         // in
   UINT32             ulSessionKeyLength,            // out
   BYTE*              rgbSessionKey,                 // out
   UINT32*            pulTCPAIdentityProofLength,       // out
   BYTE*              rgbTCPAIdentityProof              // out
);
```

Parameters

lpAppData
Pointer to application provided data as provided on registration of callback function

ulTCPAPlainIdentityProofLength
Size of the Identity Proof in plain text.

rgbTCPAPlainIdentityProof
Pointer containing the plain text identity request structure `TCPA_IDENTITY_PROOF`.

algID
The encryption algorithm to use to encrypt the identity proof.

ulSessionKeyLength
Length of the symmetric key

rgbSessionKey
Pointer containing the session key EncTcpaSymmetricKey as documented in section 9.4.1 of the TCPA 1.1b Main Specification of 'ulSessionKeyLength' bytes

pulTCPAIdentityProofLength
Size of the encrypted identity proof.

rgbTCPAIdentityProof
Pointer containing the encrypted identity proof.

Copyright TCG 2003

3.3.4.6 Tspicb_ActivateIdentity

Start of informative comment:

This method is called when the function Tspi_ActivateIdentity is used. The intent is to allow application writers to use any symmetric key algorithm they choose when decrypting the credential received from the privacy CA.

This callback is registered as an attribute of the TPM object within the calling context.

End of informative comment.

It is the application writer's responsibility to display or mask any pop-up windows that may result from this function.

Definition:

```
TSS_RESULT Tspicb_ActivateIdentity
(
   PVOID    lpAppData                      // in
   UINT32   ulSessionKeyLength,            // in
   BYTE     *rgbSessionKey,                // in
   UINT32   ulSymCAAttestationBlobLength   // in
   BYTE     *rgbSymCAAttestationBlob,      // in
   UINT32   *pulCredentialLength,          // out
   BYTE     *rgbCredential                 // out
);
```

Parameters

lpAppData
Pointer to application provided data as provided on registration of callback function.

ulSessionKeyLength
Length of the symmetric key

rgbSessionKey
pointer containing the session key of 'ulSessionKeyLength' bytes

ulSymCAAttestationBlobLength
Size of the encrypted credential received from the enhanced CA.

rgbSymCAAttestationBlob
Pointer containing the encrypted credential from the enhanced CA.

pulCredentialLength
Size of the decrypted Credential

rgbCredential
Pointer containing the Credential received from the enhanced CA.

4. TCG Core Services (TCS)

4.1 TCS Architecture and Interface Description

4.1.1 TCS Memory Manager

Start of informative comment:

The TCS interfaces use the standard cross-process memory management. The memory allocation model is supported by using the RPC data marshalling available on the platform. The memory buffers are arranged and allocate depending on the IDL parameter type:

[in] parameters – Caller-allocated memory management. The data is copied from the Caller-to-Callee. The Caller must allocate the memory before the function call and de-allocate the memory when finished with the data (typically immediately after function return).

[out] parameters - Callee-allocated memory management. The data is copied from the Callee-to-Caller. The Callee must allocate the memory before returning from the function call and the Caller must de-allocate the memory when it is finished referencing the data (typically sometime after function return). For memory pointers, a pointer-to-pointer declaration is used.

[in,out] parameters – The data is copied in both directions, Caller-to-Callee and Callee-to-Caller. The Caller and Callee are responsible for allocating and de-allocating any respective allocated memory as stated in the [in] and [out] convention above.

End of informative comment.

4.1.2 TCS Data Marshalling

Start of informative comment:

Data marshalling is typically performed using the RPC mechanism available on the specific platform. This mechanism is used to perform the inter-process communication between a TCG-enabled application and TCS platform service. Data is transferred across the process boundaries via the data marshalling proxy and stub routines. Most platforms provide support for default proxy and stub generation, but also allow for custom data marshalling code to be designed. The specifics of each platform data marshalling technique and options are outside the scope of this document.

End of informative comment.

4.1.3 TCS Interface Dynamics

Start of informative comment:

The TPM parameter block generator functions are incorporated as part of a TCS platform system service, which is typically implemented as an out-of-process server. The TCS interface (Tcsi) and TPM parameter block generator API do not define callback function entry points and interface dynamics are left up to the implementation. Either synchronous or asynchronous dynamics can be implemented depending on the type of underlying RPC support available on the platform

End of informative comment.

Copyright TCG 2003

4.2 TCS-specific Return Code Defines

With the following table the error codes common to all Tcsi functions are listed. In addition to these error codes, the TSS_E_* error code may also be returned with the layer set to the value for the TCS.

In addition each Tcsi function will list in its description the error return codes specific to the function.

Type	Definition
TCS_SUCCESS	Successful completion
TCS_E_FAIL	General failure.
TCS_E_KEY_MISMATCH	Key addressed by the application key handle does not match the key addressed by the given UUID.
TCS_E_KM_LOADFAILED	Key adressed by Key's UUID cannot be loaded because one of the required parent keys needs authorization.
TCS_E_KEY_CONTEXT_RELOAD	The Key Cache Manager could not reload the key into the TPM.
TCS_E_INVALID_CONTEXTHANDLE	The context handle supplied is invalid.
TCS_E_INVALID_KEYHANDLE	The key handle supplied is invalid.
TCS_E_INVALID_AUTHHANDLE	The authorization session handle supplied is invalid.
TCS_E_INVALID_AUTHSESSION	The auth session has been closed by the TPM itself. E.g. due to a command failure like authorization failure
TCS_E_INVALID_KEY	The key has been unloaded by the TPM itself. E.g. due to a OwnerCLear command

4.3 TSPI-specific Return code Rules

Functions or methods within this layer MAY return common errors defined in section: Common Return Code Defines 2.4.2 All the any of above return codes may returned by any of the functions in this section.

4.4 Structures and Definitions

This document utilizes structures in the function definitions that are defined in the TCPA 1.1b Main Specification and in the TSP section of this specification. In addition, the following structures are defined as follows:

4.4.1 Data Types of the Tcsi

Start of informative comment:

This section describes data type declarations especially required at the Tcsi.

For all parameters providing a buffer length a size of 32 bit should be sufficient; the same applies for the flags parameter indicating the attribute type of an object.

Handles are used as unsigned integer values to address any instantiated object.

End of informative comment.

Type	Definition	Usage
TCS_AUTHHANDLE	UINT32	Handle addressing a authorization session
TCS_CONTEXT_HANDLE	UINT32	Basic context handle
TCS_KEY_HANDLE	UINT32	Basic key handle

4.4.2 TCS_LOADKEY_INFO

Start of informative comment:

TCS_LOADKEY_INFO provides information to enable the TSS CS Key Manager Service to load a registered key if a required parent key needs authorization.

End of informative comment.

Definition:

```
typedef struct tdTCS_LOADKEY_INFO
{
   TSS_UUID             keyUUID;
   TSS_UUID             parentKeyUUID;
   TCPA_DIGEST          paramDigest;  // SHA1 digest of the TPM_LoadKey
         // Command input parameters
         // As defined in TCPA 1.1b Main Specification
   TPM_AUTH             authData;     // Data regarding a valid auth
                                      // Session including the
                                      // HMAC digest
} TCS_LOADKEY_INFO;
```

4.5 TCS Context Manager

4.5.1 TCS Context Manager Functions and Operations

All resources a calling application can work with are assigned to a certain context.

If the TCS has to allocate memory, which has to be provided to the calling application (important for variable sized output data blocks), this kind of resources must also be assigned to a certain context.

Resource Relationship:

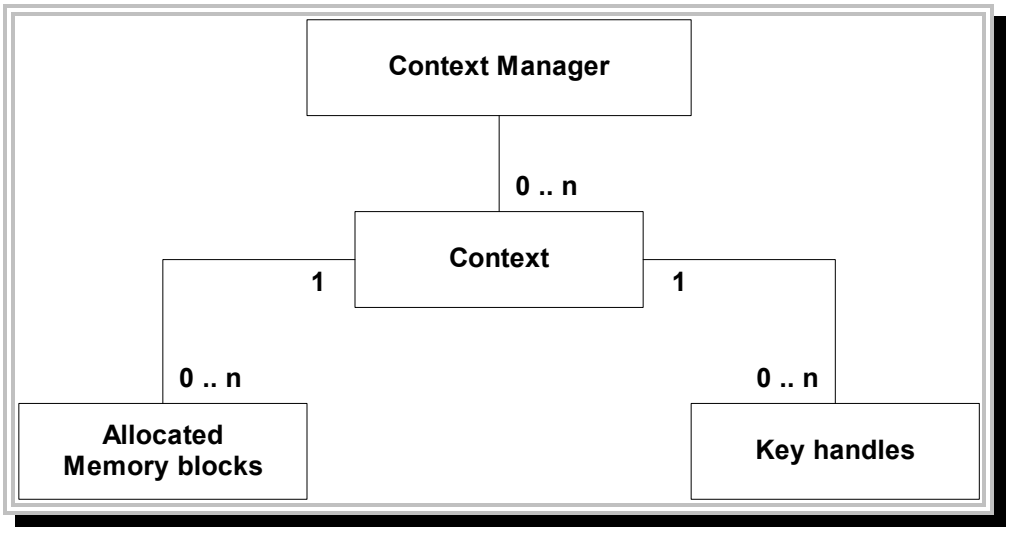


Figure 4-1 TCS Context Manager and Operations

4.5.2 TCS Context Manager Interface

4.5.2.1 Tcsi_OpenContext

Start of informative comment:

Tcsi_OpenContext is used to obtain a handle to a new context.

The context handle is used in various functions to assign resources to it. An application (i.e., TSP or application directly utilizing the TCS) may require more than one context open.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_OpenContext
(
   TCS_CONTEXT_HANDLE*  hContext // out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_OpenContext")]
TSS_RESULT Tcsi_OpenContext
(
   [out] TCS_CONTEXT_HANDLE*  hContext
);
```

Type	Label	Description
TCS_CONTEXT_HANDLE*	hContext	Return a handle to an established TSS CS context. This handle can now be supplied to other functions attempting to do work within this context.

4.5.2.2 Tcsi_CloseContext

Start of informative comment:

Tcsi_CloseContext releases all resources assigned to the given context and the context itself.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_CloseContext
(
   TCS_CONTEXT_HANDLE   hContext // in
);
```

IDL-Definition:

```
[helpstring("method Tcsi_CloseContext")]
TSS_RESULT Tcsi_CloseContext
(
   [in] TCS_CONTEXT_HANDLE hContext
);
```

Type	Label	Description
TCS_CONTEXT_HANDLE	Hcontext	Context handle to be released.

4.5.2.3 Tcsi_FreeMemory

Start of informative comment:

Tcsi_FreeMemory frees memory allocated by TSS CS on a context base. If pMemory equals NULL all allocated memory blocks will be freed.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_FreeMemory
(
   TCS_CONTEXT_HANDLE   hContext,    // in
   BYTE*                pMemory      // in
);
```

IDL-Definition:

```
[helpstring("method Tcsi_FreeMemory")]
TSS_RESULT Tcsi_FreeMemory
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] BYTE*              pMemory
}
```

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
BYTE*	pMemory	Pointer addressing memory to be freed.

Copyright TCG 2003

4.5.2.4 Tcsi_GetCapability

Start of informative comment:

Tcsi_GetCapability provides the capabilities of the TCS.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_GetCapability
(
   TCS_CONTEXT_HANDLE      hContext,    // in
   TCPA_CAPABILITY_AREA    capArea,     // in
   UINT32                  subCapSize,  // in
   BYTE*                   subCap,      // in
   UINT32*                 respSize,    // out
   BYTE**                  resp         // out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_GetCapability")]
TSS_RESULT Tcsi_GetCapability
(
   [in] TCS_CONTEXT_HANDLE           hContext,
   [in] TCPA_CAPBILITY_AREA            capArea,
   [in] UINT32                       subCapSize,
   [in, size_is(subCapSize)] BYTE*   subCap,
   [out] UINT32*                     respSize,
   [out, size_is(*respSize)] BYTE**  resp
);
```

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_CAPABILITY_AREA	capArea	Partition of capabilities to be interrogated
UINT32	subCapSize	Size of subCap parameter
BYTE*	subCap	Further definition of information
UINT32*	respSize	The length of the returned capability response
BYTE**	resp	The capability response

Defined Attributes

Capability Area	SubCap Area	Response
TSS_TCSCAP_ALG		Queries whether an algorithm is supported.
TSS_TCSCAP_VERSION		Queries the current TCS version.
TSS_TCSCAP_MANUFACTURER	TSS_TCSCAP_PROP_MANUFACTURER_ID	Returns the manufacurer or implementer of the TCS. Return SHALL be a UINT32 using the same identity system used in the main specification for: Cap: TCPA_CAP_PROPERTY

Copyright TCG 2003

		Subcap: TCPA_CAP_PROP_MANUFACTURER
TSS_TCSCAP_MANUFACTURER	TSS_TCSCAP_PROP_MANUFACTURER_STR	Returns an Unicode string of the TCS manufacturer. The contents of this string is determined by the manufacturer and is subject to change in subsequent releases of the TCS.
TSS_TCSCAP_CACHING		Queries the support of key and authorization caching.
TSS_TCSCAP_CACHING	TSS_TCSCAP_PROP_KEYCACHE	TSS_BOOL value. Indicates support of key caching
TSS_TCSCAP_CACHING	TSS_TCSCAP_PROP_AUTHCACHE	TSS_BOOL value. Indicates support of authorization session caching
TSS_TCSCAP_PERSSTORAGE		Queries the support of a persistent storage

Comment:

This command differs from the Tcsi_GetCapability (note the P missing from this command) in that it retrieves the capabilities of the Core Services not the TPM. This command is not directly sent to the TDDL but may invoke or make inquires to it in order to service this request.

4.6 TCS Key and Credential Manager

4.6.1 TCS Key & Credential Manager Functions and Operations

4.6.1.1 TCS Key Manager

Start of informative comment:

The TCS Key Manager Services allow definition of a persistent key hierarchy. The persistent key hierarchy consists of storage keys that make up the base storage key structure that will exist before any user may attempt to load a key. Additionally the persistent key hierarchy may contain system specific leaf keys as for instance identity keys.

The TCS Key Manager Services interface was designed to allow a flexible key structure such that an instance like an IT department of an enterprise may define a deep key hierarchy, a shallow hierarchy, roaming keys, migration base keys, etc.

All keys, which should be internally managed by the Key Manager Services of TCS must be registered in the persistent storage database of TCS. Each key registered in that database will be referenced by its UUID and called a persistent key from this specification's point of view.

Some registered keys have a defined fixed UUID by which they can be referenced on all systems providing the same registered key hierarchy. These UUIDs do not provide any information to identify the system the key is registered on.

An application can also load keys not registered in the TCS database. These keys are loaded utilizing the Tcsi by providing a key blob as defined by TCPA_KEY. These keys are called temporary keys from this specification's point of view.

After a key was loaded either by using a UUID or by using a key blob, this key will be addressed on further calls utilizing the key by the application key handle, which was returned from TCS on a load key command.

Using the Key Manager Services provided by TCS will simplify the whole mechanism of loading a key into the TPM from a calling context's point of view. The application must only address a key to be loaded by its well known UUID and the Key Manager Services will do all the required loading of the underlying parent keys depending on the registered key hierarchy, which may be totally hidden from the application's scope.

The key hierarchy can be defined by some instance like for example the IT department of an enterprise and the TCG-aware applications may not need to know this key hierarchy at all.

Keys once registered in PS will stay registered in PS until they are unregistered. The PS will stay valid across boots.

Application key handles got from a load key command are valid as long as the TCS is not restarted or the key is not evicted from the Key Cache Manager Service. Application key handles will not stay valid across boots.

End of informative comment.

Keys once registered in PS will stay registered in PS until they are unregistered. The PS will stay valid across boots.

4.6.1.2 TCS Key Cache Manager

Start of informative comment:

The TCS Key Cache Manager Service (KCM) allows caching keys to manage the restricted resources of a TPM. The KCM is responsible to manage the restricted resources of the TPM and to hide these restrictions from the calling applications. An application can load a key in to the TPM by utilizing the KCM functionality and can assume that this key is available for further use. The KCM is

responsible to ensure that a key, which has already been loaded by an application, is available in the TPM, when the application requires that key for a certain command. If all TPM resources are in use, the KCM has to free resources in order to load a key or to get the required key back in to the TPM.

An application must load a key into the TPM utilizing the KCM. The KCM returns an application key handle to the caller and manages a mapping mechanism between the returned application key handle and the actual TPM key handle. The actual TPM key handle will change whenever a key has to be unloaded from the TPM by the KCM in order to free resources since another key has to be loaded and the KCM reloads the key into the TPM again. The application key handle returned to the calling application remains constant as long as the key is not reloaded by the application itself.

End of informative comment.

The key cache mechanism can be implemented by using the TPM commands:

- LoadKey and EvictKey or
- LoadKeyContext and SaveKeyContext

If TPM_LoadKey / TPM_EvictKey has to be used only, because the TPM does not provide TPM_LoadKeyContext / SaveKeyContext, the Key Cache Manager can transparently reload keys only if the required parent key(s) needs no authorization. There must not be any caching of secrets required for authorization.

If TPM_LoadKeyContext / TPM_SaveKeyContext is provided by the TPM, the reloading of keys can be done totally transparent for the calling applications no matter if the required parent keys needs authorization or not.

An application key handle returned to a caller must remain unchanged as long as the caller evicts the key by itself.

The internally managed TPM key handles returned by the appropriate TPM commands to reload keys may change all the time.

The Key Cache Management Service will map the returned stable application key handles with the unstable internal TPM key handles.

A caller must address a loaded key only by the application key handle got from the Key Cache Manager.

Each returned application key handle must be bound to a certain context.

On initial loading a key for the first time the KCM utilizes the TPM command TPM_LoadKey in order to load the given key blob into the TPM. Depending on available authorization data, the TPM command is built with or without authorization data.

The returned application key handles are valid:

- As long as a caller does not evict the key providing the application key handle.
- As long as the context the application key handle is bound to is not closed.
- As long as the Key Cache Manager is not stopped.

If a key can not be reloaded, since authorization is required or since some other failure, the Tcsi command addressing that key will return an error: TCS_E_KEY_CONTEXT_RELOAD.

 Copyright TCG 2003

4.6.1.3 TCS Credential Management

Start of informative comment:

The TCS manages the endorsement credential, the platform credential and the conformance credential. The registration and management of these credentials is TCS vendor specific.

It is not possible to register any credentials or certificates by an application utilizing the Tcsi. This would simplify the credential management for applications. But retrieving credentials without any access control provided by TCS may cause privacy concerns.

The TCS provides the endorsement credential, the platform credential and the conformance credential for a calling application only if the TCS could proof an owner authorization. This TCS Credential Service is provided for making TPM identities only and to fulfill any privacy concerns.

Example of Tcsip_MakeIdentity(…):

The TSS CS first creates a new identity key by processing the TPM command TPM_MakeIdentity().

The TPM command to create an identity key is owner authorized. If this command was successfully processed an owner could be proofed and the TCS will additionally return the endorsement credential, the platform credential and the conformance credential to the caller. This should be an atomic function to start the process of making a new TPM identity.

End of informative comment.

Registration and management of the endorsement credential, the platform credential and the conformance credential is TCS vendor specific. No other credential or certificate can be registered in the TCS Credential Management Services by utilizing the Tcsi. The endorsement credential, platform credential and conformance credential is provided by TCS Credential Manager Services utilizing Tcsip_MakeIdentity only if an owner authorization can be proven.

Copyright TCG 2003

4.6.2 TCS Key and Credential Manager Interface

4.6.2.1 Interfaces

4.6.2.2 Key Registration

4.6.2.2.1 Tcsi_RegisterKey

Start of informative comment:

Tcsi_RegisterKey allows registering a key in the TCS Persistent Storage (PS). Only system specific keys (keys definitely bound to a certain system) should be registered in TCS PS.

A key can be registered in TCS PS by providing:

- A UUID for that key,
- A UUID for its wrapping parent key and
- The key blob itself.

If the same UUID is used to register a key on different systems this key can be addressed on different systems by the same UUID. This may be done for a basic roaming key, which will wrap all user storage keys in the appropriate key hierarchy.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_RegisterKey
(
   TCS_CONTEXT_HANDLE   hContext,      // in
   TSS_UUID             WrappingKeyUUID,  // in
   TSS_UUID             KeyUUID,       // in
   UINT32               cKeySize,      // in
   BYTE*                rgbKey,        // in
   UINT32               cVendorData,   // in
   BYTE*                gbVendorData   // in
);
```

IDL-Definition:

```
[helpstring("method Tcsi_RegisterKey")]
TSS_RESULT Tcsi_RegisterKey
(
   [in] TCS_CONTEXT_HANDLE       hContext,
   [in] TSS_UUID                 WrappingKeyUUID,
   [in] TSS_UUID                 KeyUUID,
   [in] UINT32                   cKeySize,
   [in, size_is(cKeySize)] BYTE* rgbKey,
   [in, defaultvalue(0)] UINT32  cVendorData,
   [in, size_is(cVendorData), defaultvalue(NULL)] BYTE* rgbVendorData
);
```


Copyright TCG 2003

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TSS_UUID	WrappingKeyUUID	UUID of the already registered wrapping parent key.
UINT32	cKeySize	Size of the provided keyblob in bytes.
BYTE*	rgbKey	Byte stream containing the key blob of the key to be registered.
UINT32	cVendorData	Size of vendor specific data blob in bytes; may be 0.
BYTE*	rgbVendorData	Vendor specific data blob; may be NULL.

Comment:

If a key has already been registered under the provided keyUUID, the function will fail with the error TCS_E_KEY_ALREADY_REGISTERED. The application may then unregister the key, which is already assigned to that UUID. Now the new key can be registered.

The key is stored in the persistent storage without any information about the context used for doing the registration.

Return Value:

TCS_SUCCESS
TCS_E_KEY_ALREADY_REGISTERED
TCS_E_KEY_NOT_REGISTERED
TCS_E_FAIL

4.6.2.2.2 Tcsip_UnregisterKey

Start of informative comment:

A key once registered in the TCS PS can be unregistered from the PS, if that key is not required any longer.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_UnregisterKey
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TSS_UUID             KeyUUID     // in
);
```

IDL-Definition:

```
[helpstring("method Tcsip_UnregisterKey")]
TSS_RESULT Tcsip_UnregisterKey
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TSS_UUID           KeyUUID,
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	HContext	Handle to established context.
TSS_UUID	KeyUUID	UUID by which the key is registered.

Comment:

If a key has not been registered under the provided KeyUUID, the function will fail with the error TCS_E_KEY_NOT_REGISTERED.

Return Value:

TCS_SUCCESS
TCS_E_KEY_NOT_REGISTERED
TCS_E_KEY_MISMATCH
TCS_E_INVALID_CONTEXTHANDLE
TCS_E_FAIL

4.6.2.3 TCS Get Key Hierarchy Information

Start of informative comment:

The Key Management Services of TCS provide information about the registered key hierarchy.

The Key Management Services will provide an array of information or only one entry of that array based on a certain key.

The returned information contains data like the following:

- The UUID of the key,
- The UUID of the wrapping parent key,
- Authorization required and
- Key already loaded.

The application can use this information to improve the strategy of how a key may be loaded into the TPM, if for instance one or more storage keys require authorization.

End of informative comment.

4.6.2.3.1 Tcsi_EnumRegisteredKeys

Start of informative comment:

Tcsi_EnumRegisteredKeys allows obtaining an array of TSS_KM_KEYINFO structures. This information reflects the registered key hierarchy. The caller will receive information of the whole key hierarchy. The keys stored in the persistent storage are totally independent from either the context provided in the function call or the context, which was provided while processing the key registration.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_EnumRegisteredKeys
(
   TCS_CONTEXT_HANDLE   hContext,         // in
   TSS_UUID*            pKeyUUID          // in
   UINT32*              pcKeyHierarchySize,  // out
   TSS_KM_KEYINFO**     ppKeyHierarchy       // out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_EnumRegisteredKeys")]
TSS_RESULT Tcsi_EnumRegisteredKeys
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TSS_UUID*          pKeyUUID,
   [out] UINT32*           pcKeyHierarchySize,
   [out, size_is(*pcKeyHierarchySize)] TSS_KM_KEYINFO** ppKeyHierarchy
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TSS_UUID*	pKeyUUID	UUID of key the key hierarchy

		should be returned of. If NULL, the whole key hierarchy will be returned.
UINT32*	pcKeyHierarchySize	Return number of array entries.
TSS_KM_KEYINFO**	ppKeyHierarchy	Return pointer to memory containing array of structures reflecting the registered key hierarchy.

Comment:

The TCS allocates memory, assigns the memory resource to the given context and returns an array of TSS_KM_KEYINFO structures.

This array will return information of the whole registered key hierarchy independent from any context.

If a certain key UUID is provided, the returned array of TSS_KM_KEYINFO structures only contains data reflecting the path of the key hierarchy regarding that key. The first array entry is the key addressed by the given UUID followed by its parent key up to the root key.

If no key UUID is provided (pKeyUUID == NULL), the returned array TSS_KM_KEYINFO structures contains data reflecting the whole key hierarchy starting with root key.

If no keys have been registered *pcKeyHierarchySize = 0 and *ppkeyHierarchy = NULL and the function returns with TCS_SUCCESS.

Return Value:

TCS_SUCCESS
TCS_E_FAIL

4.6.2.3.2 Tcsi_GetRegisteredKey

Start of informative comment:

Tcsi_GetRegisteredKey allows obtaining a TSS_KM_KEYINFO structure containing information about the registered key.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_GetRegisteredKey
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TSS_UUID             KeyUUID,    // in
   TSS_KM_KEYINFO**     ppKeyInfo   // out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_GetRegisteredKey")]
TSS_RESULT Tcsi_GetRegisteredKey
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TSS_UUID KeyUUID,
   [out] TSS_KM_KEYINFO**  ppKeyInfo
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	Hcontext	Handle to established context.
TSS_UUID	KeyUUID	UUID of the key information is required.
TSS_KM_KEYINFO**	PpKeyInfo	Return pointer to memory containing information about the key of interest.

Comment:

The TCS allocates memory, assigns the memory resource to the given context and returns a TSS_KM_KEYINFO structure.

If a key has not been registered under the provided KeyUUID, the function fails with the error TCS_E_KEY_NOT_REGISTERED and returns a NULL pointer.

Return Value:

TCS_SUCCESS
TCS_E_KEY_NOT_REGISTERED
TCS_E_FAIL

Copyright TCG 2003

4.6.2.3.3 Tcsi_GetRegisteredKeyBlob

Start of informative comment:

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_GetRegisteredKeyBlob
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TSS_UUID             KeyUUID,    // in
   UINT32*              pcKeySize,  // out
   BYTE**               prgbKey  // out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_GetRegisteredKeyBlob")]
TSS_RESULT Tcsi_GetRegisteredKeyBlob
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TSS_UUID           KeyUUID,
   [out] UINT32*           pcKeySize,
   [out, size_is(, *pcKeySize)] BYTE** prgbKey
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TSS_UUID	KeyUUID	UUID of the key information is required.
UINT32*	pcKeySize	Size of the returned keyblob in bytes.
BYTE**	prgbKey	Returned pointer to a byte stream containing the key blob of interest.

Comment:

The TCS allocates memory, assigns the memory resource to the given context and returns a registered key blob as defined by TCPA_KEY structure.

If a key has not been registered under the provided KeyUUID, the function fails with the error TCS_E_KEY_NOT_REGISTERED and returns a NULL pointer and *pcKeySize = 0.

Return Value:

TCS_SUCCESS
TCS_E_KEY_NOT_REGISTERED
TCS_E_FAIL

4.6.2.3.4 Tcsip_GetRegisteredKeyByPublicInfo

Start of informative comment:

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_GetRegisteredKeyByPublicInfo
(
   TCS_CONTEXT_HANDLE   hContext,
   TSS_ALGORITHM_ID     algID,               // in
   UINT32               ulPublicInfoLength,  // in
   BYTE*                rgbPublicInfo,       // in
   UINT32*              keySize,             //out
   BYTE**               keyBlob              //out
)
```

IDL-Definition:

```
[helpstring("method Tcsip_GetRegisteredKeyByPublicInfo ")]
TSS_RESULT Tcsip_GetRegisteredKeyByPublicInfo
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TSS_ALGORITHM_ID   algID,
   [in] UINT32             ulPublicInfoLength,
   [in] BYTE*              rgbPublicInfo,
   [out] UINT32*           keySize,
   [out, size_is(, *keySize)] BYTE** keyBlob
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TSS_ALGORITHM_ID	AlgID	Algorytm ID for public key
UINT32	ulPublicInfoLength	Size of buffer rgbPublicInfo
BYTE*	rgbPublicInfo	Buffer containing the public key
UINT32*	pcKeySize	Size of the returned keyblob in bytes.
BYTE**	prgbKey	Returned pointer to a byte stream containing the key blob of interest.

Comment:

The TCS allocates memory, assigns the memory resource to the given context and returns a registered key blob as defined by TCPA_KEY structure.

If a key has not been registered if a key has not been registered with the specified public key, the function fails with the error TCS_E_KEY_NOT_REGISTERED and returns a NULL pointer and *pcKeySize = 0.

Copyright TCG 2003

Return Value:

TSS_SUCCESS
TCS_E_KEY_NOT_REGISTERED

4.6.2.4 TCS Loading a Key

Start of informative comment:

After an application had loaded a key by utilizing the TSS Core Services (TCS), the application will receive an application key handle associated with that loaded key. If the application wants to use the loaded key later (ex. to sign or for a quote), the key will be addressed by that application key handle.

End of informative comment.

For the entity type of SRK the associated application key handle (TCS_KEY_HANDLE) MUST be 0x40000000.

4.6.2.4.1 Tcsip_LoadKeyByBlob

Start of informative comment:

A key can be loaded by providing a key blob as defined in the TCPA_KEY structure. The key defined by the key blob gets unwrapped by the already loaded parent key associated with the given application parent key handle. After the key is loaded an appropriate application key handle is returned by which the key can be addressed for further use. Depending on the parent key this can be done with or without required authorization.

This is a low level mechanism and the calling application must manage the required key blobs by its own but gives the caller as much flexibility as possible.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_LoadKeyByBlob
(
   TCS_CONTEXT_HANDLE   hContext,          // in
   TCS_KEY_HANDLE       hUnwrappingKey,    // in
   UINT32               cWrappedKeyBlobSize, // in
   BYTE*                rgbWrappedKeyBlob, // in
   TPM_AUTH*            pAuth,             // in, out
   TCS_KEY_HANDLE*      phKeyTCSI,         // out
   TCS_KEY_HANDLE*      phKeyHMAC          // out
);
```

IDL-Definition:

```
[helpstring("method Tcsip_LoadKeyByBlob")]
TSS_RESULT Tcsip_LoadKeyByBlob
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TCS_KEY_HANDLE     hUnwrappingKey,
   [in] UINT32             cWrappedKeyBlobSize,
   [in, size_is(cWrappedKeyBlobSize)] BYTE* rgbWrappedKeyBlob,
   [in, out] TPM_AUTH*     pAuth,
   [out] TCS_KEY_HANDLE*   phKeyTCSI,
   [out] TCS_KEY_HANDLE*   phKeyHMAC
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	HContext	Handle to established

		context.
TCS_KEY_HANDLE	hUnwrappingKey	Application key handle of the already loaded parent key.
UINT32	cWrappedKeyBlobSize	Size of the provided keyblob in bytes.
BYTE*	rgbWrappedKeyBlob	Key blob of the key to be loaded.
TCS_KEY_HANDLE*	phKeyTCSI	Return application key handle the loaded key can be addressed on further use.
TCS_KEY_HANDLE*	phKeyHMAC	Return TPM key handle required to evaluate the the returned HMAC digist. This TPM key handle can not be used to address the key on on further use.
TPM_AUTH*	pAuth	Authorization session data including the HMAC digest for using the unwrapping key.If NULL, no authorization is required.

Comment:

Tcsip_LoadKeyByBlob initially loads the key utilizing the Key Cache Manager Services (KCM) and returns a new created application key handle by which the key can be addressed on further use. The returned application key handle must be bound to the context provided by hContext.

After this command the key is managed by the Key Cache Management Services.

If pAuth == NULL, no authorization is required.

Loading a key must utilize the Key Cache Manager Service.

Return Value:

TCS_SUCCESS
TCS_E_FAIL

Copyright TCG 2003

4.6.2.4.2 Tcsip_LoadKeyByUUID

Start of informative comment:

A key can be loaded by only referring a UUID, if the key was registered in the TCS Persistent Storage Database (PS) before.

TCS Key Management Services will internally provide all the information stored in the PS that is required to load a key by only providing the appropriate UUID. The TCS will implicitly load all the required wrapping parent keys to get the key loaded addressed by the given UUID.

If no authorization is required, the additional required load of one or more parent keys is completely hidden from the calling application. If authorization is required for one or more parent keys the application can provide the authorization data in intermediate steps (see Key Management 2.7).

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_LoadKeyByUUID
(
   TCS_CONTEXT_HANDLE   hContext,     // in
   TSS_UUID             KeyUUID,      // in
   TCS_LOADKEY_INFO*    pLoadKeyInfo  // in, out
   TCS_KEY_HANDLE*      phKeyTCSI     // out
);
```

IDL-Definition:

```
[helpstring("method Tcsip_LoadKeyByUUID")]
TSS_RESULT Tcsip_LoadKeyByUUID
(
   [in] TCS_CONTEXT_HANDLE       hContext,
   [in] TSS_UUID                 KeyUUID,
   [in, out] TCS_LOADKEY_INFO*   pLoadKeyInfo
   [out] TCS_KEY_HANDLE*         phKeyTCSI
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	Hcontext	Handle to established context.
TSS_UUID	KeyUUID	UUID of the key to be loaded.
TCS_KEY_HANDLE*	PhKeyTCSI	Return application key handle the loaded key can be addressed on further use.
TCS_LOADKEY_INFO*	PloadKeyInfo	Information required to load a key if authorization is needed.

Comment:

The TCS Key Management Service utilizes the information stored in the system persistent storage. The service loads all required wrapping parent keys important to load that parent key which wraps the key addressed by KeyUUID.

When this last parent key is loaded, the key addressed by KeyUUID is initially loaded utilizing the Key Cache Manager Service and a new created application key handle is returned. Using that application

Copyright TCG 2003

key handle the key can be addressed on further use. The returned application key handle must be bound to the context given by hContext.

After that command the key is managed by the Key Cache Management Services.

If one of the required parent keys needs authorization, the load key information structure is filled with:

Type	Description
TCS_LOADKEY_INFO->hContext	Context as provided by Tcsip_LoadKeyByUUID call.
TCS_LOADKEY_INFO->keyUUID	UUID of key to be loaded next up the key hierarchy.
TCS_LOADKEY_INFO->parentKeyUUID	UUID of the parent key wrapping the key addressed by TCS_LOADKEY_INFO->keyUUID. This key requires authorization.
TCS_LOADKEY_INFO->paramDigest	Digest of the TPM_LoadKey command input parameters as defined in the TCPA 1.1b Main Specification.

The function call will return with the error TCS_E_KM_LOADFAILED.

The caller may use the returned information to:

- Start an OIAP authorization session
- Calculate the HMAC digest using TCS_LOADKEY_INFO->paramDigest and the started OIAP session required for loading the key addressed by TCS_LOADKEY_INFO->keyUUID and wrapped with the parent key addressed by TCS_LOADKEY_INFO->parentKeyUUID which requires authorization.
- Fill in the information in TCS_LOADKEY_INFO->authData
- Recall Tcsip_LoadKeyByUUID with the same input parameters and the completely filled TCS_LOADKEY_INFO structure

The Key Management Services now utilizes the provided TCS_LOADKEY_INFO->authData to call the TPM command TPM_LoadKey providing the appropriate authorization data to get that key loaded which is addressed by TCS_LOADKEY_INFO->keyUUID.

Loading a key must utilize the Key Cache Manager Service.

Return Value:

TCS_SUCCESS
TCS_E_KM_LOADFAILED
TCS_E_FAIL

Copyright TCG 2003

4.6.2.4.3 Tcsip_EvictKey

Start of informative comment:

Tcsip_EvictKey allows flushing a key from the cache managed by the Key Cache Manager Services.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_EvictKey
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TCS_KEY_HANDLE       hKey        // in
);
```

IDL-Definition:

```
[helpstring("method Tcsip_EvictKey")]
TSS_RESULT Tcsip_EvictKey
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TCS_KEY_HANDLE     hKey
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TCS_KEY_HANDLE	hKey	Application key handle to be evicted.

Comment:

Tcsip_EvictKey flushes the key addressed by hKey from the key cache managed by the TSS CS Key Management Services.

If key object addressed with hKey is not assigned to context addressed with hContext the function fails with the error TCS_E_INVALID_CONTEXTHANDLE

All resources bound to the application key handle must be released and the application key handle is not longer valid on return.

Return Value:

TCS_SUCCESS
TCS_E_INVALID_CONTEXTHANDLE
TCS_E_FAIL

Copyright TCG 2003

4.6.2.5 TCS Creating a Key

4.6.2.5.1 Tcsip_CreateWrapKey

Start of informative comment:

Tcsip_CreateWrapKey allows creating a new key, which is wrapped by the already loaded wrapping key addressed by hWrappingKey handle.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CreateWrapKey
(
   TCS_CONTEXT_HANDLE   hContext,          // in
   TCS_KEY_HANDLE       hWrappingKey,      // in
   TCPA_ENCAUTH         KeyUsageAuth,      // in
   TCPA_ENCAUTH         KeyMigrationAuth,  // in
   UINT32               keyInfoSize,       // in
   BYTE*                keyInfo,           // in
   TPM_AUTH*            pAuth              // in, out
   UINT32*              keyDataSize,       // out
   BYTE**               keyData,           // out
);
```

IDL-Definition:

```
[helpstring("method Tcsip_CreateWrapKey")]
TSS_RESULT Tcsip_CreateWrapKey
(
   [in]TCS_CONTEXT_HANDLE  hContext,          // in
   [in]TCS_KEY_HANDLE      hWrappingKey,      // in
   [in]TCPA_ENCAUTH        KeyUsageAuth,      // in
   [in]TCPA_ENCAUTH        KeyMigrationAuth,  // in
   [in]UINT32              keyInfoSize,       // in
   [in, size_is( keyInfoSize )]BYTE*  keyInfo,   // in
   [in, out]TPM_AUTH*      pAuth              // in, out
   [out]UINT32*            keyDataSize,       // out
   [out, size_is(, *keyDataSize )]BYTE**  keyData, //out
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TCS_KEY_HANDLE	hWrappingKey	Application key handle of the already loaded wrapping parent key.
TCPA_ENCAUTH	KeyUsageAuth	Encrypted usage authorization data for the key to be created.
TCPA_ENCAUTH	KeyMigrationAuth	Encrypted migration authorization data for the key to be created.
UINT32*	pcKeySize	Size of the provided/returned byte

		stream in bytes.
BYTE**	prgbKey	IN: Information about key to be created, pubkey.keyLength and pKey->encSize elements are 0. OUT: The key blob as defined in TCPA_KEY structure which includes the public and encrypted private key.
TPM_AUTH*	pAuth	Authorization session data including the HMAC digest for using the wrapping key. If NULL, no authorization is required.

Comment:

Tcsip_CreateWrapKey creates a new key as defined by the parameters provided by pKey. The new key gets a usage authorization secret and a migration authorization secret as given by the input parameters KeyUsageAuth and KeyMigrationAuth. Both secrets are encrypted as defined in the TCPA 1.1b Main Specification.

The TCS must utilize the TPM command TPM_CreateWrapKey to create and wrap the new key.

Return a key blob wrapped with the key addressed by the application key handle of the wrapping key, which must already be loaded.

All required memory resources to return the public key and the encrypted private key data must be allocated by TCS. The appropriate memory resources must be bound to the context provided by hContext.

If pAuth == NULL, no authorization for using the wrapping key is required.

Return Value:

TCS_SUCCESS
TCS_E_KM_LOADFAILED
TCS_E_FAIL

4.6.2.6 TCS Working with Keys

4.6.2.6.1 Tcsip_GetPubKey

Start of informative comment:

Tcsip_GetPubKey allows obtaining the public key data of a key loaded in the TPM. This information may have privacy concerns so the command must have authorization from the key owner.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_GetPubKey
(
   TCS_CONTEXT_HANDLE   hContext,     // in
   TCS_KEY_HANDLE       hKey,         // in
   TPM_AUTH*            pAuth,        // in, out
   UINT32*              pcPubKeySize, // out
   BYTE**               prgbPubKey    // out
);
```

IDL-Definition:

```
[helpstring("method Tcsip_GetPubKey")]
TSS_RESULT Tcsip_GetPubKey
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TCS_KEY_HANDLE     hKey,
   [in, out] TPM_AUTH*     pAuth,
   [out] UINT32*           pcPubKeySize
   [out, size_is(, *pcPubKeySize)] BYTE** prgbPubKey
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TCS_KEY_HANDLE	hKey	Application key handle of the loaded key.
UINT32*	pcPubKeySize	Size of the returned byte stream in bytes.
BYTE**	prgbPubKey	Returned pointer to a byte stream containing information about the public key of interest.
TPM_AUTH*	pAuth	Authorization session data including the HMAC digest for authorizing the key. If NULL, no authorization is required.

Comment:

Tcsip_GetPubKey obtains the public key value of a key loaded in the TPM and addressed by the application key handle. The TPM command TPM_GetPubKey must be utilized.

Return a blob containing the public key data to the caller.

All required memory resources to return the public key data, must be allocated by TSS CS. The appropriate memory resources must be bound to the context assigned to the application key handle provided by hKey.

If pAuth == NULL, no authorization for using the wrapping key is required.

If key object addressed with hKey or auth session addressed with pAuth is not assigned to context addressed with hContext the function fails with the error TCS_E_INVALID_CONTEXTHANDLE.

Return Value:

TCS_SUCCESS
TCS_E_KEY_CONTEXT_RELOAD
TCS_E_INVALID_CONTEXTHANDLE
TCS_E_FAIL

Copyright TCG 2003

4.6.2.7 TCS Credential Management

4.6.2.7.1 Tcsip_MakeIdentity

Start of informative comment:

Tcsip_MakeIdentity allows creating a TPM identity and additionally returns the endorsement credential, the platform credential and the conformance credential.

These three credentials are stored TCS vendor specific and are provided by Tcsip_MakeIdentity only. This simplifies the management of these credentials because each calling application can get this platform and system specific information from the appropriate system the information belongs to and ensures that the credentials are only provided if an owner authorization could be proofed.

Only the Owner of the TPM has the privilege of creating a TPM identity. This ensures that the credentials are provided to the caller only after a TPM owner authorization could be proofed.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_MakeIdentity
(
   TCS_CONTEXT_HANDLE   hContext,                    // in
   TCPA_ENCAUTH         identityAuth,                // in
   TCPA_CHOSENID_HASH      IDLabel_PrivCAHash,          // in
   UINT32               idIdentityKeyInfoSize,       // in
   BYTE*                idIdentityKeyInfo,           // in
   TPM_AUTH*            pSrkAuth,                    // in, out
   TPM_AUTH*            pOwnerAuth,                  // in, out
   UINT32*              idIdentityKeySize,           // out
   BYTE**               idIdentityKey,               // out
   UINT32*              pcIdentityBindingSize,       // out
   BYTE**               prgbIdentityBinding,         // out
   UINT32*              pcEndorsementCredentialSize, // out
   BYTE**               prgbEndorsementCredential,   // out
   UINT32*              pcPlatformCredentialSize,    // out
   BYTE**               prgbPlatformCredential,      // out
   UINT32*              pcConformanceCredentialSize, // out
   BYTE**               prgbConformanceCredential    // out
);
```

IDL-Definition:

```
[helpstring("method Tcsip_MakeIdentity")]
TSS_RESULT Tcsip_MakeIdentity
(
   [in]TCS_CONTEXT_HANDLE  hContext,                          // in
   [in]TCPA_ENCAUTH        identityAuth,                      // in
   [in]TCPA_CHOSENID_HASH  IDLabel_PrivCAHash,                // in
   [in]UINT32              idIdentityKeyInfoSize,             // in
   [in, size_is( idIdentityKeyInfoSize )]BYTE*
                                idIdentityKeyInfo,            // in
   [in, out]TPM_AUTH*      pSrkAuth,                          // in, out
   [in, out]TPM_AUTH*      pOwnerAuth,                        // in, out
   [out]UINT32*            idIdentityKeySize,                 // out
   [out, size_is(, *idIdentityKeySize )]BYTE**
                           idIdentityKey,                     // out
   [out]UINT32*            pcIdentityBindingSize,             // out
   [out, size_is(, *pcIdentityBindingSize )]BYTE**
                                prgbIdentityBinding,          // out
   [out]UINT32*            pcEndorsementCredentialSize,       // out
   [out, size_is(, *pcEndorsementCredentialSize )]BYTE**
                              prgbEndorsementCredential,      // out
   [out]UINT32*               pcPlatformCredentialSize,       // out
   [out, size_is(, *pcPlatformCredentialSize
                           )]BYTE**prgbPlatformCredential,    // out
   [out]UINT32*               pcConformanceCredentialSize,    // out
   [out, size_is(, *pcConformanceCredentialSize )]BYTE**
                           prgbConformanceCredential          // out
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TCPA_ENCAUTH	identityAuth	Encrypted usage authorization data for new identity.
TCPA_CHOSENID_HASH	IDLabel_PrivCAHash	The digest of the identity label and privacy CA chosen for the new TPM identity.
UINT32	pcIdentityKeySize	Size of the provided/returned byte stream in bytes.
BYTE**	prgbIdentityKey	IN: byte strem containing all parameters defining how to create the new identity key. OUT: new created identity key wrapped by SRK.
TPM_AUTH*	pSrkAuth	SRK authorization session data including the HMAC digest.If NULL, no authorization is

		required.
TPM_AUTH*	pOwnerAuth	Owner authorization session data including the HMAC digest. If NULL, no authorization is required.
UINT32*	pcIdentityBindingSize	Returned size of output for identity binding information in bytes.
BYTE**	prgbIdentityBinding	Return pointer to memory containing signature of TCPA_IDENTITY_CONTENTS using the private key of the new created identity key.
UINT32*	pcEndorsementCredentialSize	Returned size of output for endorsement credential information in bytes.
BYTE**	prgbEndorsementCredential	Return pointer to memory containing the endorsement credential.

Comment:

TCS processes the TPM_MakeIdentity command, which requires an owner authorization.

The TCS Credential Management Service must return the endorsement credential, the platform credential and the conformance credential only after a TPM identity was successfully created.

The TCS allocates memory resources for each of these credentials, the identity binding information, the public key and the encrypted private key data of the identity and binds the memory resources to the provided hContext.

Return Value:

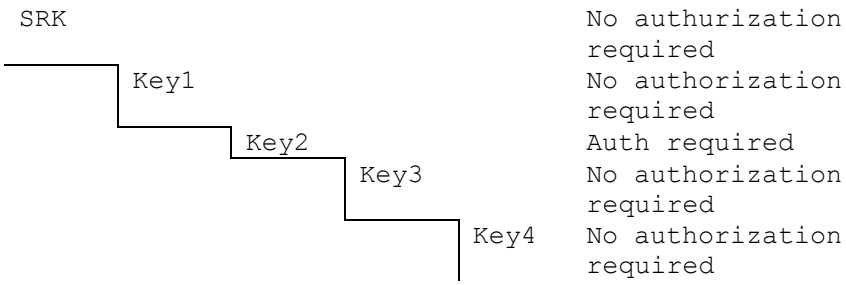
TCS_SUCCESS
TCS_E_FAIL

4.6.3 TCS Use Models

4.6.3.1 TCS Load Key by UUID

Assumptions:

- Key Hierarchy: all keys registered in Persisent Storage



- No key is loaded in TPM
- TCS knows that loading Key3 will fail since Key2 requires authorization.

Application Pseudo Code:

```
TCS_LOADKEY_INFO LoadKeyInfo
TSS_UUID keyUUID;
TCS_KEY_HANDLE tcsiKeyHandle;

// . . .

      // initialze TSS_UUID structure with UUID of key4
InitUuid (keyUUID, key4);
      // load key4 by UUID
while (TCS_E_KM_LOADFAILED == Tcsip_LoadKeyByUuid(hContext,
                                                  keyUUID,
                                                  &tcsiKeyHandle,
                                                  &LoadInfo))
{
     // 1) initialize an OIAP session
     // 2) calculate the authorization HMAC digest
     //      using the secret of key2 and LoadInfo.paramDigest
     // 3) fill in the auth session data and HMAC digest of step #2
     //      in LoadInfo.authData
}

// get public key of already loaded key4
// addressed by the application key handle tcsiKeyHandle,
// no authorization is required, since key4 requires no authorization

TCPA_PUBKEY* pPubKey = NULL;
Tcsip_GetPubKey(tcsiKeyHandle,
                NULL,
                &pPubKey);

// evict key4 from Key Cache Manager and
// free allocated resources assigned to tcsiKeyHandle
Tcsi_EvictKey(tcsiKeyHandle);
```


Copyright TCG 2003

```
// . . .
```

Comments:

```
TCS actions on first Tcsip_LoadKeyByUuid() call:
   -  load Key1, no authorization is required for SRK
   -  load Key2, no authorization is required for key1
   -  return from function call with a failure providing the
      TCS_LOADKEY_INFO blob (KeyUUID = Key3, parentKeyUUID = Key2).

TCS actions on second Tcsip_LoadKeyByUuid() call:
   -  load Key3 using the provided LOADKEY_INFO data
      (including the authortization data: authData)
   -  load Key4, no authorization is required for key3
   -  return from function call with TCS_SUCCESS
```

4.7 TCS Event Manager

4.7.1 TCS Event Manager Functions and Operations

Start of informative comment:

The TCS Event Log Services maintains the TCG Event Log. The TCS Event Log Services allow TPM extend PCR events to be logged, and allow challengers interested in extend PCR information contained in these logs, to access it.

End of informative comment.

TCS Event Manager structures and functions use the idioms: PcrEvent or TSS_PCR_EVENT (and not just “event”) in order to distinguish them from the TCS Audit Manager structures and functions, which also use the word event.

4.7.2 TCS Event Manager Interface

4.7.2.1 TCS Event Manager Interface Structures and Definitions

4.7.2.1.1 TCS The Event Log

Start of informative comment:

The TCS Event Log Services maintain a database of events called the Event Log. Conceptually, this log will consist of an array of events in which each entry is in the format of TSS_PCR_EVENT (defined below).

TCG defines certain event-type information (for instance, validation certificates). Other application-specific types may be added using the naming convention described.

The Event Log need not be held in TCG-shielded locations, and the logging and retrieval operations need not be TCG-protected capabilities. This is because servers or other software can detect tampering with the log.

The TCS is free to reallocate Event Log storage as it sees fit. The TCS also is free to maintain additional data structures that permit fast random access to events.

End of informative comment.

4.7.2.2 TCS Event Manager Interface Functions

Start of informative comment:

The Tcsi_LogPcrEvent operation adds a new event to the end of the array associated with the named PCR.

Logged information is retrieved using the Tcsi_GetPcrEvent, Tcsi_GetPcrEventsByPcr and Tcsi_GetPcrEventLog calls.

In Tcsi_GetPcrEvent, events are accessed by PCR index and number. Tcsi_GetPcrEventsByPcr returns a pointer to a data structure describing events related with a single PCR. Tcsi_GetPcrEventLog returns a pointer to a data structure that describes the entire log.

End of informative comment.

4.7.2.2.1 Tcsi_LogPcrEvent

Start of informative comment:

The Tcsi_LogPcrEvent operation adds a new event to the end of the array associated with the named PCR.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_LogPcrEvent
(
   TCS_CONTEXT_HANDLE   hContext,    // in
   TSS_PCR_EVENT        Event,       // in
   UINT32*              pNumber      // out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_LogPcrEvent")]
TSS_RESULT Tcsi_LogPcrEvent
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TSS_PCR_EVENT      Event
   [out] UINT32*           pNumber
);
```

Parameter Description:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TSS_PCR_EVENT	Event	Details of the event being logged.
UNIT32*	pNumber	The number of the event just logged is returned in this variable. The TCS number events for each PCR monotonically from 0.

Action:

The Tcsi_LogPcrEvent operation MUST add supporting information for the named TPM_Extend event to the end of the Event Log. The TCS MUST maintain an array of event-supporting data with events identified by the register to which they belong and the order in which the events occurred.

The log need not be in a TCG-shielded location, and the Tcsi_LogPcrEvent action need not be a TCG-protected capability. The TCS MUST NOT impose arbitrary size limitations on the size of the event log. The event log size should be limited by physical memory, memory accessible in the given operating mode, or memory allocated to the log by system firmware or other software.

TSS_PCR_EVENT→PCRValue should be the actual digest-sized event passed to TPM_Extend.

Return Value:

TCS_SUCCESS
TCS_BAD_INDEX
TCS_E_BAD_PARAMETER
TSS_E_OUTOFMEMORY
TCS_E_FAIL

Copyright TCG 2003

4.7.2.2.2 Tcsi_GetPcrEvent

Start of informative comment:

Tcsi_GetPcrEvent is used to retrieve events logged with Tcsi_LogPcrEvent .

Tcsi_GetPcrEvent need not be a TCG-protected capability, and the log events retrieved need not be in TCG-shielded locations. Tcsi_GetPcrEvent returns all the data stored in TSS_PCR_EVENT.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_GetPcrEvent
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   UINT32               PcrIndex,   // in
   UINT32*              pNumber,    // in, out
   TSS_PCR_EVENT**      ppEvent     // out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_GetPcrEvent")]
TSS_RESULT Tcsi_GetPcrEvent
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] UINT32             PcrIndex,
   [in, out] UINT32*       pNumber,
   [out] TSS_PCR_EVENT**   ppEvent
);
```

Parameter Description:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
UINT32	PcrIndex	The index of the PCR.
UINT32*	pNumber	Number of event required. Events are numbered from 0 to the number of events logged on the named PCR.
TSS_PCR_EVENT**	ppEvent	Pointer to the returned event.

Actions:

The Tcsi_GetPcrEvent operation retrieves events previously logged using Tcsi_LogPcrEvent. The format of the data returned is identical to that previously logged. This operation interface retrieves log entries by PCR index and event number. On TCS initialization the event log for each PCR is empty. Then, for each PCR, the first event logged is numbered 0; the next is numbered 1, and so on. Attempts to receive log items beyond the end of the log return an error.

The Tcsi_GetPcrEvent allocates memory for the event and returns a pointer to the event. The caller can free this memory by calling the CS function: FreeMemory.

If ppEvent == NULL, Tcsi_GetPcrEvent returns the number of actually logged events in pNumber.

Note that that the event log is required to be accessible in the form of an array. TCS implementation MAY choose to provide supplemental data structures to make random array access through Tcsi_GetPcrEvent more efficient.

Copyright TCG 2003

Return Value:

TCS_SUCCESS
TCS_BAD_INDEX
TCS_SIZE
TCS_E_FAIL

Copyright TCG 2003

4.7.2.2.3 Tcsi_GetPcrEventsByPcr

Start of informative comment:

Tcsi_GetPcrEventsByPcr returns an event log bound to a single PCR. The event log is returned as an ordered sequence of TSS_PCR_EVENT structures.

The caller can limit the size of the returned array using EventCount. The caller can also specify the number of the first event on the returned event log using FirstEvent. These controls allow the caller to retrieve the event log step by step, or to retrieve a partial event log when required.

The array elements are of variable size, and the TSS_PCR_EVENT structure defines the size of the current event and the register with which it is associated. This data structure is not required to be thread-safe, so upper-level software should ensure that it is not modified during parsing. If the event log is kept in a TCG-shielded location, then a copy must be made in an unprotected area that can be traversed by non-TPM protected calling code.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_GetPcrEventsByPcr
(
   TCS_CONTEXT_HANDLE   hContext,     // in
   UINT32               PcrIndex,     // in
   UINT32               FirstEvent,   // in
   UINT32*              pEventCount,  // in,out
   TSS_PCR_EVENT**      ppEvents      // out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_GetPcrEventsByPcr")]
TSS_RESULT Tcsi_GetPcrEventByPcr
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] UINT32             PcrIndex,
   [in] UINT32             FirstEvent,
   [in, out] UINT32*       pEventCount,
   [out, size_is(, *pEventCount)] TSS_PCR_EVENT** ppEvents
);
```

Parameter Description:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
UINT32	PcrIndex	The index of the PCR.
UINT32	FirstEvent	The number of the first event in the returned array.
UINT32*	pEventCount	In: max number of events to be returned. Out: actual number of events in the returned array.
TSS_PCR_EVENT**	ppEvents	Pointer to the event log returned as an ordered sequence of TSS_PCR_EVENT structures.

Copyright TCG 2003

Actions:

This command returns a pointer to a “Partial PCR Event Log”, which is an array reflecting a list of events bound to a single PCR, starting withevent number: FirstEvent. The first event of the PCR event log is indexed with FirstEvent=0.

The size of the “Partial PCR Event Log” is determined by the number of events bound to that PCR, and the input value of EventCount:

If EventCount is set to –1, or if EventCount is greater than the actual number of events (related to that PCR and numbered above FirstEvent), the “Partial PCR Event Log” will consist all events related to that PCR and numbered above FirstEvent. In this case the command sets EventCount to that actual size.

If EventCount is smaller than the number of events (related to that PCR and numbered above FirstEvent) the command will return the first “EventCount” number of events starting with the event numbered: FirstEvent. In this case the returned EventCount is the same as the input EventCount.

If FirstEvent points to an event number that does not exist for that PCR, the command fails with the error code TCS_E_BAD_PARAMETER.

Return Value:

TCS_SUCCESS
TCS_BAD_INDEX
TCS_E_BAD_PARAMETER
TCS_SIZE
TCS_E_FAIL

4.7.2.2.4 Tcsi_GetPcrEventLog

Start of informative comment:

Tcsi_GetPcrEventLog returns the event log of all events since the TPM was initialized. The event log is returned as an ordered sequence of TSS_PCR_EVENT structures in the following order: all events bound to PCR 0 (in the order they have arrived), all events bound to PCR 1 (in the order they have arrived), etc.

The array elements are of variable size, and the TSS_PCR_EVENT structure defines the size of the current event and the register with which it is associated. This data structure is not required to be thread-safe, so upper-level software should ensure that it is not modified during parsing.. If the event log is kept in a TCG-shielded location, then a copy must be made in an unprotected area that can be traversed by non-TPM protected calling code.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_GetPcrEventLog
(
   TCS_CONTEXT_HANDLE   hContext,     // in
   UINT32*              pEventCount,  // out
   TSS_PCR_EVENT**      ppEvents,     // out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_GetPcrEventLog")]
TSS_RESULT Tcsi_GetPcrEventLog
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [out] UINT32*           pEventCount,
   [out, size_is(, *pEventCount)] TSS_PCR_EVENT** ppEvents
);
```

Parameter Description:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
UINT32*	pEventCount	Number of entries in the entire Event Log is returned in this variable.
TSS_PCR_EVENT**	ppEvents	Pointer to the head of the Event Log data structures.

Action:

This command returns to the caller the complete Event Log. When the Event Log is empty, Tcsi_GetPcrEventLog will return TCS_SUCCESS with EventCount set to 0.

The returned Event Log MUST consists of pointers to all events, in the following order (N indicated the number of PCRs in a particular TPM):

- All events bound to PCR 0, in increasing order starting from event number 0 of PCR 0.
- All events bound to PCR 1, in increasing order starting from event number 0 of PCR 1.
- All events bound to PCR N-1, in increasing order starting from event number 0 of PCR N-1.

Copyright TCG 2003

Return Value:

TCS_SUCCESS
TCS_SIZE
TCS_E_FAIL

4.8 TCS Audit Manager

4.8.1 TCS Audit Manager Functions and Operations

Start of informative comment:

End of informative comment.

Audit Manager is not supported in this version of the TSS Specification.

4.9 TCS TPM Parameter Block Generator

4.9.1 TCS TPM Parameter Block Generator Functions and Operations

Start of informative comment:

The TPM parameter block generator is a functional block within TSS Core Services (TCS). External access to this block is via the TSS Core Services Interface (TCSI). Direct access to the TPM device from applications is not permitted. The TPM parameter block generator is responsible for serializing, synchronizing, and processing TPM commands. This block builds byte streams to input to the TPM and converts byte streams output from the TPM. This block provides access to the TPM commands for TCG applications via the TCS platform service.

The TPM parameter block generator also contains internal interfaces to the Key and Credential Manager, Audit Manager, and Event Manager functional blocks. Interaction with these TCS blocks is required to support TPM data management, in and out of the TPM device. These blocks require knowledge of the data that is passing in/out of the TPM. Since these interfaces are internal to the TCS platform service, they are not discussed in this document.

End of informative comment.

4.9.2 TCS TPM Parameter Block Generator Interface

4.9.2.1 Functions

Start of informative comment:

The TPM parameter block generator exposes protected TPM commands through a set of interfaces. TPM functionality is exposed through the Tcsi via the TPM parameter block generator functional block. All TPM commands must pass through the TPM parameter block generator. This function set consists of TPM functions that TCG-enabled and TPM management applications require. The TPM parameter block generator communicates with the TPM using the low-level TPM diver via the TPM DDL. The function declarations, C-style and IDL, are defined in this document, while the TPM parameter block definitions are documented in the TCPA 1.1b Main Specification.

Interface Types:

Every function must be defined in C, IDL, and TSS Parameter Block format. Individual computing platforms do not need to expose every interface type. A default interface type must be decided for each platform. For instance, the default Tcsi interface for a PC may be a C-style interface. Any additional interfaces could be implemented by adding encoder/decoder modules that plug into the C-style interface. The possibility of all interfaces must exist to support multiple platform types.

Authorization:

The TPM parameter block generator does not have knowledge of any authorization secrets for protected services. Therefore, all authorization data associated with the TPM function must be generated outside of the TCS. This includes nonce generation, HMAC (Auth Data) generation, and TPM response validation.

Context Control:

Before performing any TPM command, an application must first call Tcsi_OpenContext to obtain a context for a TPM command session. The application controls the memory resources during the session using Tcsi_FreeMemory and must close the context using Tcsi_CloseContext when it is finished.

End of informative comment.

For all functions in this section TSS_RESULT is the result from the TPM command. For the command ordinal and return values see the TCPA 1.1b Main Specification.

Copyright TCG 2003

4.9.2.2 TPM Ownership, Authorization, and Identity

4.9.2.2.1 Tcsip_SetOwnerInstall

Start of informative comment:

Tcsip_SetOwnerInstall determines if the TPM has a current owner. The TPM validates the assertion of physical access and then sets the value of TCPA_PERSISTENT_FLAGS -> ownership to the value in state.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_SetOwnerInstall
(
   TCS_CONTEXT_HANDLE  hContext,  // in
   TSS_BOOL            state      // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_SetOwnerInstall")]
TSS_RESULT Tcsip_SetOwnerInstall
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TSS_BOOL           state
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_BOOL	state	New disable flag state.

Comment:

TPM command – TPM_SetOwnerInstall
TPM ordinal – TPM_ORD_SetOwnerInstall

4.9.2.2.2 Tcsip_TakeOwnership

Start of informative comment:

Tcsip_TakeOwnership inserts the Owner-authorization data and creates a new Storage Root Key (SRK). This function fails if there is already an TPM Owner set.

After inserting the authorization data, this function creates the SRK.

To validate that the operation completes successfully, the TPM HMACs the response to the Tcsip_TakeOwnership function.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_TakeOwnership
(
   TCS_CONTEXT_HANDLE   hContext,         // in
   UINT16               protocolID,       // in
   UINT32               encOwnerAuthSize, // in
   BYTE*                encOwnerAuth,     // in
   UINT32               encSrkAuthSize,   // in
   BYTE*                encSrkAuth,       // in
   UINT32               srkKeyInfoSize,   // in
   BYTE*                srkKeyInfo,       // in
   TPM_AUTH*            ownerAuth         // in, out
   UINT32*              srkKeyDataSize,   // out
   BYTE**               srkKeyData        // out
);
```

IDL-Definition:

```
[helpstring("method Tcsip_TakeOwnership")]
TSS_RESULT Tcsip_TakeOwnership
(
   [in]TCS_CONTEXT_HANDLE   hContext,          // in
   [in]UINT16               protocolID,        // in
   [in]UINT32               encOwnerAuthSize,  // in
   [in, size_is( encOwnerAuthSize )]BYTE*
                            encOwnerAuth,      // in
   [in]UINT32               encSrkAuthSize,    // in
   [in, size_is( encSrkAuthSize )]BYTE*
                            encSrkAuth,        // in
   [in]UINT32               srkKeyInfoSize,    // in
   [in, size_is( srkKeyInfoSize )]BYTE*
                            srkKeyInfo,        // in
   [in, out]TPM_AUTH*       ownerAuth          // in, out
   [out]UINT32*             srkKeyDataSize,    // out
   [out, size_is( srkKeyDataSize )]BYTE**
                            srkKeyData         // out
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	HContext	Handle of established context.

Copyright TCG 2003

UINT32	ProtocolID	The ownership protocol in use.
UINT32	encOwnerAuthSize	The size of the encrypted owner authorization data.
BYTE*	encOwnerAuth	The encrypted owner authorization data.
UINT32	encSrkAuthSize	The size of the encrypted SRK authorization data.
BYTE*	EncSrkAuth	The encrypted Storage Root Key (SRK) authorization data.
UINT32*	SrkSize	The size of the TCPA_KEY byte stream
BYTE**	Srk	TCPA_KEY byte stream of the storage root key blob
TPM_AUTH*	OwnerAuth	The authorization from the TPM Owner. There is no validation of in parameters, just validation on the return that the proper authorization data was used. HMAC Key: the new ownerAuth value.

Comment:

TPM command – TPM_TakeOwnership
TPM ordinal – TPM_ORD_TakeOwnership

Copyright TCG 2003

4.9.2.2.3 Tcsip_OIAP

Start of informative comment:

Tcsip_OIAP allows the creation of an authorization handle and the tracking of the handle by the TPM. The TPM generates the handle and nonce.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_OIAP
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TCS_AUTHHANDLE*      authHandle, // out
   TCPA_NONCE*          nonce0      // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_OIAP")]
TSS_RESULT Tcsip_OIAP
(
   [in] TCS_CONTEXT_HANDLE  hContext,
   [out] TCS_AUTHHANDLE*    authHandle,
   [out] TCPA_NONCE*        nonce0
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_AUTHHANDLE*	authHandle	Handle that TPM creates that points to the authorization state. The value is TPM specific and has no meaning except to identity the session.
TCPA_NONCE*	nonce0	Nonce generated by TPM and associated with session.

Comment:

TPM command – TPM_OIAP
TPM ordinal – TPM_ORD_OIAP

4.9.2.2.4 Tcsip_OSAP

Start of informative comment:

TPM_OSAP creates the authorization handle, the shared secret and generates nonceEven and nonceEvenOSAP.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_OSAP
(
   TCS_CONTEXT_HANDLE   hContext,     // in
   TCPA_ENTITY_TYPE     entityType,   // in
   UINT32               entityValue,  // in
   TCPA_NONCE           nonceOddOSAP, // in
   TCS_AUTHHANDLE*      authHandle,   // out
   TCPA_NONCE*          nonceEven,    // out
   TCPA_NONCE*          nonceEvenOSAP // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_OSAP")]
TSS_RESULT Tcsip_OSAP
(
   [in]  TCS_CONTEXT_HANDLE  hContext,
   [in]  TCPA_ENTITY_TYPE    entityType,
   [in]  UINT32                 entityValue,
   [in]  TCPA_NONCE          nonceOddOSAP,
   [out] TCS_AUTHHANDLE*     authHandle,
   [out] TCPA_NONCE*         nonceEven,
   [out] TCPA_NONCE*         nonceEvenOSAP
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_ENTITY_TYPE	entityType	The type entity in use.
UINT32	entityValue	The selection value based on entity type.
TCPA_NONCE	nonceOddOSP	The nonce generated by the caller associated with shared secret.
TCS_AUTHHANDLE*	authHandle	Handle which points to an authorization state.
TCPA_NONCE*	nonceEven	Nonce generated by TPM and associated with a session.
TCPA_NONCE*	nonceEvenOSAP	Nonce generated by TPM and associated with shared secret.

Comment:

TPM command – TPM_OSAP
TPM ordinal – TPM_ORD_OSAP

Copyright TCG 2003

4.9.2.2.5 Tcsip_ChangeAuth

Start of informative comment:

Tcsip_ChangeAuth allows the owner of an entity to change the authorization data for the entity.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ChangeAuth
(
   TCS_CONTEXT_HANDLE   contextHandle, // in
   TCS_KEY_HANDLE       parentHandle,  // in
   TCPA_PROTOCOL_ID     protocolID,    // in
   TCPA_ENCAUTH         newAuth,       // in
   TCPA_ENTITY_TYPE     entityType,    // in
   UINT32               encDataSize,   // in
   BYTE*                encData,       // in
   TPM_AUTH*            ownerAuth,     // in, out
   TPM_AUTH*            entityAuth,    // in, out
   UINT32*              outDataSize,   // out
   BYTE**               outData        // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ChangeAuth")]
TSS_RESULT Tcsip_ChangeAuth
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [in] TCS_KEY_HANDLE              parentHandle,
   [AUTH, in] TCPA_PROTOCOL_ID      protocolID,
   [AUTH, in] TCPA_ENCAUTH          newAuth,
   [AUTH, in] TCPA_ENTITY_TYPE      entityType,
   [AUTH, in] UINT32                encDataSize,
   [AUTH, in, size_is(endDataSize)] BYTE*   encData,
   [in, out] TPM_AUTH*              ownerAuth,
   [in, out] TPM_AUTH*              entityAuth,
   [AUTH, out] UINT32*              outDataSize,
   [AUTH, out, size_is(, *outDataSize)] BYTE** outData
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	parentHandle	Handle of the parent key to the entity.
TCPA_PROTOCOL_ID	protocolID	The ownership protocol in use.
TCPA_ENCAUTH	newAuth	The encrypted new authorization data for the entity. The encryption key is the shared secret from the OS-AP protocol.
TCPA_ENTITY_TYPE	entityType	The type of entity to be modified.
UINT32	encDataSize	The size of the encrypted entity.
BYTE	encData	The encrypted entity that is to be

Copyright TCG 2003

		modified.
UINT32*	outDataSize	The size of the modified encrypted entity.
BYTE**	outData	The modified encrypted entity.
TPM_AUTH*	ownerAuth	The authorization and inputs from the TPM owner. There is no validation of in parameters, just validation on the return that the proper authorization data was used. HMAC key: parentKey.usageAuth.
TPM_AUTH*	entityAuth	The authorization and inputs from the encrypted entity. There is no validation of in parameters, just validation on the return that the proper authorization data was used. HMAC key: entity.usageAuth.

Comment:

TPM command – TPM_ChangeAuth
TPM ordinal – TPM_ORD_ChangeAuth

If the entity to be changed is a key the Key Manager SHALL replace key blobs associated with this key with *new key blobs that contain the new authorization value.*

Note: One method for finding the associated key would be to compare the value (or hash of the value) of encData with values stored in the Key Manager.

Copyright TCG 2003

4.9.2.2.6 Tcsip_ChangeAuthOwner

Start of informative comment:

Tcsip_ChangeAuthOwner allows the owner of an entity to change the authorization data for the TPM Owner or the SRK.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ChangeAuthOwner
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TCPA_PROTOCOL_ID     protocolID, // in
   TCPA_ENCAUTH         newAuth,    // in
   TCPA_ENTITY_TYPE     entityType, // in
   TPM_AUTH*            ownerAuth   // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ChangeAuthOwner")]
TSS_RESULT Tcsip_ChangeAuthOwner
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [AUTH, in] TCPA_PROTOCOL_ID      protocolID,
   [AUTH, in] TCPA_ENCAUTH          newAuth,
   [AUTH, in] TCPA_ENTITY_TYPE      entityType,
   [in, out] TPM_AUTH*              ownerAuth
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_PROTOCOL_ID	protocolID	The ownership protocol in use.
TCPA_ENCAUTH	newAuth	The encrypted new authorization data for the entity. The encryption key is the shared secret from the OS-AP protocol.
TCPA_ENTITY_TYPE	entityType	The type of entity to be modified.
TPM_AUTH*	ownerAuth	The authorization and inputs for OwnreHandle. There is no validation of in parameters, just validation on the return that the proper authorization data was used. HMAC key: tpmOwnerAuth. This is the new tpmOwnerAuth value if this command changed that value.

Comment:

TPM command – TPM_ChangeAuthOwner
TPM ordinal – TPM_ORD_ChangeAuthOwner

Copyright TCG 2003

4.9.2.2.7 Tcsip_ChangeAuthAsymStart

Start of informative comment:

Tcsip_ChangeAuthAsymStart starts the process of changing authorization for an entity. It sets up an OI-AP session that must be retained for use by its twin Tcsip_ChangeAuthAsymFinish command.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ChangeAuthAsymStart
(
   TCS_CONTEXT_HANDLE   hContext,         // in
   TCS_KEY_HANDLE       idHandle,         // in
   TCPA_NONCE           antiReplay,       // in
   UINT32               TempKeyInfoSize,  // in
   BYTE*                TempKeyInfoData,  // in
   TPM_AUTH*            pAuth,            // in, out
   UINT32*              TempKeySize,      // out
   BYTE**               TempKeyData,      // out
   UINT32*              CertifyInfoSize,  // out
   BYTE**               CertifyInfo,      // out
   UINT32*              sigSize,          // out
   BYTE**               sig,              // out
   TCS_KEY_HANDLE*      ephHandle         // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ChangeAuthAsymStart")]
TSS_RESULT Tcsip_ChangeAuthAsymStart
(
   [in]TCS_CONTEXT_HANDLE  hContext,         // in
   [in]TCS_KEY_HANDLE      idHandle,         // in
   [in]TCPA_NONCE          antiReplay,       // in
   [in]UINT32              TempKeyInfoSize,  // in
   [in, size_is( TempKeyInfoSize )]BYTE*
                           TempKeyInfoData,  // in
   [in]TPM_AUTH*           pAuth,            // in, out
   [out]UINT32*            TempKeySize,      // out
   [out, size_is(, *TempKeySize )]BYTE**
                           TempKeyData,      // out
   [out]UINT32*            CertifyInfoSize,  // out
   [out, size_is(, *CertifyInfoSize )]BYTE**
                           CertifyInfo,      // out
   [out]UINT32*            sigSize,          // out
   [out, size_is(, *sigSize )]BYTE**
                           sig,              // out
   [out]TCS_KEY_HANDLE*    ephHandle         // out
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	idHandle	Handle to identify loaded identity ID key.

Copyright TCG 2003

TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
UINT32*	TempKeySize	Size of TCPA_KEY byte stream of ephemeral key
BYTE**	TempKey	TCPA_KEY byte stream of ephemeral key
UINT32*	CertifyInfoSize	Size of TCPA_CERTIFY_INFO byte stream
BYTE**	CertifyInfo	TCPA_CERTIFY_INFO byte stream that was signed
UINT32*	sigSize	Used size of the output area for the signature.
BYTE**	sig	Signature of the certify info parameter.
TCS_KEY_HANDLE*	ephHandle	keyHandle identifier to be used by ChangeAuthAsymFinish for the ephemeral.
TPM_AUTH*	pAuth	The authorization and inputs from the TPM owner. There is no validation of in parameters, just validation on the return that the proper authorization data was used. HMAC key: parentKey.usageAuth.

Comment:

TPM command – TPM_ChangeAuthAsymStart
TPM ordinal – TPM_ORD_ChangeAuthAsymStart

4.9.2.2.8 Tcsip_ChangeAuthAsymFinish

Start of informative comment:

TPM_ChangeAuthAsymFInish completes the process of changing authorization for an entity. The owner uses tempKkey to encrypt the desired new authorization data and inserts that encrypted data in a TPM_ChangeAuthAsymFinish command, in the knowledge that only a TPM with a specific identity can interpret the new authorization data.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ChangeAuthAsymFinish
(
   TCS_CONTEXT_HANDLE   hContext,         // in
   TCS_KEY_HANDLE       parentHandle,     // in
   TCS_KEY_HANDLE       ephHandle,        // in
   TCPA_ENTITY_TYPE     entityType,       // in
   TCPA_HMAC            newAuthLink,      // in
   UINT32               newAuthSize,      // in
   BYTE*                encNewAuth,       // in
   UINT32               encDataSizeIn,    // in
   BYTE*                encDataIn,        // in
   TPM_AUTH*            ownerAuth,        // in, out
   UINT32*              encDataSizeOut,   // out
   BYTE**               encDataOut,       // out
   TCPA_NONCE*          saltNonce,        // out
   TCPA_DIGEST*         changeProof       // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ChangeAuthAsymFinish")]
TSS_RESULT Tcsip_ChangeAuthAsymFinish
(
   [in]TCS_CONTEXT_HANDLE  hContext,         // in
   [in]TCS_KEY_HANDLE      parentHandle,     // in
   [in]TCS_KEY_HANDLE      ephHandle,        // in
   [in]TCPA_ENTITY_TYPE    entityType,       // in
   [in]TCPA_HMAC           newAuthLink,      // in
   [in]UINT32              newAuthSize,      // in
   [in, size_is( newAuthSize )]BYTE*
                           encNewAuth,       // in
   [in]UINT32              encDataSizeIn,    // in
   [in, size_is( encDataSizeIn )]BYTE*
                           encDataIn,        // in
   [in, out]TPM_AUTH*      ownerAuth,        // in, out
   [out]UINT32*            encDataSizeOut,   // out
   [out, size_is(, *encDataSizeOut )]BYTE**
                           encDataOut,       // out
   [out]TCPA_NONCE*        saltNonce,        // out
   [out]TCPA_DIGEST*       changeProof       // out
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	parentHandle	Key handle of the parent key for input data.
TCS_KEY_HANDLE	ephHandle	Key handle identifier for the ephemeral key.
TCPA_ENTITY_TYPE	entityType	Type of entity to be modified.
TCPA_HMAC	newAuthLink	HMAC calculation that links the new and old authorization values together.
UNINT32	newAuthSize	Size of new authorization data and ephemeral key.
BYTE*	encNewAuth	New authorization data and ephemeral key.
TCPA_NONCE*	saltNonce	Nonce value from TPM RNG to add entropy to the changeProof value.
TCPA_DIGEST*	changeProof	Proof that authorization data has changed.
UINT32*	encDataSize	Encrypted entity data size.
BYTE**	encData	Encrypted entity data – input -> modified output.
TPM_AUTH*	ownerAuth	The authorization and inputs from the TPM owner. There is no validation of in parameters, just validation on the return that the proper authorization data was used. HMAC key: parentKey.usageAuth.

Comment:

TPM command – TPM_ChangeAuthAsymFinish
TPM ordinal – TPM_ORD_ChangeAuthAsymFinish

4.9.2.2.9 Tcsip_TerminateHandle

Start of informative comment:

Tcsip_TerminateHandle allows the TPM driver to clear out information in an authorization handle.

The TPM may maintain the authorization session even though a key attached to it has been unloaded or the authorization session itself has been unloaded in some way.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_TerminateHandle
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TCS_AUTHHANDLE       handle      // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_TerminateHandle")]
TSS_RESULT Tcsip_TerminateHandle
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TCS_AUTHHANDLE     handle
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_AUTHHANDLE	handle	The handle to terminate.

Comment:

TPM command – TPM_Terminate_Handle
TPM ordinal – TPM_ORD_Terminate_Handle

Copyright TCG 2003

4.9.2.2.10 Tcsip_ActivateTPMIdentity

Start of informative comment:

Tcsip_ActivateTPMIdentity purpose is twofold. The first purpose is to obtain assurance that the credential in the TCPA_SYM_CA_ATTESTATION is for this TPM. The second purpose is to obtain the session key used to encrypt the TCPA_IDENTITY_CREDENTIAL.

This function checks that the symmetric session key corresponds to a TPM-identity before releasing that session key.

Only the Owner of the TPM has the privilege of activating a TPM identity. The owner may authorize this function using either the TPM_OIAP or TPM_OSAP authorization protocols.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ActivateTPMIdentity
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TCS_KEY_HANDLE       idKey,   // in
   UINT32               blobSize,   // in
   BYTE*                blob, // in
   TPM_AUTH*            idKeyAuth,  // in, out
   TPM_AUTH*            ownerAuth,  // in, out
   UINT32*              SymmetricKeySize, // out
   BYTE**               SymmetricKey  // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ActivateTPMIdentity")]
TSS_RESULT Tcsip_ActivateTPMIdentity
(
   [in] TCS_CONTEXT_HANDLE    hContext,
   [in] TCS_KEY_HANDLE        idKey,
   [AUTH, in]                 blobSize,
   [AUTH, in, size_is(blobSize)] BYTE* blob,
   [AUTH, in, out] TPM_AUTH*  idKeyAuth,
   [AUTH, in, out] TPM_AUTH*  ownerAuth,
   [AUTH, out] UINT32*        SymmetricKeySize,
   [AUTH, out, size_is(, *SymmetricKeySize)] BYTE** SymmetricKey
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	HContext	Handle of established context.
TCS_KEY_HANDLE	IdKey	Identity key to be activated.
UINT32	BlobSize	Size of encrypted blob from CA.
BYTE*	Blob	Encrypted ASYM_CA_CONTENTS structure.
UINT32	SymmetricKey Size	Size of decrypted TCPA_SYMMETRIC_KEY byte stream
BYTE**	SymmetricKey	decrypted TCPA_SYMMETRIC_KEY byte stream
TPM_AUTH*	IdKeyAuth	The authorization and inputs from the TPM owner. There is no validation of

		in parameters, just validation on the return that the proper authorization data was used. HMAC key: idKey.usageAuth.
TPM_AUTH*	OwnerAuth	The authorization and inputs from the TPM owner. There is no validation of in parameters, just validation on the return that the proper authorization data was used. HMAC key: ownerAuth.

Comment:

TPM command – TPM_ActivateTPMIdentity
TPM ordinal – TPM_ORD_ActivateTPMIdentity

4.9.2.3 TPM Mandatory

4.9.2.3.1 Tcsip_Extend

Start of informative comment:

Tcsip_Extend causes the modification of a specific PCR register.

End of informative comment.

C-Definition:

TSS_RESULT Tcsip_Extend

```
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TCPA_PCRINDEX        pcrNum,     // in
   TCPA_DIGEST          inDigest,   // in
   TCPA_PCRVALUE*       outDigest   // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Extend")]
TSS_RESULT Tcsip_Extend
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TCPA_PCRINDEX      pcrNum,
   [in] TCPA_DIGEST        inDigest,
   [out] TCPA_PCRVALUE*    outDigest
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_PCRINDEX	pcrNum	Index of the PCR to be modified.
TCPA_DIGEST	inDigest	160-bit value representing the event to be recorded.
TCPA_PCRVALUE*	outDigest	Pointer to a DIGEST-sized memory location that is updated by the TPM_Extend operation to be the contents of the named PCR when internal processing is complete. If this parameter is NULL, no value is returned. If the TPM is disabled, NULL is returned.

Comment:

TPM command – TPM_Extend
TPM ordinal – TPM_Extend

Copyright TCG 2003

4.9.2.3.2 Tcsip_PcrRead

Start of informative comment:

Tcsip_PcrRead provides non-cryptographic reporting of the contents of a named PCR.

End of informative comment

C-Definition:

```
TSS_RESULT Tcsip_PcrRead
(
   TCS_CONTEXT_HANDLE   hContext,    // in
   TCPA_PCRINDEX        pcrNum,      // in
   TCPA_PCRVALUE*       outDigest    // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_PcrRead")]
TSS_RESULT Tcsip_PcrRead
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TCPA_PCRINDEX      pcrNum,
   [out] TCPA_PCRVALUE*    outDigest
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_PCRINDEX	pcrNum	Index of the PCR to be read.
TCPA_PCRVALUE *	outDigest	Pointer to the current contents of the named PCR.

Comment:

TPM command – TPM_PcrRead
TPM ordinal – TPM_PcrRead

4.9.2.3.3 Tcsip_Quote

Start of informative comment:

Tcsip_Quote provides cryptographic reporting of PCR values. A loaded key is required for operation. Tcsip_Quote uses a key to sign a statement that names the current value of a chosen PCR and externally supplied data (which may be a nonce supplied by a Challenger).

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Quote
(
   TCS_CONTEXT_HANDLE   hContext,      // in
   TCS_KEY_HANDLE       keyHandle,     // in
   TCPA_NONCE           antiReplay,    // in
   UINT32               pcrTargetSize, // in
   BYTE*                pcrTarget,     // in
   TPM_AUTH*            privAuth,      // in, out
   UINT32*              pcrDataSize,   // out
   BYTE**               pcrData,       // out
   UINT32*              sigSize,       // out
   BYTE**               sig            // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Quote")]
TSS_RESULT Tcsip_Quote
(
   [in]TCS_CONTEXT_HANDLE  hContext,      // in
   [in]TCS_KEY_HANDLE      keyHandle,     // in
   [in]TCPA_NONCE          antiReplay,    // in
   [in]UINT32              pcrTargetSize, // in
   [in, size_is( pcrTargetSize )]BYTE*
                           pcrTarget,     // in
   [in, out]TPM_AUTH*      privAuth,      // in, out
   [out]UINT32*            pcrDataSize,   // out
   [out, size_is(, *pcrDataSize )]BYTE**
                           pcrData,       // out
   [out]UINT32*            sigSize,       // out
   [out, size_is( *sigSize )]BYTE**
                           sig            // out
);
```

Parameters:

Return Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	Handle associated with Key used to provide the Quote.
TCPA_NONCE	antiReplay	Nonce provided to fight replay attacks.
UINT32*	PcrDataSize	Size of TCPA_PCR_COMPOSITE byte stream getting signed
BYTE**	PcrData	TCPA_PCR_COMPOSITE byte stream getting signed

TPM_AUTH*	privAuth	Authorization digest for keyHandle and input/returned parameters. HMAC key: Key -> usageAuth.
UINT32	sigSize	The used size of the output area for the signature.
BYTE**	sig	The signature

Request:

SHA (TPM_ORD_Quote, antiReplay, TargetPCR)

HMAC (SHA, authLastNonceEven, nonceOdd, continueAuthSession)

Response:

SHA (returnCode, TPM_ORD_Quote, pcrData, sigSize, sig)

HMAC (SHA, nonceEven, nonceOdd, continueAuthSession)

Comment:

TPM command – TPM_Quote
TPM ordinal – TPM_Quote

4.9.2.3.4 Tcsip_DirWriteAuth

Start of informative comment:

Tcsip_DirWriteAuth provides write access to the Data Integrity Registers.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_DirWriteAuth
(
   TCS_CONTEXT_HANDLE   hContext,     // in
   TCPA_DIRINDEX        dirIndex,     // in
   TCPA_DIRVALUE        newContents,  // in
   TPM_AUTH*            ownerAuth     // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_DirWriteAuth")]
TSS_RESULT Tcsip_DirWriteAuth
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [AUTH, in] TCPA_DIRINDEX          dirIndex,
   [AUTH, in] TCPA_DIRVALUE          newContents,
   [AUTH, in, out] TPM_AUTH*  ownerAuth
);
```

Parameters:

Return Type	Label	Description
TCS_CONTEXT_HANDLE	HContext	Handle of established context.
TCPA_DIRINDEX	DirIndex	Index of the DIR.
TCPA_DIRVALUE	newContents	New value to be stored in the named DIR.
TPM_AUTH*	OwnerAuth	Authorization digest for the inputs and returned parameters. HMAC key: Key -> ownerAuth.

Comment:

TPM command – TPM_DirWriteAuth
TPM ordinal – TPM_DirWriteAuth

4.9.2.3.5 Tcsip_DirRead

Start of informative comment:

Tcsip_DirRead provides read access to the DIRs.

End of informative comment

C-Definition:

```
TSS_RESULT Tcsip_DirRead
(
   TCS_CONTEXT_HANDLE   hContext,     // in
   TCPA_DIRINDEX        dirIndex,     // in
   TCPA_DIRVALUE*       dirValue    // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_DirRead")]
TSS_RESULT Tcsip_DirRead
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TCPA_DIRINDEX      dirIndex,
   [out] TCPA_DIRVALUE*    dirValue
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_DIRINDEX	dirIndex	Index of the DIR to be read.
TCPA_DIRVALUE *	dirValue	Pointer to the current contents of the named DIR.

Comment:

TPM command – TPM_DirRead
TPM ordinal – TPM_DirRead

4.9.2.3.6 Tcsip_Seal

Start of informative comment:

Tcsip_Seal allows software to explicitly state the future "trusted" configuration that the platform must be in for the secret to be revealed. The SEAL operation also implicitly includes the relevant platform configuration (PCR-values) when the SEAL operation was performed. The SEAL operation uses the tpmProof value to BIND the blob to an individual.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Seal
(
   TCS_CONTEXT_HANDLE   hContext,      // in
   TCS_KEY_HANDLE       keyHandle,     // in
   TCPA_ENCAUTH         encAuth,       // in
   UINT32               pcrInfoSize,   // in
   BYTE*                PcrInfo,       // in
   UINT32               inDataSize,    // in
   BYTE*                inData,        // in
   TPM_AUTH*            pubAuth,       // in, out
   UINT32*              SealedDataSize,   // out
   BYTE**               SealedData     // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Seal")]
TSS_RESULT Tcsip_Seal
(
   [in] TCS_CONTEXT_HANDLE    hContext,
   [in] TCS_KEY_HANDLE        keyHandle,
   [AUTH, in] TCPA_ENCAUTH    encAuth,
   [AUTH, in] UINT32          pcrInfoSize,
   [AUTH, in, size_is(pcrInfoSize)] BYTE* PcrInfo,
   [AUTH, in] UINT32          inDataSize,
   [AUTH, in, size_is(inDataSize)] BYTE*  inData,
   [AUTH, in, out] TPM_AUTH*  pubAuth,
   [AUTH, out] UINT32*        SealedDataSize,
   [AUTH, out, size_is(, *SealedDataSize)] BYTE**  SealedData
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	Application key handle of the loaded key.
TCPA_ENCAUTH	encAuth	The encrypted authorization data for the sealed data. The encryption key is the shared secret from the OS-AP protocol.
UINT32	pcrInfoSize	The size of the pcrInfo parameter. If 0 there are no PCR registers in use
BYTE*	PcrInfo	The PCR selection information

Copyright TCG 2003

UINT32	inDataSize	The size of the inData parameter
BYTE*	inData	The data to be sealed to the platform and any specified PCRs
TPM_AUTH*	pubAuth	Authorization digest for the inputs and returned parameters. HMAC key: entity.usageAuth
UINT32*	SealedDataSize	Size of sealed data
BYTE**	SealedData	Encrypted, integrity-protected data object that is the result of the TPM_Seal operation.

Comment:

TPM command – TPM_Seal
TPM ordinal – TPM_Seal

4.9.2.3.7 Tcsip_Unseal

Start of informative comment:

Tcsip_Unseal will reveal TPM_Sealed data only if it was encrypted on this platform and the current configuration (as defined by the named PCR contents) is the one named as qualified to decrypt it. Internally, Tcsip_Unseal accepts a data blob generated by a Tcsip_Seal operation. Tcsip_Unseal decrypts the structure internally, checks the integrity of the resulting data, *and* checks that the PCR named has the value named during Tcsip_Seal. Additionally, the caller must supply appropriate authorization data for blob and for the key that was used to seal that data.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Unseal
(
   TCS_CONTEXT_HANDLE   hContext,          // in
   TCS_KEY_HANDLE       keyHandle,         // in
   UINT32               SealedDataSize,    // in
   BYTE*                SealedData,        // in
   TPM_AUTH*            keyAuth,           // in
   TPM_AUTH*            dataAuth,          // in, out
   UINT32*              DataSize,          // out
   BYTE**               Data               // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Unseal")]
TSS_RESULT Tcsip_Unseal
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [in] TCS_KEY_HANDLE              parentHandle,
   [AUTH, in] UINT32                SealedDataSize,
   [AUTH, in, size_is(SealedDataSize)] BYTE* SealedData,
   [AUTH, in, out] TPM_AUTH*  parentAuth,
   [AUTH, in, out] TPM_AUTH*  dataAuth,
   [AUTH, out] UINT32*              DataSize,
   [AUTH, out, size_is(, *DataSize)] BYTE**  Data
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	Handle of the key that can decrypt the encData.
UINT32	SealedDataSize	Size of sealed data
BYTE*	SealedData	Encrypted, integrity-protected data object that is the result of the TPM_Seal operation.
TPM_AUTH*	keyAuth	Authorization digest for the key usage and input/ returned parameters. HMAC key: Key. Usaage Auth.
TPM_AUTH*	dataAuth	The encrypted authorization data

Copyright TCG 2003

		for the sealed data. The decryption key is the shared secret from the OS-AP protocol.
UINT32*	DataSize	The size of the Data parameter
BYTE**	Data	The data that was unsealed.

Comment:

TPM command – TPM_Unseal
TPM ordinal – TPM_Unseal

4.9.2.3.8 Tcsip_UnBind

Start of informative comment:

Tcsip_UnBind takes the data blob that is the result of a Tspi_Data_Bind command and decrypts it for export to the User. The caller must authorize the use of the key that will decrypt the incoming blob.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_UnBind
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TCS_KEY_HANDLE       keyHandle,  // in
   UINT32               inDataSize, // in
   BYTE*                inData,     // in
   TPM_AUTH*            privAuth,   // in, out
   UINT32*              outDataSize,  // out
   BYTE**               outData       // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_UnBind")]
TSS_RESULT Tcsip_UnBind
(
   [in] TCS_CONTEXT_HANDLE                hContext,
   [in] TCS_KEY_HANDLE                    keyHandle,
   [AUTH, in] UINT32                      inDataSize,
   [AUTH, in, size_is(inDataSize)] BYTE*  inData,
   [AUTH, in, out] TPM_AUTH*  privAuth,
   [AUTH, out] UINT32*                    outDataSize,
   [AUTH, out, size_is(, *outDataSize)] BYTE**  outData
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	Handle of the key that can decrypt the inData.
UINT32	inDataSize	Size of encrypted data
BYTE*	inData	Encrypted, data object that is the result of the Tcsi_Bind operation.
TPM_AUTH*	privAuth	The authorization digest that authorizes the inputs and use of keyHandle. HMAC key: key.usageAuth.
UINT32*	outDataSize	The length of the returned decrypted data
BYTE**	outData	The resulting decrypted data.

Comment:

TPM command – TPM_UnBind
TPM ordinal – TPM_UnBind

Copyright TCG 2003

4.9.2.3.9 Tcsip_CreateMigrationBlob

Start of informative comment:

Tcsip_CreateMigrationBlob implements the first step in the process of moving a migratable key to a new parent or platform. Execution of this command requires knowledge of the migrationAuth field of the key to be migrated.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CreateMigrationBlob
(
   TCS_CONTEXT_HANDLE      hContext,               // in
   TCS_KEY_HANDLE          parentHandle,           // in
   TSS_MIGRATE_SCHEME      migrationType,          // in
   UINT32                  MigrationKeyAuthSize,   // in
   BYTE*                   MigrationKeyAuth,       // in
   UINT32                  encDataSize,            // in
   BYTE*                   encData,                // in
   TPM_AUTH*               parentAuth,             // in, out
   TPM_AUTH*               entityAuth,             // in, out
   UINT32*                 randomSize,             // out
   BYTE**                  random,                 // out
   UINT32*                 outDataSize,            // out
   BYTE**                  outData                 // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CreateMigrationBlob")]
TSS_RESULT Tcsip_CreateMigrationBlob
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [in] TCS_KEY_HANDLE              parentHandle,
   [AUTH, in] TSS_MIGRATE_SCHEME migrationType,
   [AUTH, in] UINT32                MigrationKeyAuthSize,
   [AUTH, in, size_is(MigrationKeyAuthSize)] BYTE* MigrationKeyAuth,
   [AUTH, in] UINT32                encDataSize,
   [AUTH, in, size_is(encDataSize)] BYTE* encData,
   [AUTH, in, out] TPM_AUTH*        parentAuth,
   [AUTH, in, out] TPM_AUTH*        entityAuth,
   [AUTH, out] UINT32*              randomSize,
   [AUTH, out, size_is(, *randomSize)] BYTE**   random,
   [AUTH, out] UINT32*              outDataSize,
   [AUTH, out, size_is(, *outDataSize)] BYTE**  outData
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	parentHandle	Handle of the parent key that can decrypt the encData.
TSS_MIGRATE_SCHEME	migrationType	Migration type, either MIGRATE or REWRAP.
UINT32	MigrationKeyAuth	Size of TCPA_MIGRATONKEYAUTH

Copyright TCG 2003

	Size	byte stream with public key and authorization digest
BYTE*	MigrationKeyAuth	TCPA_MIGRATONKEYAUTH byte stream with public key and authorization digest
UINT32	encDataSize	Size of encData.
BYTE*	encData	Encrypted entity to be modified.
UINT32*	randomDataSize	Used size of the output area for radndomData.
BYTE**	randomData	String used for XOR encryption.
UNIT32*	outDataSize	Used size of the output area for outData.
BYTE**	outData	Modified encrypted entity.
TPM_AUTH*	parentAuth	Authorization digest for the owner and input/ returned parameters. HMAC key: parentKey. Usaage Auth.
TPM_AUTH*	entityAuth	Authorization digest for the owner and nput/ returned parameters. HMAC Key: entityKey.migrationAuth.

Comment:

TPM command – TPM_CreateMigrationBlob
TPM ordinal – TPM_CreateMigrationBlob

4.9.2.3.10 Tcsip_ConvertMigrationBlob

Start of informative comment:

Tcsip_ConvertMigrationBlob takes a migration blob and creates a normal wrapped blob. The migrated blob must be loaded into the TPM using the normal TPM_LoadKey function.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ConvertMigrationBlob
(
   TCS_CONTEXT_HANDLE   hContext,     // in
   TCS_KEY_HANDLE       parentHandle, // in
   UINT32               inDataSize,   // in
   BYTE*                inData,       // in
   UINT32*              randomSize,   // in
   BYTE**               random,       // in
   TPM_AUTH*            parentAuth,   // in, out
   UINT32*              outDataSize,  // out
   BYTE**               outData       // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ConvertMigrationBlob")]
TSS_RESULT Tcsip_ConvertMigrationBlob
(
   [in] TCS_CONTEXT_HANDLE                hContext,
   [in] TCS_KEY_HANDLE                    parentHandle,
   [AUTH, in] UINT32                      inDataSize,
   [AUTH, in, size_is(inDataSize)] BYTE*  inData,
    [AUTH, in] UINT32*                    randomSize,
   [AUTH, in, size_is(randomSize)] BYTE*  random,
   [AUTH, in, out] TPM_AUTH*  parentAuth,
   [AUTH, out] UINT32*                    outDataSize,
   [AUTH, out, size_is(, *outDataSize)] BYTE**  outData
);
```

Parameters*:*

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	parentHandle	Handle of the parent key that can decrypt the encData.
UINT32	inDataSize	Size of inData.
BYTE*	inData	XOR'd and encrypted key.
UINT32	randomDataSize	Size of randomData.
BYTE*	randomData	Random value used to hide the key data.
UNIT32*	outDataSize	Used size of the output area for outData.
BYTE**	outData	The encrypted private key that can be loaded with TPM_LoadKey.
TPM_AUTH*	parentAuth	Authorization digest for the owner and input/returned parameters. HMAC key: parentKey.usageAuth.

Copyright TCG 2003

Comment:

TPM command – TPM_ConvertMigrationBlob
TPM ordinal – TPM_ConvertMigrationBlob

Copyright TCG 2003

4.9.2.3.11 Tcsip_AuthorizeMigrationKey

Start of informative comment:

Tcsip_AuthorizeMigrationKey creates an authorization blob, to allow the TPM owner to specify which migration facility they will use and allow users to migrate information without further involvement with the TPM owner.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_AuthorizeMigrationKey
(
   TCS_CONTEXT_HANDLE   hContext,               // in
   TSS_MIGRATE_SCHEME   migrateScheme,          // in
   UINT32               MigrationKeySize,       // in
   BYTE*                MigrationKey,           // in
   TPM_AUTH*            ownerAuth,              // in, out
   UINT32*              MigrationKeyAuthSize,   // out
   BYTE**               MigrationKeyAuth        // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_AuthorizeMigrationKey")]
TSS_RESULT Tcsip_AuthorizeMigrationKey
(
   [in] TCS_CONTEXT_HANDLE                      hContext,
   [AUTH, in] TSS_MIGRATE_SCHEME                migrateScheme,
   [AUTH, in] UINT32                            MigrationKeySize,
   [AUTH, in, size_is(MigrationKeySize)] BYTE*  MigrationKey,
   [AUTH, in, out] TPM_AUTH*        ownerAuth,
   [AUTH, out] UINT32*                          MigrationKeyAuthSize,
   [AUTH, out, size_is(, *MigrationKeyAuthSize)] BYTE**  MigrationKeyAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_MIGRATE_SCHEME	migrateScheme	Type of migration operation that is to be permitted for this key.
UINT32	MigrationKeySize	Size of TCPA_PUBKEY byte stream with Public key to be authorized
BYTE*	MigrationKey	TCPA_PUBKEY byte stream with Public key to be authorized
TPM_AUTH*	ownerAuth	Authorization digest for the owner and input/returned parameters. HMAC key: ownerAuth.
UINT32*	MigrationKeyAuthSize	Size of TCPA_MIGRATIONKEYAUTH byte stream with public key and authorization digest
BYTE**	MigrationKeyAuth	TCPA_MIGRATIONKEYAUTH byte stream with public key and authorization digest

Comment:

TPM command – TPM_AuthorizeMigrationKey
TPM ordinal – TPM_AuthorizeMigrationKey

Copyright TCG 2003

4.9.2.4 TPM Cryptographic Capabilities

4.9.2.4.1 Tcsip_CertifyKey

Start of informative comment:

Tcsip_CertifyKey allows a key to certify the public portion of certain storage and signing keys.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CertifyKey
(
   TCS_CONTEXT_HANDLE   hContext,      // in
   TCS_KEY_HANDLE       certHandle,    // in
   TCS_KEY_HANDLE       keyHandle,     // in
   TCPA_NONCE           antiReplay,    // in
   TPM_AUTH*            certAuth,      // in, out
   TPM_AUTH*            keyAuth,       // in, out
   UINT32*              CertifyInfoSize,  // out
   BYTE**               CertifyInfo,   // out
   UINT32*              outDataSize,   // out
   BYTE**               outData        // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CertifyKey")]
TSS_RESULT Tcsip_CertifyKey
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [in] TCS_KEY_HANDLE              certHandle,
   [in] TCS_KEY_HANDLE              keyHandle,
   [AUTH, in] TCPA_NONCE             antiReplay,
   [AUTH, in, out] TPM_AUTH*  certAuth,
   [AUTH, in, out] TPM_AUTH*  keyAuth,
   [AUTH, out] UINT32*              CertifyInfoSize,
   [AUTH, out, size_is(, *CertifyInfoSize)] BYTE** CertifyInfo,
   [AUTH, out] UINT32*              outDataSize,
   [AUTH, out, size_is(, *outDataSize)] BYTE**  outData
);
```


Copyright TCG 2003

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	certHandle	Handle of the key to be used to certify the key.
TCS_KEY_HANDLE	keyHandle	Handle of the key to be certified.
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
TPM_AUTH*	certAuth	The authorization handle used for certHandle.
TPM_AUTH*	keyAuth	The authorization handle used for the key to be signed.
UINT32*	CertifyInfoSize	Size of the CertifyInfo
BYTE**	CertifyInfo	The certifyInfo structure that corresponds to the signed key.
UINT32*	outDataSize	The used size of the output area for outData
BYTE**	outData	The signed public key.

Comment:

TPM command – TPM_CertifyKey
TPM ordinal – TPM_CertifyKey

4.9.2.4.2 Tcsip_Sign

Start of informative comment:

Tcsip_Sign signs a digest and returns the resulting digital signature. This command uses a properly authorized signature key.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Sign
(
   TCS_CONTEXT_HANDLE   hContext,       // in
   TCS_KEY_HANDLE       keyHandle,      // in
   UINT32               areaToSignSize,   // in
   BYTE*                areaToSign,     // in
   TPM_AUTH*            privAuth,       // in, out
   UINT32*              sigSize,        // out
   BYTE**               sig             // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Sign")]
TSS_RESULT Tcsip_Sign
(
   [in] TCS_CONTEXT_HANDLE                    hContext,
   [in] TCS_KEY_HANDLE                        keyHandle,
   [AUTH, in] UINT32                          areaToSignSize,
   [AUTH, in size_is(areaToSignSize] BYTE*    areaToSign,
   [AUTH, in, out] TPM_AUTH*  privAuth,
   [AUTH, out] UINT32*                        sigSize,
   [AUTH, out, size_is(, *sigSize)] BYTE**    sig
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can perform digital signatures.
UINT32	areaToSignSize	The size of the areaToSign parameter
BYTE*	areaToSign	The value to sign
TPM_AUTH*	privAuth	The authorization digest that authorizes the use of keyHandle. HMAC key: key.usageAuth
UINT32*	sigSize	The length of the returned digital signature
BYTE**	sig	The resulting digital signature.

Comment:

TPM command – TPM__Sign
TPM ordinal – TPM__Sign

4.9.2.4.3 Tcsip_GetRandom

Start of informative comment:

Tcsip_GetRandom returns the next bytesRequested bytes from the random number generator to the caller.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_GetRandom
(
   TCS_CONTEXT_HANDLE   hContext,      // in
   UINT32*              bytesRequested,   // in, out
   BYTE**               randomBytes    // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_GetRandom")]
TSS_RESULT Tcsip_GetRandom
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in, out] UINT32*       bytesRequested,
   [out, size_is(, *bytesRequested)] BYTE**  randomBytes
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32*	bytesRequested	Number of bytes to return
BYTE**	randomBytes	The returned bytes

Comment:

TPM command – TPM_GetRandom
TPM ordinal – TPM_GetRandom

4.9.2.4.4 Tcsip_StirRandom

Start of informative comment:

Tcsip_StirRandom adds entropy to the RNG state.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_StirRandom
(
   TCS_CONTEXT_HANDLE   hContext,// in
   UINT32            inDataSize, // in
   BYTE*             inData      // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_StirRandom")]
TSS_RESULT Tcsip_StirRandom
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] UINT32             inDataSize,
   [in, size_is(*inDataSize)] BYTE* inData
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32	inDataSize	Number of bytes of input (<256)
BYTE*	inData	Data to add entropy to RNG state

Comment:

TPM command – TPM_StirRandom
TPM ordinal – TPM_StirRandom

Copyright TCG 2003

4.9.2.4.5 Tcsip_GetCapability

Start of informative comment:

Tcsip_GetCapability allows the TPM to report back to the requestor what type of TPM it is dealing with.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_GetCapability
(
   TCS_CONTEXT_HANDLE      hContext,  // in
   TCPA_CAPABILITY_AREA    capArea,   // in
   UINT32                  subCapSize, // in
   BYTE*                   subCap,    // in
   UINT32*                 respSize,  // out
   BYTE**                  resp       // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_GetCapability")]
TSS_RESULT Tcsip_GetCapability
(
   [in] TCS_CONTEXT_HANDLE           hContext,
   [in] TCPA_CAPBILITY_AREA          capArea,
   [in] UINT32                       subCapSize,
   [in, size_is(subCapSize)] BYTE*   subCap,
   [out] UINT32*                     respSize,
   [out, size_is(*respSize)] BYTE**  resp
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_CAPABILITY_AREA	capArea	Partition of capabilities to be interrogated
UINT32	subCapSize	Size of subCap parameter
BYTE*	subCap	Further definition of information
UINT32*	respSize	The length of the returned capability response
BYTE**	resp	The capability response

Comment:

TPM command – TPM_GetCapability
TPM ordinal – TPM_GetCapability

Information about capArea and rgbSubCap is transmitted to the TPM without any interpretation by TCS. The TPM will return an appropriate error on wrong values.

4.9.2.4.6 Tcsip_GetCapabilitySigned

Start of informative comment:

NOTE: The TPM function TPM_GetCapabilitySigned that actually performs this functions was found to contain a vulnerability that makes its security questionable therefore its use unadvised. Since the final TPM specification contained this function and products have shipped with this function it is exposed at the TPM layer. However, the TSS Working Group has decided that TSS should not require the implementation of this function for any TSS. However, if a TSS provider should decided to include this function the TSS WG recommends the implementation contained here.

Tcsip_GetCapabilitySigned is almost the same as Tcsip_GetCapability. The differences are that the input includes a challenge (a nonce) and the response includes a digital signature to vouch for the source of the answer.

C-Definition:

```
TSS_RESULT Tcsip_GetCapabilitySigned
(
    TCS_CONTEXT_HANDLE      hContext,  // in
    TCS_KEY_HANDLE          keyHandle, // in
    TCPA_NONCE              antiReplay, // in
    TCPA_CAPABILITY_AREA    capArea,   // in
    UINT32                  subCapSize, // in
    BYTE*                   subCap,    // in
    TPM_AUTH*               privAuth,  // in, out
    TCPA_VERSION*           Version,   // out
    UINT32*                 respSize,  // out
    BYTE**                  resp,      // out
    UINT32*                 sigSize,   // out
    BYTE**                  sig        // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_GetCapabilitySigned")]
TSS_RESULT Tcsip_GetCapabilitySigned
(
    [in] TCS_CONTEXT_HANDLE          hContext,
    [in] TCS_KEY_HANDLE              keyHandle,
    [AUTH, in] TCPA_NONCE              antiReplay,
    [AUTH, in] TCPA_CAPBILITY_AREA     capArea,
    [AUTH, in] UINT32                subCapSize,
    [AUTH, in, size_is(subCapSize)] BYTE*  subCap,
    [AUTH, in, out] TPM_AUTH*        privAuth,
    [AUTH, out] TCPA_VERSION*        Version,
    [AUTH, out] UINT32*              respSize,
    [AUTH, out, size_is(, *respSize)] BYTE**  resp,
    [AUTH, out] UINT32*              sigSize,
    [AUTH, out, size_is(, *sigSize)] BYTE**   sig,
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	The handle of a loaded key that can perform digital signatures.
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
TCPA_CAPABILITY_AREA	capArea	Partition of capabilities to be interrogated
UINT32	subCapSize	Size of subCap parameter
BYTE*	subCap	Further definition of information
TPM_AUTH*	privAuth	The authorization digest that authorizes the use of keyHandle. HMAC key: key.usageAuth
TCPA_VERSION*	Version	A properly filled out version structure.
UINT32*	respSize	The length of the returned capability response
BYTE**	resp	The capability response
UINT32*	sigSize	The length of the returned digital signature
BYTE**	sig	The resulting digital signature.

Comment:

TPM command – TPM_GetCapabilitySigned
TPM ordinal – TPM_GetCapabilitySigned

Information about capArea and rgbSubCap is transmitted to the TPM without any interpretation by TCS. The TPM will return an appropriate error on wrong values.

End of informative comment.

4.9.2.4.7 Tcsip_GetCapabilityOwner

Start of informative comment:

Tcsip_GetCapabilityOwner enables the TPM Owner to retrieve information belonging to the TPM Owner.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_GetCapabilityOwner
(
   TCS_CONTEXT_HANDLE      hContext,            // in
   TPM_AUTH*               pOwnerAuth,          // in out
   TCPA_VERSION*           pVersion,            // out
   UINT32*                 pNonVolatileFlags,   // out
   UINT32*                 pVolatileFlags       // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_GetCapabilityOwner")]
TSS_RESULT Tcsip_GetCapabilityOwner
(
   [in] TCS_CONTEXT_HANDLE       hContext,
   [in, out, ptr] TPM_AUTH* pOwnerAuth,
   [out] TCPA_VERSION*           pVersion,
   [out] UINT32*                 pNonVolatileFlags,
   [out] UINT32*                 pVolatileFlags
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_AUTH*	pOwnerAuth	Owner authorization
TCPA_VERSION*	pVersion	A properly filled out version structure.
UINT32*	pNonVolatileFlags	The current state of the non-volatile flags.
UINT32*	pVolatileFlags	The current state of the volatile flags.

Comment:

TPM command – TPM_GetCapabilityOwner
TPM ordinal – TPM_GetCapabilityOwner

Information about capArea and rgbSubCap is transmitted to the TPM without any interpretation by TCS. The TPM will return an appropriate error on wrong values.

4.9.2.5 TPM Endorsement Credentials

4.9.2.5.1 Tcsip_CreateEndorsementKeyPair

Start of informative comment:

Tcsip_CreateEndorsementKeyPair generates the endorsement key pair.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CreateEndorsementKeyPair
(
   TCS_CONTEXT_HANDLE   hContext,               // in
   TCPA_NONCE           antiReplay,             // in
   UINT32               endorsementKeyInfoSize, // in
   BYTE*                endorsementKeyInfo,     // in
   UINT32*              endorsementKeySize,     // out
   BYTE**               endorsementKey,         // out
   TCPA_DIGEST*         checksum                // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CreateEndorsementKeyPair")]
TSS_RESULT Tcsip_CreateEndorsementKeyPair
(
   [in]TCS_CONTEXT_HANDLE  hContext,               // in
   [in]TCPA_NONCE          antiReplay,             // in
   [in]UINT32              endorsementKeyInfoSize, // in
   [in, size_is( endorsementKeyInfoSize )]BYTE*
                           endorsementKeyInfo,     // in
   [out]UINT32*            endorsementKeySize,     // out
   [out, size_is( *endorsementKeySize )]BYTE**
                           endorsementKey,         // out
   [out]TCPA_DIGEST*       checksum                // out
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
UINT32	endorsementKeyInfoSize	Endorsement key info size
BYTE*	endorsementKeyInfo	Endorsement key info
UINT32*	endorsementKeySize	Size of the endorsement key
BYTE**	endorsementKey	The public endorsement key
TCPA_DIGEST*	Checksum	Hash of pubEndorsementKey and antiReplay

Comment:

TPM command – TPM_CreateEndorsementKeyPair
TPM ordinal – TPM_CreateEndorsementKeyPair

Copyright TCG 2003

4.9.2.5.2 Tcsip_ReadPubek

Start of informative comment:

Tcsip_ReadPubek returns the public portion of the endorsement key.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ReadPubek
(
   TCS_CONTEXT_HANDLE   hContext,            // in
   TCPA_NONCE           antiReplay,          // in
   UINT32*              pubEndorsementKeySize,  // out
   BYTE**               pubEndorsementKey,      // out
   TCPA_DIGEST*         checksum             // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ReadPubek")]
TSS_RESULT Tcsip_ReadPubek
(
   [in] TCS_CONTEXT_HANDLE    hContext,
   [in] TCPA_NONCE            antiReplay,
   [out] UINT32*              pubEndorsementKeySize,
   [out, size_is(, *pubEndorsementKeySize)] BYTE** pubEndorsementKey,
   [out] TCPA_DIGEST*         checksum
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
UINT32*	pubEndorsementKeySize	Size of puEndorsementKey
BYTE**	pubEndorsementKey	The public endorsement key
TCPA_DIGEST*	checksum	Hash of pubEndorsementKey and antiReplay

Comment:

TPM command – TPM_ReadPubek
TPM ordinal – TPM_ReadPubek

Copyright TCG 2003

4.9.2.5.3 Tcsip_DisablePubekRead

Start of informative comment:

Tcsip_DisablePubekRead allows the TPM owner to prevent any entity from reading the public portion of the endorsement key.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_DisablePubekRead
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TPM_AUTH*            ownerAuth   // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_DisablePubekRead")]
TSS_RESULT Tcsip_DisablePubekReak
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [AUTH, in, out] TPM_AUTH*  ownerAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_AUTH*	ownerAuth	Owner's authorization

Comment:

TPM command – TPM_DisablePubekRead
TPM ordinal – TPM_DisablePubekRead

4.9.2.5.4 Tcsip_OwnerReadPubek

Start of informative comment:

Tcsip_OwnerReadPubek allows the TPM owner to read the public endorsement key.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_OwnerReadPubek
(
   TCS_CONTEXT_HANDLE   hContext,            // in
   TPM_AUTH*            ownerAuth,           // in, out
   UINT32*              pubEndorsementKeySize,  // out
   BYTE**               pubEndorsementKey       // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_OwnerReadPubek")]
TSS_RESULT Tcsip_OwnerReadPubek
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [AUTH, in, out] TPM_AUTH*  ownerAuth,
   [AUTH, out] UINT32*              pubEndorsementKeySize,
   [AUTH, out, size_is(, *pubEndorsementKeySize)] BYTE** pubEndorsementKey
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_AUTH*	ownerAuth	Owner's authorization
UINT32*	pubEndorsement KeySize	Size of puEndorsementKey
BYTE**	pubEndorsement Key	The public endorsement key

Comment:

TPM command – TPM_OwnerReadPubek
TPM ordinal – TPM_OwnerReadPubek

4.9.2.6 TPM Self-Test and Management

4.9.2.6.1 Tcsip_SelfTestFull

Start of informative comment:

Tcsip_SelfTestFull test all of the TPM protected capabilities.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_SelfTestFull
(
   TCS_CONTEXT_HANDLE   hContext // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_SelfTestFull")]
TSS_RESULT Tcsip_SelfTestFull
(
   [in] TCS_CONTEXT_HANDLE hContext
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.

Comment:

TPM command – TPM_OwnerReadPubek
TPM ordinal – TPM_OwnerReadPubek

4.9.2.6.2 Tcsip_CertifySelfTest

Start of informative comment:

Tcsip_CertifySelfTest performs a full TPM self-test and returns an authenticated value if the test passes.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CertifySelfTest
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TCS_KEY_HANDLE       keyHandle,  // in
   TCPA_NONCE           antiReplay, // in
   TPM_AUTH*            privAuth,   // in, out
   UINT32*              sigSize,    // out
   BYTE**               sig         // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CertifySelfTest")]
TSS_RESULT Tcsip_CertifySelfTest
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [in] TCS_KEY_HANDLE              keyHandle,
   [AUTH, in] TCPA_NONCE             antiReplay,
   [AUTH, in, out] TPM_AUTH*  privAuth,
   [AUTH, out] UINT32*              sigSize,
   [AUTH, out, size_is(, *sigSize)] BYTE**   sig
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can perform digital signatures.
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
TPM_AUTH*	privAuth	The authorization digest that authorizes the inputs and use of keyHandle. HMAC key: key.usageAuth
UINT32*	sigSize	The length of the returned digital signature
BYTE**	sig	The resulting digital signature.

Comment:

TPM command – TPM_CertifySelfTest
TPM ordinal – TPM_CertifySelfTest

4.9.2.6.3 Tcsip_ContinueSelfTest

Start of informative comment:

CotinueSelfTest informs the TPM that it may complete the self test of all TPM functions.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ContinueSelfTest
(
   TCS_CONTEXT_HANDLE   hContext,     // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_ContinueSelfTest")]
TSS_RESULT Tcsip_ContinueSelfTest
(
   [in] TCS_CONTEXT_HANDLE hContext,
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.

Comment:

TPM command – TPM_ContinueSelfTest
TPM ordinal – TPM_ContinueSelfTest

4.9.2.6.4 Tcsip_GetTestResult

Start of informative comment:

Tcsip_GetTestResult provides manufacturer specific information regarding the results of the self-test. This command will work when the TPM is in self-test failure mode.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_GetTestResult
(
   TCS_CONTEXT_HANDLE   hContext,     // in
   UINT32*              outDataSize,  // out
   BYTE**               outData,      // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_GetTestResult")]
TSS_RESULT Tcsip_GetTestResult
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [AUTH, out] UINT32*     outDataSize,
   [AUTH, out, size_is(, *outDataSize)] BYTE**  outData
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32*	outDataSize	The size of the outData area
BYTE**	outData	The outData this is manufacturer specific

Comment:

TPM command – TPM_GetTestResult
TPM ordinal – TPM_GetTestResult

4.9.2.6.5 Tcsip_OwnerSetDisable

Start of informative comment:

Tcsip_OwnerSetDisable is used to change the status of the TCPA_PERSISTENT_DISABLE flag.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_OwnerSetDisable
(
   TCS_CONTEXT_HANDLE   hContext,      // in
   TSS_BOOL             disableState,  // in
   TPM_AUTH*            ownerAuth      // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_OwnerSetDisable")]
TSS_RESULT Tcsip_OwnerSetDisable
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [AUTH, in] TSS_BOOL              disableState,
   [AUTH, in, out] TPM_AUTH*  ownerAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_BOOL	disableState	Value for disable state – enable if TRUE
TPM_AUTH*	ownerAuth	Owner authorization

Comment:

TPM command – TPM_OwnerSetDisable
TPM ordinal – TPM_OwnerSetDisable

4.9.2.6.6 Tcsip_OwnerClear

Start of informative comment:

Tcsip_OwnerClear performs the clear operation under TPM owner authorization.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_OwnerClear
(
   TCS_CONTEXT_HANDLE   hContext,      // in
   TPM_AUTH*            ownerAuth      // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_OwnerClear")]
TSS_RESULT Tcsip_OwnerClear
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [AUTH, in, out] TPM_AUTH*  ownerAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_AUTH*	ownerAuth	Owner authorization

Comment:

TPM command – TPM_OwnerClear
TPM ordinal – TPM_OwnerClear

4.9.2.6.7 Tcsip_DisableOwnerClear

Start of informative comment:

Tcsip_DisableOwnerClear disables the ability to execute the TPM_OwnerClear command permanently.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_DisableOwnerClear
(
   TCS_CONTEXT_HANDLE      hContext,  // in
   TPM_AUTH*            ownerAuth   // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_DisableOwnerClear")]
TSS_RESULT Tcsip_DisableOwnerClear
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [AUTH, in, out] TPM_AUTH*  ownerAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_AUTH*	ownerAuth	Owner authorization

Comment:

TPM command – TPM_DisableOwnerClear
TPM ordinal – TPM_DisableOwnerClear

4.9.2.6.8 Tcsip_ForceClear

Start of informative comment:

Tcsip_ForceClear performs the Clear operation under physical access.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ForceClear
(
   TCS_CONTEXT_HANDLE   hContext // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_ForceClear")]
TSS_RESULT Tcsip_ForceClear
(
   [in] TCS_CONTEXT_HANDLE hContext
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.

Comment:

TPM command – TPM_ForceClear
TPM ordinal – TPM_ForceClear

4.9.2.6.9 Tcsip_DisableForceClear

Start of informative comment:

Tcsip_DisableForceClear disables the execution of the ForceClear command until the next startup cycle.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_DisableForceClear
(
   TCS_CONTEXT_HANDLE   hContext // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_DisableForceClear")]
TSS_RESULT Tcsip_DisableForceClear
(
   [in] TCS_CONTEXT_HANDLE hContext
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.

Comment:

TPM command – TPM_DisableForceClear
TPM ordinal – TPM_DisableForceClear

4.9.2.6.10 Tcsip_PhysicalDisable

Start of informative comment:

Tcsip_PhysicalDisable disables TPM physical presence.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_PhysicalDisable
(
   TCS_CONTEXT_HANDLE   hContext // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_PhysicalDisable")]
TSS_RESULT Tcsip_PhysicalDisable
(
   [in] TCS_CONTEXT_HANDLE hContext
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.

Comment:

TPM command – TPM_PhysicalDisable
TPM ordinal – TPM_PhysicalDisable

4.9.2.6.11 Tcsip_PhysicalEnable

Start of informative comment:

Tcsip_PhysicaEnable enables TPM physical presence.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_PhysicalEnable
(
   TCS_CONTEXT_HANDLE   hContext // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_PhysicalEnable")]
TSS_RESULT Tcsip_PhysicalEnable
(
   [in] TCS_CONTEXT_HANDLE hContext
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.

Comment:

TPM command – TPM_PhysicalEnable
TPM ordinal – TPM_PhysicalEnable

4.9.2.6.12 Tcsip_PhysicalSetDeactivated

Start of informative comment:

Sets the TCPA_PERSISTENT_FLAGS.deactivated flag to the value in the state parameter

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_PhysicalSetDeactivated
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TSS_BOOL             state       // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_PhysicalEnable")]
TSS_RESULT Tcsip_PhysicalEnable
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TSS_BOOL           state
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_BOOL	state	State to which deactivated flag is to be set.

Comment:

TPM command – TPM_PhysicalSetDeactivated
TPM ordinal – TPM_PhysicalSetDeactivated

4.9.2.6.13 Tcsip_SetTempDeactivated

Start of informative comment:

Sets the flag TCPA_VOLATILE_FLAGS.deactivated to the value TRUE which temporality deactivate the TPM.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_SetTempDeactivated
(
   TCS_CONTEXT_HANDLE   hContcxt,   // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_SetTempDeactivated")]
TSS_RESULT Tcsip_SetTempDeactivated
(
   [in] TCS_CONTEXT_HANDLE hContext,
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.

Comment:

TPM command – TPM_SetTempDeactivated
TPM ordinal – TPM_SetTempDeactivated

4.9.2.6.14 Tcsip_PhysicalPresence

Start of informative comment:

This method sets the physical presence flags.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_PhysicalPresence
(
   TCS_CONTEXT_HANDLE      hContext,         // in
   TCPA_PHYSICAL_PRESENCE  fPhysicalPresence // in
);
```

IDL Definition:

```
 [helpstring("method Tcsip_ PhysicalPresence")]
TSS_RESULT Tcsip_PhysicalPresence
(
   [in] TCS_CONTEXT_HANDLE       hContext,
   [in] TCPA_PHYSICAL_PRESENCE   fPhysicalPresence
);
```

Parameters

fPhysicalPresence

Value of the physical presence flag.

Return Values

TSS_SUCCESS
TSS_E_NOTIMPL

Remarks

The TSC_PhysicalPresence command is only available on platforms that provide the command method for indicating physical presence of the operator. This is determined by the nature and design of the platform. Further, execution of this command, if implemented, requires the platform be in a predetermined state. This state is usually, but not required, to be pre-OS. Because of these restrictions, this command will likely not be available and will return TSS_E_NOTIMPL. It is included here for the benefit of platforms will execute the TSS in a “restricted” environment.

Copyright TCG 2003

4.9.2.6.15 Tcsip_FieldUpgrade

Start of informative comment:

The TPM needs a mechanism to allow for updating the protected capabilities once a TPM is in the field. Given the varied nature of TPM implementations there will be numerous methods of performing an upgrade of the protected capabilities. This command, when implemented, provides a manufacturer specific method of performing the upgrade.

The manufacturer can determine, within the listed requirements, how to implement this command. The command may be more than one command and actually a series of commands.

The IDL definition is to create an ordinal for the command, however the remaining parameters are manufacturer specific.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_FieldUpgrade
(
   TCS_CONTEXT_HANDLE   hContext,     // in
   UINT32               dataInSize,   // in
   BYTE*                dataIn,       // in
   TPM_AUTH*            ownerAuth     // in, out
   UINT32*              dataOutSize,  // out
   BYTE**               dataOut,      // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_FieldUpgrade")]
TSS_RESULT Tcsip_FieldUpgrade
(
   [in]TCS_CONTEXT_HANDLE  hContext,      // in
   [in]UINT32              dataInSize,    // in
   [in]BYTE*               dataIn,        // in
   [in, out]TPM_AUTH*      ownerAuth,     // in, out
   [out]UINT32*            dataOutSize,   // out
   [out, size_is(, *dataOutSize )]BYTE**
                           dataOut,       // out
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32	dataInSize	Size of field upgrade input data
BYTE*	dataIn	Field upgrade input data
TPM_AUTH*	ownerAuth	Owner authorization
UINT32*	dataOutSize	Size of field upgrade output data
BYTE**	dataOut	Field upgrade output data

Comment:

TPM command – TPM_FieldUpgrade
TPM ordinal – TPM_FieldUpgrade

4.9.2.6.16 Tcsip_SetRedirection

Start of informative comment:

'Redirected" keys enable the output of a TPM to be directed to non-TCG security functions in the platform, without exposing that output to non-security functions.

It is sometimes desirable to direct the TPM's output directly to specific platform functions without exposing that output to other platform functions. To enable this, the key in a leaf node of TCG Protected Storage can be tagged as a "redirect" key. Any plaintext output data secured by a redirected key is passed by the TPM directly to specific platform functions and is not interpreted by the TPM.

Since redirection can only affect leaf keys, redirection applies to: TPM_Unbind, TPM_Unseal, TPM_Quote, TPM_Sign.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_SetRedirection
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TCS_KEY_HANDLE       keyHandle,  // in
   UINT32               c1,         // in
   UINT32               c2,         // in
   TPM_AUTH*            privAuth    // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_SetRedirection")]
TSS_RESULT Tcsip_SetRedirection
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [in] TCS_KEY_HANDLE              keyHandle,
   [AUTH, in] UINT32                c1,
   [AUTH, in] UINT32                c2,
   [AUTH, in, out] TPM_AUTH*  ownerAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can implement redirection.
UINT32	c1	Manufacturer parameter
UINT32	c2	Manufacturer parameter
TPM_AUTH*	privAuth	The authorization handle used for keyHandle authorization

Comment:

TPM command – TPM_SetRedirection
TPM ordinal – TPM_SetRedirection

4.9.2.7 TPM Optional

4.9.2.7.1 Tcsip_CreateMaintenanceArchive

Start of informative comment:

Tcsip_CreateMaintenanceArchive creates a TPM maintenance archive.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CreateMaintenanceArchive
(
   TCS_CONTEXT_HANDLE   hContext,      // in
   TSS_BOOL             generateRandom,   // in
   TPM_AUTH*            ownerAuth,     // in, out
   UINT32*              randomSize,    // out
   BYTE**               random,        // out
   UINT32               archiveSize,   // out
   BYTE**               archive        // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CreateMaintenanceArchive")]
TSS_RESULT Tcsip_CreateMaintenanceArchive
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [AUTH, in] TSS_BOOL              generateRandom,
   [AUTH, in, out] TPM_AUTH*  ownerAuth,
   [AUTH, out] UINT32*              randomSize,
   [AUTH, out, size_is(, *randomSize)] BYTE**   random,
   [AUTH, out] UINT32*              archiveSize,
   [AUTH, out, size_is (, *archiveSize)] BYTE** archive
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_BOOL	generateRandom	Use RNG or Owner auth to generate 'random'.
TPM_AUTH*	ownerAuth	The authorization handle used for owner authorization.
UINT32*	randomSize	Size of the returned random data. Will be 0 if generateRandom is FALSE.
BYTE**	random	Random data to XOR with result.
UINT32	archiveSize	Size of the encrypted archive
BYTE**	archive	Encrypted key archive.

Comment:

TPM command – TPM_CreateMaintenanceArchive
TPM ordinal – TPM_CreateMaintenanceArchive

4.9.2.7.2 Tcsip_LoadMaintenanceArchive

Start of informative comment:

Tcsip_LoadMaintenanceArchive loads a TPM maintenance archive that has been massaged by the manufacturer to load into another TPM.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_LoadMaintenanceArchive
(
   TCS_CONTEXT_HANDLE   hContext,     // in
   UINT32               dataInSize,   // in
   BYTE*                dataIn,       // in
   TPM_AUTH*            ownerAuth,    // in, out
   UINT32*              dataOutSize,  // out
   BYTE**               dataOut       // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_LoadMaintenanceArchive")]
TSS_RESULT Tcsip_LoadMaintenanceArchive
(
   [in]TCS_CONTEXT_HANDLE  hContext,      // in
   [in]UINT32              dataInSize,    // in
   [in]BYTE*               dataIn,        // in
   [in]TPM_AUTH*           ownerAuth,     // in, out
   [out]UINT32*            dataOutSize,   // out
   [out, size_is( *dataOutSize )]BYTE**
                           dataOut        // out
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32	dataInSize	Size of vendor-specific data
BYTE*	dataIn	Vendor specific data
TPM_AUTH*	ownerAuth	Owner authorization
UINT32*	dataOutSize	Size of Vendor specific data
BYTE**	dataOut	Vendor specific data

Comment:

TPM command – TPM_LoadMaintenanceArchive
TPM ordinal – TPM_LoadMaintenanceArchive

Copyright TCG 2003

4.9.2.7.3 Tcsip_KillMaintenanceArchive

Start of informative comment:

Tcsip_KillMaintenanceFeature is a permanent action that prevents ANYONE from creating a TPM maintenance archive until a new TPM owner is set.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_KillMaintenanceFeature
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TPM_AUTH*            ownerAuth   // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_KillMaintenaceFeature")]
TSS_RESULT Tcsip_KillMaintenanceFeature
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [AUTH, in, out] TPM_AUTH*  ownerAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_AUTH*	ownerAuth	Owner authorization

Comment:

TPM command – TPM_KillMaintenanceFeature
TPM ordinal – TPM_KillMaintenanceFeature

4.9.2.7.4 Tcsip_LoadManufaturerlMaintenancePub

Start of informative comment:

Tcsip_LoadManuMainPub loads the TPM manufacture's public key for use in the maintenance process.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_LoadManuMaintPub
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TCPA_NONCE           antiReplay, // in
   UINT32               PubKeySize, // in
   BYTE*                PubKey,     // in
   TCPA_DIGEST*         checksum    // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_LoadManuMaintPub")]
TSS_RESULT Tcsip_LoadManuMaintPub
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TCPA_NONCE         antiReplay,
   [in] UINT32             PubKeySize,
   [in, size_is(PubKeySize)] BYTE*  PubKey,
   [out] TCPA_DIGEST*      checksum
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
UINT32	PubKeySize	Size of the public key
BYTE*	PubKey	The public key of the manufacturer to be in use for maintenance
TCPA_DIGEST*	checksum	Digest of pubKey and antiReplay

Comment:

TPM command – TPM_LoadManuMaintPub
TPM ordinal – TPM_LoadManuMaintPub

4.9.2.7.5 Tcsip_ReadManufacturerMaintenancePub

Start of informative comment:

Tcsip_ReadManuMainPub is used to check whether the manufacturer's public maintenance key in a TPM has the expected value.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ReadManuMaintPub
(
   TCS_CONTEXT_HANDLE   hContext,   // in
   TCPA_NONCE           antiReplay, // in
   TCPA_DIGEST*         checksum    // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ReadManuMaintPub")]
TSS_RESULT Tcsip_ReadManuMaintPub
(
   [in] TCS_CONTEXT_HANDLE hContext,
   [in] TCPA_NONCE         antiReplay,
   [out] TCPA_DIGEST*      checksum
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
TCPA_DIGEST*	checksum	Digest of pubKey and antiReplay

Comment:

TPM command – TPM_ReadManuMaintPub
TPM ordinal – TPM_ReadManuMaintPub

Copyright TCG 2003

5. TCG Device Driver Library (TDDL)

5.1 TDDL Architecture

Start of informative comment:

The intent of this document is to describe an interface between the TCG Software Stack (TSS) and Trusted Platform Module (TPM) in a TCG-enabled Trusted Platform. This interface is called the TPM Device Driver Library Interface (TPM DDLI). The TPM device driver library (TPM DDL) is a module that exists between TSS and the low-level TPM device driver (TPM DD). The TPM DDL is implemented in user-mode and performs processing in the calling application context (i.e. TSS core system service). The TPM DDL is designed to be single-threaded, single-instance, and assumes that TPM command serialization has been performed by the calling application. The TPM DDLI is of a synchronous nature. The TPM vendor is responsible for defining the interface between this library and the actual TPM device. The TPM vendor can choose the communication and resource allocation mechanisms between this library and any kernel-mode TPM driver or software TPM simulator.

In most platform implementations, the TPM DLL is loaded when a TSS application (i.e. TCS) initializes. On most platforms, this will occur during operating system startup. To guarantee access to the TPM DDL by any TSS application module, a strict library naming convention must be followed for each operating system implementation.

End of informative comment.

5.2 Memory Management

Start of informative comment:

The “classical” memory allocation approach is used, by the TPM DDL, where the calling application allocates memory for the in and out parameters associated with each interface call. Symmetrically, the calling application is responsible for de-allocating the memory associated with any call to the TPM DDL.

Retrieving the required parameter size from the callee accepting to call the callee twice or always checking for an error return code “TPMDDL_INSUFFICIENT_BUFFER” is not supported since not every TPM command can be repeated getting the same results (e.g. TPM_Extend).

In the TPM DDLI described in this document, parameters are documented as follows:

in parameters with the comment “// in”

out parameters with the comment “// out”

End of informative comment.

5.3 TDDL Error Code Defines

With the following table the error codes common to all TDDL functions are listed. In addition to these error codes, the TSS_E_* error codes out of the range of common errors may also be returned with the layer set to the value for the TDDL.

In addition each Tddli function will list in its description the error return codes specific to the function.

Type	Definition
TDDL_SUCCESS	Successful completion of the operation.
TDDL_E_FAIL	The operation failed.
TDDL_E_BAD_PARAMETER	Same as TSS_E_BAD_PARAMETER
TDDL_E_OUTOFMEMORY	Same as TSS_E_OUTOFMEMORY

Copyright TCG 2003

TDDL_E_COMPONENT_NOT_FOUND	TPM device driver is not running
TDDL_E_ALREADY_OPENED	TPM device driver opened.
TDDL_E_BADTAG	The capability or sub capability code is not correct or not supported.
TDDL_E_TIMEOUT	The operation has timed out.
TDDL_E_INSUFFICIENT_BUFFER	The receive buffer is too small.
TDDL_E_COMMAND_COMPLETED	The command has already completed.
TDDL_E_ALREADY_CLOSED	TPM device driver closed.
TDDL_E_IOERROR	An IO error occurred transmittinginformation to the TPM.
TDLL_E_COMMAND_ABORTED	TPM aborted processing of command.

5.4 TDDL-specific Return code Rules

Only return codes specified within each function MAY be returned for each function.

5.5 TDDL Interface

5.5.1 Tddli_Open

Start of informative comment:

This function establishes a connection with the TPM device driver. Following a successful response to this function, the TPM device driver must be prepared to process TPM command requests from the calling application. The application utilizing the TPM DDL is guaranteed to have exclusive access to the TPM device. If this call fails, it may be an indication that the TPM device driver is not loaded, started, or the TPM cannot support any protected requests.

This function must be called before calling Tddli_GetStatus, Tddli_GetCapability, Tddli_SetCapability, or Tddli_TransmitData.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_Open( );
```

Parameters:

None.

Return Value:

```
TDDL_SUCCESS
TDDL_E_COMPONENT_NOT_FOUND
TDDL_E_ALREADY_OPENED
TDDL_E_FAIL
```

5.5.2 Tddli_Close

Start of informative comment:

This function closes a connection with the TPM device driver. Following a successful response to this function, the TPM device driver can clean up any resources used to maintain a connection with the TPM device driver library. If this call fails, it may provide an indication that the TPM device driver cannot clean up or may need to be restarted or reloaded.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_Close( );
```

Parameters:

None.

Return Value:

TDDL_SUCCESS
TDDL_E_ALREADY_CLOSED
TDDL_E_FAIL

5.5.3 Tddli_Cancel

Start of informative comment:

This function cancels an outstanding TPM command. An application can call this function, in a separate context, to interrupt a TPM command that has not completed. The previous TPM command must be the result of a call to the Tddli_TransmitData function. The TPM device driver must acknowledge this function if it has not returned from a previous TPM command and return TDDL_COMMAND_ABORTED for the call in process.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_Cancel( );
```

Parameters:

None.

Return Value:

TDDL_SUCCESS
TDDL_COMMAND_COMPLETED
TDDL_E_FAIL

5.5.4 Tddli_GetCapability

Start of informative comment:

This function queries the TPM hardware, firmware and device driver attributes such as firmware version, driver version, etc.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_GetCapability
(
   UINT32   CapArea,    // in
   UINT32   SubCap,     // in
   BYTE*    pCapBuf,    // out
   UINT32*  pCapBufLen  // in, out
);
```

Parameters:

Type	Name	Description
UINT32	CapArea	Partition of capabilities to be interrogated.
UINT32	SubCap	Subcode of the requested capabilities.
BYTE*	pCapBuf	Pointer to a buffer containing the received attribute data.
UINT32*	pCapBufLen	[in] Size of the receive buffer in bytes [out] Number of written bytes.

Return Values:

TDDL_SUCCESS
TDDL_E_BAD_PARAMETER
TDDL_E_OUTOFMEMORY
TDDL_E_BADTAG
TDDL_E_FAIL

Defined Capability Areas	Defined Capability Sub-Codes	Response
TCPA_CAP_VERSION	TSS_CAP_PROP_DRV	Returns the version of the TPM device driver. The version is coded in the TPM_VERSION format.
TCPA_CAP_VERSION	TSS_CAP_PROP_FW	Returns the version of the current TPM firmware. The version is coded in the TPM_VERSION format.
TCPA_CAP_VERSION	TSS_CAP_PROP_FW_DATE	Returns the release date of the firmware. The date is coded in three bytes mm/dd/yy (mm=month, dd=day, yy=year).
TCPA_CAP_PROPERTY	TCPA_CAP_PROP_MANUFACTURER	Returns the name of the device vendor. The

		returned data is coded in an ASCII string without the trailing null.
TCPA_CAP_PROPERTY	TSS_CAP_PROP_MODULE_TYPE	Returns the vendor specific designation type of the device. The returned data is coded in an ASCII string without the trailing null.
TCPA_CAP_PROPERTY	TSS_CAP_PROP_GLOBAL_STATE	Returns the global state of the module, (e.g. initialized or personalized).
TCPA_CAP_VENDOR	TCPA_CAP_VENDOR_XXX	Returns the vendor specific capabilities of the TPM.

5.5.5 Tddli_SetCapability

Start of informative comment:

This function sets parameters in the TPM hardware, firmware and device driver attributes. An application can set TPM device driver and operating parameters that may be defined by the TPM vendor. For now, the parameter definitions are vendor-defined.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_SetCapability
(
   UINT32   CapArea,       // in
   UINT32   SubCap,        // in
   BYTE*    pSetCapBuf,    // in
   UINT32   SetCapBufLen   // in
);
```

Parameters:

Type	Name	Description
UINT32	CapArea	Partition of capabilities to be set.
UINT32	SubCap	Subcode of the capabilities to be set.
BYTE*	pSetCapBuf	Pointer to a buffer containing the capability data to be sent.
UINT32	SetCapBufLen	[in] Size of the request buffer in bytes.

Return Values:

TDDL_SUCCESS
TDDL_E_OUTOFMEMORY
TDDL_E_BAD_PARAMETER
TDDL_E_BADTAG
TDDL_E_FAIL

5.5.6 Tddli_GetStatus

Start of informative comment:

This function queries the status the TPM driver and device. An application can determine the health of the TPM subsystem by utilizing this function.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_GetStatus
(
   UINT32   ReqStatusType, // in
                                        UINT32* pStatus,       // out
);
```

Parameters:

Type	Name	Description
UINT32	ReqStatusType	Requested type of status information, driver or device.
UINT32*	punStatus	[out] Requested status.

Return Values:

TDDL_SUCCESS
TDDL_E_BAD_PARAMETER
TDDL_E_INSUFFICIENT_BUFFER
TDDL_E_FAIL

Defined Status Type	Defined Response Code-Code	Description
TDDL_DRIVER_STATUS	TDDL _DRIVER_OK	TPM driver is functionaing okay.
TDDL _DRIVER_STATUS	TDDL _DRIVER_FAILED	TPM driver is not functioning.
TDDL _DRIVER_STATUS	TDDL _DRIVER_NOT_OPENED	Device was found, but the corresponding driver could not be opened.
TDDL _DEVICE_STATUS	TDDL _DEVICE_OK	TPM device is functioning okay.
TDDL _DEVICE_STATUS	TDDL _DEVICE_UNRECOVERABLE	TPM device contains an unrecoverable error.
TDDL _DEVICE_STATUS	TDDL _DEVICE_RECOVERABLE	TPM device contains a recoverable error.
TDDL _DEVICE_STATUS	TDDL _DEVICE_NOT_FOUND	TPM device is not found.

5.5.7 Tddli_TransmitData

Start of informative comment:

The function sends a TPM command directly to a TPM device driver, causing the TPM to perform the corresponding operation. This function provides a pass through for the TPM parameter block definitions are defined in the TCPA 1.1b Main Specification.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_TransmitData
(
   BYTE*    pTransmitBuf,     // in
   UINT32   TransmitBufLen,   // in
   BYTE*    pRececeiveBuf,    // out
   UINT32*  pRececeiveBufLen // in, out
);
```

Parameters:

Type	Name	Description
BYTE*	pTransmitBuf	Pointer to a buffer containing TPM transmit data.
UINT32	TransmitBufLen	Size of TPM transmit data in bytes.
BYTE*	pRececeiveBuf	Pointer to a buffer containing TPM receive data
UINT32*	pRececeiveBufLen	[in] Size of TPM receive buffer in bytes [out] Number of written bytes.

Return Value:

TDDL_SUCCESS
TDDL_E_INSUFFICIENT_BUFFER
TDDL_E_IOERROR
TDDL_E_FAIL

Copyright TCG 2003