

# TCG Inside? A Note on TPM Specification Compliance

Ahmad-Reza Sadeghi  
sadeghi@crypto.rub.de

Marcel Selhorst  
tpm@selhorst.net

Christian Stüble  
stueble@acm.org

Marcel Winandy  
winandy@crypto.rub.de

Horst-Görtz-Institute for IT-Security  
Ruhr-University Bochum, Germany

May 3, 2006

## Abstract

The Trusted Computing Group (TCG) has addressed a new generation of computing platforms employing both supplemental hardware and software with the primary goal to improve the security and the trustworthiness of future IT-Systems. The core component of the TCG proposal is the Trusted Platform Module (TPM) providing certain cryptographic functions. Currently, different instantiation of the TPM specification are available as chips placed on the mainboard of the underlying platform.

In practice different manufacturers may implement TPM differently: they may exploit the flexibility that the specification itself provides, or they may deviate from the specification by inappropriate design that might lead to security weaknesses. Many vendors currently equip their platforms and devices with TPM claiming to be TCG compliant. However, there is no feasible way for users of these systems to verify this fact. Further, in the near future many applications may use TPM functionalities and need to rely on their correctness. Hence it is crucial to have an independent means for testing the compliance as well as analyzing the se-

curity of different TPMs.

Due to the above reasons we have developed a prototype test suite for TPM compliance testing. This paper introduces our testing strategy as well as our sample test results and their analysis for different TPM implementations. We show that some TPM implementations do not meet the TCG specification and have bugs.

Moreover, we discuss that non-compliance and inappropriate implementation may have crucial security impact, and point out the corresponding security problems in case of a widespread TPM chip.

## 1 Introduction

The Trusted Computing Group (TCG) [20] aims at developing and supporting open industry specifications for trusted computing across multiple platform types. The TCG is a large organization consisting of industrial (and academic) members including component vendors, software developers, systems vendors, and network and infrastructure companies.

The TCG has released several documents and specifications describing how to add trust to existing computing systems and how to ex-

tend security in certain environments, e.g., in network infrastructures. A core component the TCG specification specifies is the Trusted Platform Module (TPM). The current widespread implementation of the TPM is a small chip capable of securely storing cryptographic keys and other security critical information and providing cryptographic functions like asymmetric encryption and signature schemes (e.g., RSA) or hash functions (SHA-1, HMAC) as well as strong (hardware-based) random numbers (that are important for generating cryptographic keys). Using the TPM functionalities one can attest the initial configuration of the underlying platform and seal and bind data to a specific platform configuration. Further, the digital signature functionality can be used for a public key infrastructure (PKI).

Hence, the TPM acts as root of trust and basis for all other specifications of the TCG. Vendors already deploy computer systems, e.g., laptop computers, that are equipped with a TPM chip placed on the mainboard of the underlying platform. The first TPM chips were delivered in early 2003. Meanwhile, there are several companies producing TPM chips. Most of them have advanced their secure smartcard microcontrollers and use this technology for the TPM. Commonly used TPMs are, e.g., from (in alphabetic order) Atmel [3, 4, 5, 2], Broadcom [11, 8, 9, 10], Infineon [12], National Semiconductor [15], and ST-Microelectronics [19, 16, 17, 18].

The TCG specification of TPM is available and revised in two version, 1.1b [1] and 1.2 [21]. Since these documents have a certain degree of complexity, it is assumable that not all TPM chip models operate exactly as specified. The TCG specifications describe what *must* be implemented, what *may* be implemented and specify the main requirements for a TPM.

In practice different vendors may implement the TPM differently. They may exploit the flexibility that the specification itself provides

or may deviate from the specification by inappropriate design and implementation that might lead to security weaknesses. Many vendors currently equip their platforms and devices with TPM claiming to be TCG compliant. However, there is no feasible way for users of these systems to verify this fact. Moreover, many applications may use the functionalities provided by the TPM in the near future and rely on the claimed functionalities and their correctness.

In this context it is crucial to have means for testing the compliance as well as analyzing the security of different TPMs. Furthermore, this test should be developed by an organization independent from a TPM vendor in order to provide an unbiased test, which will allow the comparison of results of TPM chips from different vendors.

In this paper we introduce the prototype test suite we have developed for TPM compliance tests. Based on our test results we point out the non-compliance and bugs of several TPM implementations. We present our testing strategy, sample test results and analysis for different TPM implementations of different manufacturers. The test environment we use is open source and can be used widely. Moreover, we discuss how one can construct attacks by exploiting certain deviations from the specification and deficiencies of protection mechanisms. We then show this on a widespread TPM chip.

We stress that, although our test suite does not cover the whole specification, we believe that it is representative enough to show the importance of a compliance test. Currently we are developing a complete test suite for TCG specification.

## 2 TPM Preliminaries

In this section we introduce some TPM properties, which are used throughout this paper.

The TCG specification requires every TPM chip to have several basic functions and properties, for example regarding the number of internal registers, cryptographic functions and the strength of cryptographic keys.

The TPM has volatile and non-volatile memory, which store information about the TPM's state and its capabilities (see below).

**Random Number Generator:** Every TPM is equipped with a hardware random number generator in order to deliver secure random numbers, e.g., for key generation or to provide seed for software based random number generators, which are usually faster than the hardware (true) random number generators inside TPMs.

**Cryptographic Functions:** The TCG requires mandatory cryptographic functions, which are usually provided by a cryptographic co-processor inside the TPM. Those functions are asymmetric key generation (RSA), asymmetric encryption / decryption (RSA), hashing (SHA-1) and keyed hashing (HMAC).

**Registers:** The TPM has different types of registers:

1. *Platform Configuration Register (PCR):* A PCR is a 160-bit register for storing *integrity measurements*. The TPM has to provide at least 16 PCRs in TCG 1.1b and 24 PCRs in TCG 1.2 inside the TPM's protected memory. The content of a PCR can only be extended<sup>1</sup> and can only be reset by a reboot.<sup>2</sup>
2. *Data Integrity Register (DIR):* The DIR is a non-volatile register specified in TCG

---

<sup>1</sup>  $PCR_{i,New} = SHA1(PCR_{i,Old} || new\_value)$

<sup>2</sup>This concerns only PCR 0...15. PCR 16...23 (as provided in TCG 1.2) can be reset anytime by the appropriate Locality.

specification 1.1b. TCG 1.2 has specified a dedicated NV-Index to replace the previous DIR. Furthermore, non-volatile storage is directly supported in TCG 1.2.

**Taking Ownership:** Before using a TPM it is necessary to install an owner into the TPM. In this case the TPM stores the authorization information (e.g., password) for the new owner. Furthermore, the TPM creates an asymmetric key pair called *Storage Root Key (SRK)*, which is used to securely store keys (generated by the TPM) outside the TPM. Keys generated directly under the SRK are encrypted using the SRK public key.<sup>3</sup>

**Capabilities:** TPM capabilities represent information about the TPM vendor, the TPM version, the number of registers, the supported algorithms, protocols and commands of the TPM as well as owner information. They are stored in the so-called *capability areas*. They also contain run-time information, such as the number of keys loaded into the TPM's keyslots or their corresponding key handles. Capability information may change during interaction with a TPM. In TCG specification 1.1b, some capabilities are only accessible by performing owner authorization, but this feature has been deleted in specification version 1.2.

**Cryptographic Keys:** The TPM is able to create different types of keys. The TPM can create *subkeys*, which are defined as an asymmetric key pair created within the TPM. The input parameters required to create a key are the key type (e.g., legacy, storage, signing, binding), key size (e.g., 512, 1024, 2048 bits), key attributes (e.g., migratable, non-migratable) and authorization data (e.g., key

---

<sup>3</sup>The key hierarchy might be more deep, so not all keys are encrypted with the SRK public key.

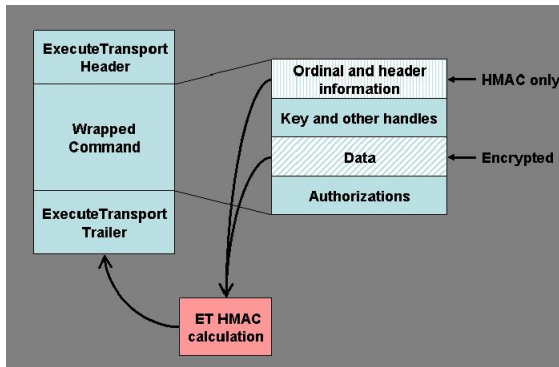


Figure 1: General overview of a TPM command structure [21, part 1, page 96].

password, migration password). A subkey encrypted (under SRK) is stored outside the TPM. In order to use a key, it should first be loaded into the TPM. The TPM internally decrypts the key (under its parent decryption key) and provide a key handle (see Section 2.1.2). The TCG gives no minimum requirements on the time consumption of a key generation for asymmetric keys[21, ch. 4.3].

**Command Structure:** Every TPM command provides one TPM functionality and has an own command ordinal assigned to its function. A command is wrapped by a transport header and a transport trailer. The wrapped data itself contains the command ordinal<sup>4</sup> and any information needed to perform the requested operation (e.g., command size, handles, data, authorization). Figure 1 gives a general overview of the TPM command structure. The structure defines all input parameters, which have to be sent to the TPM, and the corresponding outputs.

**Authorization Protocols:** The TPM has different authorization protocols, which

<sup>4</sup>e.g., the command ordinal for "TPM.CreateWrapKey" is 31

are needed to create a shared session secret between the user and the TPM. This secret is needed for encrypting data during the next TPM commands to be executed. In order to select the corresponding secret, the TPM returns a session handle upon a successful session request, which internally points to the shared secret. After using a session, it has to be terminated. The most common authorization protocols are:

**OIAP:** Object-Independent Authorization Protocol sessions were designed for reasons of efficiency; only one setup process is required for potentially many authorizations. The session can live indefinitely until either party request the session termination.

**OSAP:** The Object-Specific Authorization Protocol allows establishment of an authentication session for a single entity (e.g., the TPM owner). An OSAP session is doubly efficient because only one setup process is required for potentially many authorization calculations and the entity authorization secret is required only once.

## 2.1 Compliance and Security Aspects

### 2.1.1 Compliance

The TCG specification 1.1b does not evaluate compliance directly and suggest that the manufacturer shall perform it [1, ch. 10.15]. This requirement has been replaced by a more generic normative sentence in [21, part 1, ch. 2].

### 2.1.2 Handles

A handle provides pointers to TPM internal resources. The TCG specification 1.2 requires,

that session and key handles should provide the ability to locate a value without collision. For this reason, the three least significant bytes (LSB) of the handle must contain the collision resistance values. One attempt to fulfill this requirement is to have the three LSB of the handle to be generated randomly [1, ch. 4.4].

### 2.1.3 Dictionary Attack

The TCG specification version 1.1 does not suggest any requirements for dictionary attack mitigation. Version 1.2 adds the requirement that the TPM vendor provides some measures against dictionary attacks where the internal mechanism is vendor specific.

The protection mechanism against dictionary attacks – as suggested in the TCG specification 1.2 – is to delay the response of the TPM to any request for a time out period, if a threshold of failed authorization exceeds. This time out period doubles each time the threshold exceeds again. The owner of a TPM is able to resolve (reset) this locked state by performing a “TPM\_ResetLockValue” command. [1, ch. 8.1].

## 2.2 TPM Modes Of Operation

The TPM has several states (enabled/disabled, active/inactive, owned/unowned). Figure 2 illustrates all possible modes of operation. Since the behavior of the TPM varies depending on its current state, testing the TPM for specification compliance has to consider all modes [1, ch. 9.4].

## 3 Compliance Test

In the following we describe our compliance test, its requirements and strategy, and derive goals for a TPM specification compliance. These goals define the requirements of the test implementation.

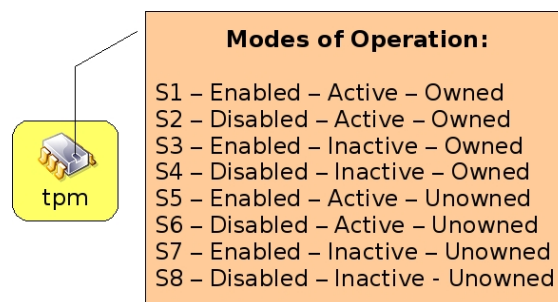


Figure 2: TPM modes of operation.

## 3.1 Test Requirements

**Traceability of Test Results:** The test results must be understandable and reproducible based on the the configuration of hardware and software on which the test have been performed.

**Comparability of Test Results:** The test results of a TPM chip implementations of different vendors must be comparable. Thus, there must be no assumptions on the TPM implementation except those stated in the TPM specification.

**Extensibility of Test Suite:** The test suite must be easily extensible to add additional tests at a later date. Extensibility of the test suite is an important issue, since there may be certain aspects which will probably be identified at a later time. Another reason for having such a requirement is the extension of the framework for testing the compliance to a possible later version of the specification.

**Cost Efficiency:** Performing the compliance test should be cost-efficient. This means that it should not be necessary to have expensive test equipment and complex test procedures which can only be executed by highly trained personnel.

## 3.2 Main Concept

From the perspective of a software application a TPM acts as a black box providing functions that compute an output on given input. Thus, we can interpret a TPM as another software module that has clearly specified interfaces. For testing the specification compliance of the TPM we need to test these interfaces regarding the specified input-output correlation. This enables us to use testing techniques similar to those of software modules [6, 7].

In the following we categorize and identify the required tests, whereas we derive the categorizations from the test discipline of the *Unified Process* [13, 14], and explain how they are used to determine the TPM compliance.

## 3.3 Test Categorization

It is reasonable to structure tests into different categories. Since there are different types of tests which can be performed, we arrange them into groups and map them to respective quality dimensions which are useful to determine the specification compliance of a concrete TPM. In a first step, we analyze the TPM specification and derive individual test cases for different TPM functionalities. In a second step, we categorize each test case into three dimensions of testing: the quality dimension, the level of testing, and the type of test.

### 3.3.1 Quality Dimension

The specification compliance of a TPM implementation can be assessed through different qualities. Below we describe the quality dimensions we use in our test suite.

**Functionality:** This quality assesses whether the TPM executes the desired behavior as intended.

**Reliability:** This quality assesses the TPM's ability to perform consistently and in a deterministic way. Therefore, we test if a TPM implementation is compliant to the specification and we also test if the TPM performs resistantly to failures (robustness).

**Security (implicit):** This quality evaluates the security properties of the TPM, e.g., its resistance against dictionary attacks, etc. However, this quality may only belong partly to the TPM specification compliance, i.e., when explicitly stated in the specification.

**Performance (optional):** This quality assesses the performance of the TPM. Therefore, we use tests that measure the time consumed by each TPM operation. However, this quality does not belong to the TPM specification compliance.

To measure the different qualities we perform different tests for each quality dimension. In the following, we describe the levels of testing and the test types we use.

### 3.3.2 Levels of Testing

We distinguish two levels of testing:

- **Application Level:** High-level tests that test the TPM functionality from the view (and interfaces) of an application.
- **Protocol Level:** Low-level tests that operate on the command data structures used to communicate with the TPM.

We operate on these two levels in order to cover most operational conditions. The high-level tests use the TPM functionality as real applications would do. Thus, this simulates a real-life usage scenario in which the TPM would be deployed and would have to operate appropriately.

The low-level tests give us the opportunity to test specific conditions which probably would arise very rarely in applications deployed in real life environments.

### 3.3.3 Test Types

We distinguish between the following types of test:

**Functional test:** This test checks the intended behavior, i.e., if the outputs for certain given inputs are conform to the functional specification.

**Integrity test:** This test considers the aspects of reliability (if the TPM is reliable when operated in a specified manner) and robustness (if the TPM is even reliable when operated in a not specified manner, e.g., by providing incorrect input parameters).

**Stress test:** Determines the stability of the system under abnormal or extreme condition beyond the common operational capacity. An example is to call a certain command many times one after another.

### 3.4 Test Strategy

Our strategy is to perform one or more types of test on each level per quality dimension. This means, we test a TPM for the functionality dimension by performing functional tests on both the application level and the protocol level. For testing the reliability dimension we perform integrity and stress tests on both the application level and the protocol level. Figure 3 shows the mapping of the test types to the test level and quality dimensions. Having defined the mapping this way gives us the ability to state the compliance to the TPM specification from a multiple point of view but in a

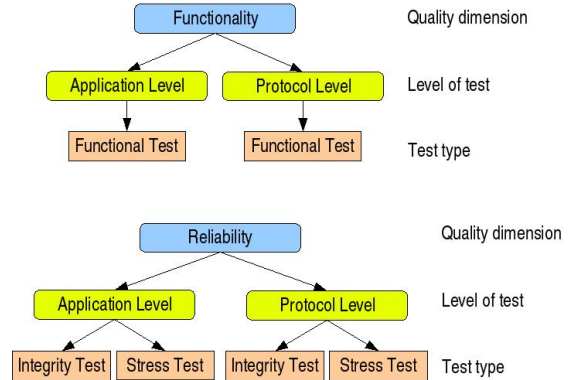


Figure 3: Mapping of test types to levels and quality dimensions.

comprehensive manner. It will help to specify the test cases and to verify that all relevant test cases are covered in order to determine the compliance. At the same time it will help to structure and audit the implementation of the tests. Moreover, we will be able to state a certain degree of compliance, e.g., in percentage of application-level functionality, by using this test categorization.

To keep the test space feasible, we do not test all possible combinations of TPM command calling, but we test all command parameters at the application and protocol level. For this reason, we need to define appropriate input data sets so that we cover the most common cases. To find those input data sets we also categorize the TPM commands regarding their main functionality and the different states a TPM can take according the specification.

A detailed analysis of the TPM specification results in a dependency graph that defines the input/output and calling dependencies of all TPM commands. To determine the compliance to the TPM specification we need to find a mapping to the graph. We do not need to test all possible paths in the graph, we only have to find reduced paths to cover all relevant

test cases. Those paths can be used to derive appropriate input data sets for the tests.

Having the dependency graph we can derive a general abstract finite state machine of the TPM. A state will contain several combination of commands. For determining the specification compliance it will only be necessary to test the TPM in all states and all state transitions instead of testing all combination of commands. This will reduce the test space and keep it feasible.

## 4 Testsuite Implementation

Our test suite implementation is a restricted instantiation of the test strategy defined in Section 3. All tests performed within the test suite have been selected by analyzing the possible functionalities offered by the selected software needed for the test suite implementation (see Section 4.2). Missing tests which are needed for a whole TPM specification compliance test suite are work in progress. The first implementation shall be seen as a prototype to give a first impression about the need for compliance tests. Before implementing a test suite, the following aspects should be considered:

**Test Subjects:** A test subject does always consist of only one TPM command (target of test). After identifying and defining the test subject, one shall derive the test cases that shall be tested for this test subject within the test suite by analyzing the dependencies and the command structure.

**Modularity:** Every test subject is implemented separately.

**Results:** Depending on the test subject, the test case and the mode of operation, one shall describe the expected test results. Take into

account, that the test results may differ depending on the state the TPM is currently in.

**Dependencies (Requirements):** The dependencies for a test subject have to be described. For example if the sealing functionality shall be tested, the requirements would be: TPM owner set, storage key created, storage key loaded, key password and data password available and the corresponding storage key handle. The requirements for our test implementation can be seen in our dependency graph shown in Figure 6.

**Coverage:** It should be assured that all test subjects are considered. Therefore, a test coverage has to be defined to fulfill the determination of compliance requirement.

**Cleanup:** After performing a test, it should be assured, that the TPM is in the same state as it was before. In particular, this means: unload all loaded keys, free all used resources, terminate all key handles, switch back eventually changed authorization data to the original. In case the test not being undoable (e.g., PCR extension), make sure to not harm any security related issue (e.g., do not clear ownership, if a subset of keys exist, which are actually used during normal operations). This will support the fulfillment of the traceability and the comparability requirements.

### 4.1 Test Configuration

In order to allow end-users as well as TPM developers to do TPM testing, the test environment shall meet the requirement of cost-efficiency. Ideally, a regular PC system (e.g., desktop board with standard CPU or a TPM-equipped laptop computer) should be usable. This means that the surrounding environment is not comparable to the environment of a de-

velopment laboratory. However, this will fulfill the comparability requirement.

The test suite itself shall be open source software, so that the test suite implementation is extensible, reviewable and that it can be evaluated. In order to communicate with the TPM, a TPM device driver has to be available and must work under the operating system. Ideally, this driver shall be equipped with enhanced debugging support (if available from the chip). In order to do time measurements, the test environment shall be *idle*, which means that no CPU consuming applications are running in the background. Thus, the test suite has to be executed in a pure console mode to minimize the amount of occurring interrupt requests (if the TPM uses IRQ handling).

In order to compare the test results with other TPM chips, the test suite shall be identical on every platform to be tested. Another requirement is the software used for the test suite (i.e., operating system, libraries, compiler). These should be well tested itself and manifested.

## 4.2 TPM Library *libtpm*

In order to fulfill the test suite requirements as described before, it was a reasonable way to implement a test suite prototype by reusing as much as possible of existing open source TPM software. Currently, only some open source TPM applications exist, each following different approaches.

The *libtpm*<sup>5</sup> is the first open source implementation of the TCG specification 1.1b. It has been developed by IBM in 2003 and offers the most common features of the TPM. Since the *libtpm* is working directly with the TPM on the protocol level, we develop our test suite prototype based on the *libtpm*<sup>6</sup>. The

<sup>5</sup>We use *libtpm* version 3.0.3 under Linux

<sup>6</sup>Meanwhile, IBM has developed an open source TSS implementation called "*TrouSerS*", which we might use

advantages are:

- Direct access to the input buffer which will be sent to the TPM.
- Direct access to the output buffer which is received from the TPM (and therefore the return codes).
- Single point of interaction, because all communication with the TPM is done in one *transmit*-function inside the *libtpm*.
- TPM features are combined into a library, which can easily be linked to own test applications.
- All test subjects can easily be selected and called separately.

Figure 4 shows the overall test hierarchy and the interaction between our test suite implementation and the *libtpm*. In order to manage the test subjects and their dependencies, we have developed a test suite management application. The management software selects the test subjects and the test cases. Afterwards, the *libtpm* is notified about the selected test subject and test cases. We use the tpm utilities provided by the *libtpm* to perform the desired tests. Those utilities themselves use the *libtpm* to build the command structure as defined by the specification and to communicate with the TPM by using a *transmit*-function. Instead of allowing the *libtpm* to directly communicate with the TPM chip, our test framework modifies the buffer in the selected way specified by the test case, sends it to the TPM, receives the response from the chip and afterwards analyzes the results. The obtained information will then be collected and returned to the test suite management application.

Although the *libtpm* does not fully implement the whole functionality provided by the libtpm in future implementations

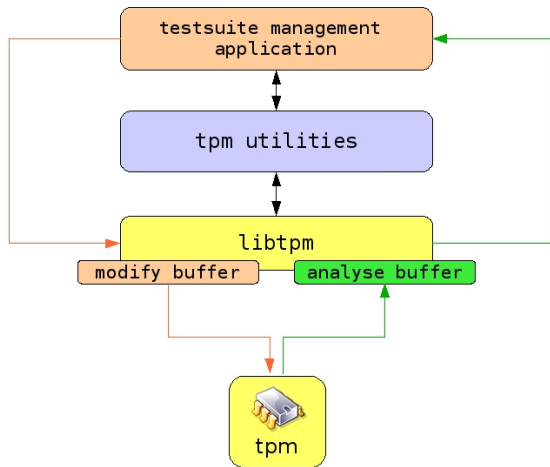


Figure 4: libtspm modification

TCG specification 1.1b, it is nevertheless useful as a prototype implementation to make a first testimony about TCG compliance. Figure 5 shows an overview of the current implementation status of our test suite with respect to the previously defined test strategy. In the next Section we describe, which tests can be achieved with the usage of the *libtpm*.

### 4.3 Application-Level Tests

The first part of tests will cover the functionality of the TPM. Due to the dependencies between different TPM commands, a TPM functionality test on the application level will cover more than one TPM command<sup>7</sup>. Figure 6 shows the application level functional tests we perform and their dependencies where each node represents a function at application-level, and a dashed arrow represents the dependency between the functionalities. Note that Figure 6 only gives an abstract view of the dependency

<sup>7</sup>For example, creating a key within the application level for the TPM will cover at least “TPM\_OSAP”, “TPM\_GetCapability”, “TPM\_CreateWrapKey” and “TPM\_TerminateHandle” commands on the protocol level.

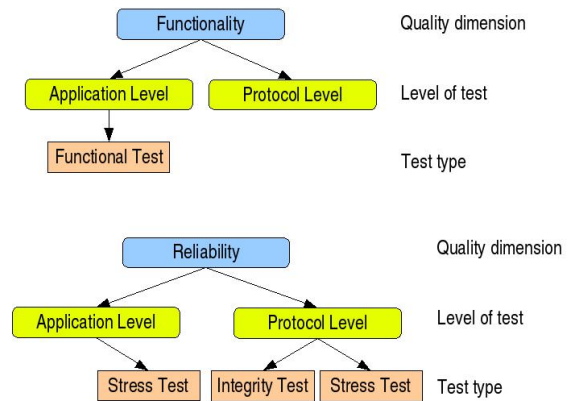


Figure 5: Mapping of test types in the current test suite implementation.

graph considering the main relations. However, in some cases the dependencies may be more subtle and hence a the graph may include more details (e.g., because commands can change internal states and capabilities of a TPM during runtime.)

#### 4.3.1 Functional Tests

The required functional tests at the application level are briefly described below.

**TPM Capabilities:** Read out all possible TPM capabilities as described in Section 2.

**Hardware random number generator:** Request the maximum number of available random bytes and measure the speed. By collecting a huge number of random numbers, one can perform additional tests on the received random numbers in order to qualify the randomness and security of the TPM’s random number generator.

**Take Ownership:** Take Ownership of the TPM in order to enable further TPM com-

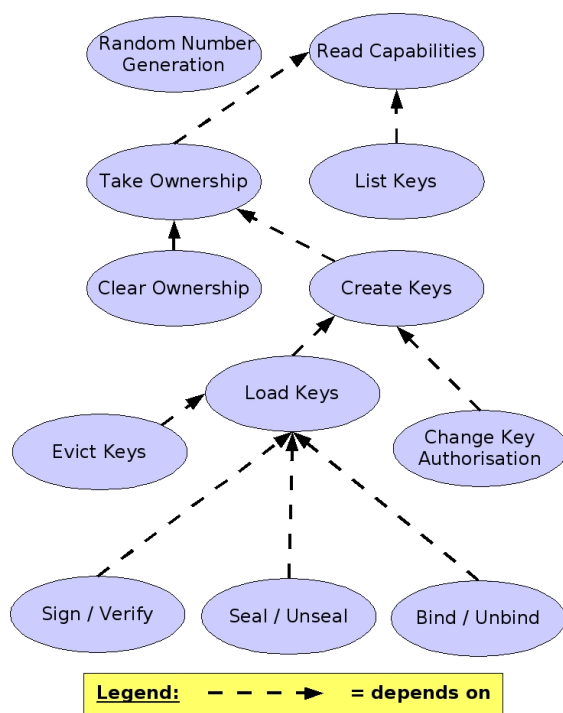


Figure 6: Dependency graph of functional tests.

mands.<sup>8</sup>

**Create keys:** Create all possible subkeys as defined in Section 2.

**Load keys:** Load all created subkeys. This will test, if the TPM is able to decrypt the given key with its private part of the SRK. Afterwards, the TPM will return a key handle which is needed to perform key operations with this key (depending on the key properties).

**List keys:** Tests if the TPM has loaded any keys and returns the depending key handles.

**Change Key Authorization:** Changes the authorization data for a given key. This key can either be the Storage Root Key (SRK) or any subkey created within the TPM (see Section 2).

**Key operations:** Use keys loaded into the TPM to perform key operations, e.g., by sealing, unsealing, unbinding or signing of data.

**Evict keys:** After successful usage of keys, evict<sup>9</sup> loaded keys out of the TPM in order to free TPM resources.

**Clear Ownership:** Clear the ownership of the TPM.

### 4.3.2 Integrity Tests

Currently, we do not have implemented integrity tests at the application level. This is

<sup>8</sup>e.g., “TPM.CreateWrapKey”, which depends on an owner being set inside the TPM.

<sup>9</sup>By evicting a key, the key will be deleted out of TPMs internal memory. All associated key handles are invalidated and the keyslot resource is available again. The key itself is still available outside the TPM.

due to the fact that the application-level software, i.e., *libtpm*, already intercepts and handles incorrect input parameters at this level. Thus, no TPM command would really be tested for correct behavior this way. Instead, we perform integrity tests at the protocol level, where we are able to modify the parameter data structures of the TPM commands in order to insert incorrect values to test the TPM behavior.

### 4.3.3 Stress Tests

**Key related stress tests:** This test consists of all actions that are related to keys, e.g.,

- Create all possible subkeys regarding the combination of key type, key size and key attributes as defined in Section 2.
- Load all possible subkeys.
- Load more keys into TPM than keyslots available.
- Evict all keys.
- Evict a key with an invalid key handle.
- Load two keys, evict the first one and load a third key and compare the received key handle to the first freed to check, whether the TPM assigns new key handles or if it uses recurrent and therefore predictable key handles.
- Test the quality of the key handles regarding the security aspects defined in Section 2.1.2.
- Try all possible combinations of keys and key operations, e.g., use a storage key to perform binding.
- Perform long-term key creation and loading. In detail, try to create  $n$  keys of any type and try to load and evict these keys alternately.

### Access/authorization related stress test

This test concerns the TPM behavior when stressing the authorization/access mechanisms.

By performing a dictionary attack<sup>10</sup> on the TPM, the behavior of the TPM is analyzed. If changes in behavior is observed because of countermeasures (see Section 2.1.3) perform a valid authentication (e.g., creating or loading a key) and afterwards continue the dictionary attack.

## 4.4 Protocol-Level Tests

Tests on the protocol level will mainly focus on performing integrity and stress tests on one specified test subject only. The functionality tests on the protocol level have – in our test suite prototype implementation – already been implicitly covered by the tests done in the application level.

### 4.4.1 Functional Tests

Currently, we do not have implemented functional test at the protocol level. The coverage of functional tests on the protocol level is implicitly done through functional tests of the application level, since tests on the higher level can only be successful if the underlying commands have been executed correctly and if the TPM behaves in the correct manner. Moreover, most commands on the protocol level require certain input parameters (e.g., key or session handles), which have to be available before executing the desired command. Due to this fact, the inputs of the command are depending on the output of prior commands. There-

---

<sup>10</sup>A dictionary attack is a technique for defeating an authentication mechanism by trying to determine its passphrase by searching a large number of possibilities. In contrast to a brute force attack, where all possibilities are searched, a dictionary attack only tries possibilities which are most likely to succeed, typically derived from a list of words in a dictionary.

fore, either test vectors have to be defined or the tests have to be done implicitly on a higher level.

The dependency graph on the application level in Figure 6 is transformed into an activity graph in Figure 8. This graph shows the TPM commands (in the required order) where each path from start to end represents one application level node of Figure 6. For a better overview the dependencies between the application level functions and the corresponding set of TPM commands are shown in Figure 7. Any high level dependencies between the different commands are not considered. This activity graph is related to the tests we have performed in our implementation and it represents a path of the complete test suite needed for TCG specification.

#### 4.4.2 Integrity Tests

Integrity tests at the protocol level are performed on one single TPM command by manipulating the command structure described in Section 2. Manipulating either one of those parameters has to lead to an error. Integrity tests on the protocol level will test the behavior of the TPM depending on the modified field entry inside the command. For every command (test subject) which has to be tested, there exist as many test cases as the command consists of field entries. During the modification, only one parameter field has to be changed in order to predict the expected result. Those are defined by the TCG specification in the Section “Return Codes”<sup>11</sup>. In our test suite implementation we cover integrity tests on the protocol level for all commands, which are executable via the *libtpm* utilities as described in Section 4.3. We therefore call the selected test subject as often as needed to be able to modify every field entry once (if possible).

<sup>11</sup>For example, modifying the “TPM\_AUTHDATA”-field shall lead to *Error 1 - Authentication failed*

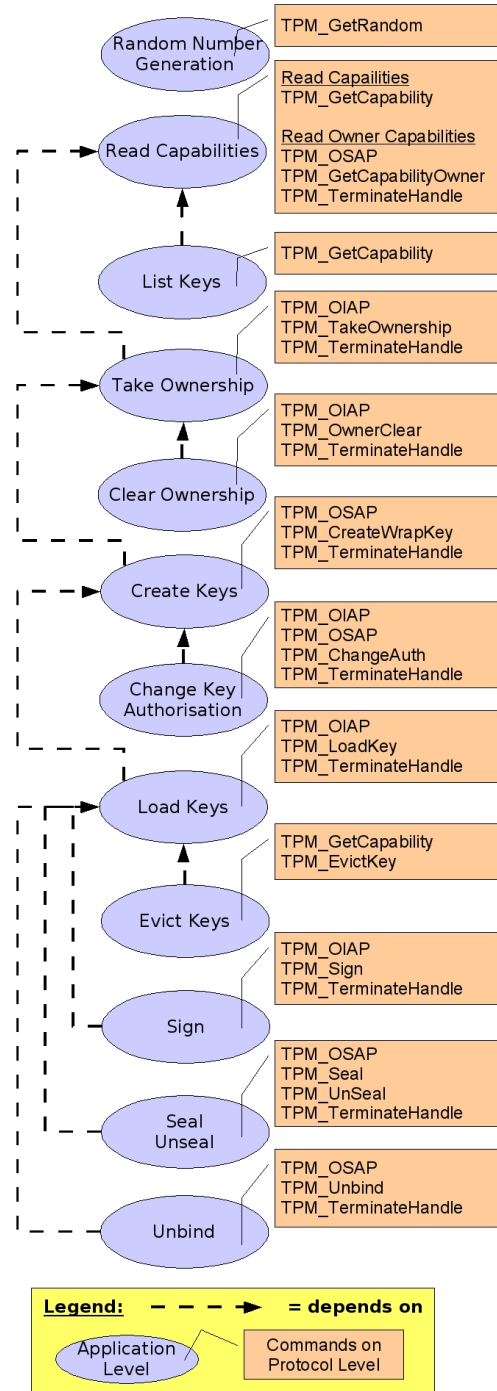


Figure 7: Dependency graph merging application and protocol level.

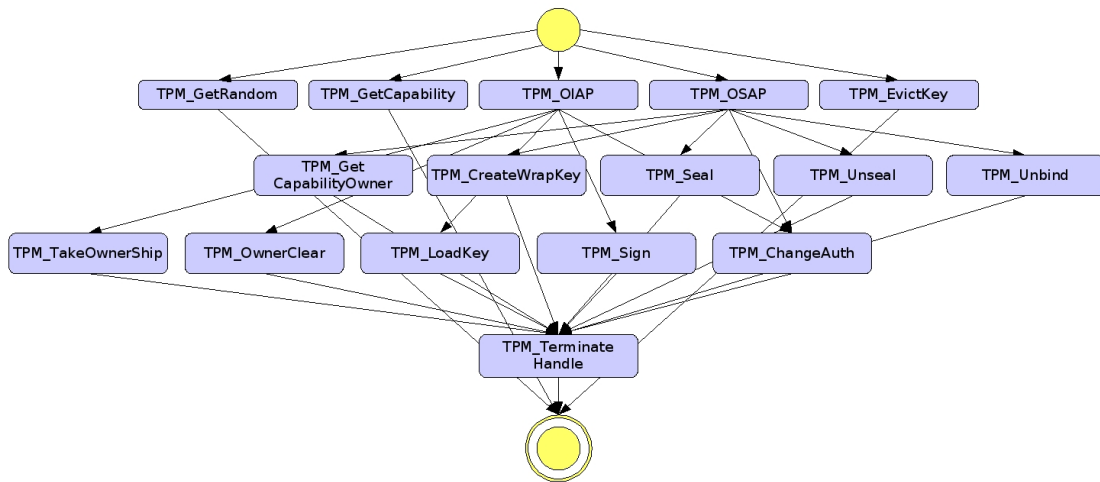


Figure 8: Activity graph of TPM commands on protocol level.

**Remark:** In concrete implementation of test suite, one should take care, that manipulating the authorization field might result into a locked TPM state (see Section 2.1.3).

#### 4.4.3 Stress Tests

**Session related stress tests:** This test consists of all actions that are related to sessions. The TPM specification defines several protocols on how to authenticate with the TPM and how to create shared session secrets (see Section 2). For each session, which is agreed upon via the session protocols, the TPM has to have internal memory to store the session handle, the session shared secret and all other properties assigned to this session. Due to the limited amount of the TPMs internal memory, its number of concurrent sessions is limited. In our implementation, we have performed the following stress tests:

- Open as many concurrent OIAP / OSAP sessions possible with the TPM.
- Close all sessions.

- Close a non-existing session.
- Open two sessions, close the first one and open a third session and compare the received session handle to the first freed to check, whether the TPM assigns new session handles or if it uses recurrent and therefore predictable session handles.
- Open an OSAP session with all possible entity types.
- Open alternating OIAP and OSAP sessions to see, if the session handles come from the same session pool.
- Analyse the quality of the session handles (see Section 2.1.2).

## 5 Test Result Sample

Based on our test strategy in Section 3 and our test prototype described in Section 4 we have tested several TPMs as listed in Table 1. A sample of relevant results can be found in Appendix A. For functional tests we also include

some additional side information, i.e., performance measurements, which are not relevant for compliance tests but we believe that they are important for the practice.<sup>12</sup> More concretely, the appended tables contain the following tests including a bug summary:

1. Application Level - Function Tests
2. Application Level - Stress Tests
3. Protocol Level - Integrity Tests
4. Protocol Level - Stress Tests

TPM Vendor	Chip Type
Infineon	1.1b
Atmel	1.1b
National Semiconductor	1.1b
ST Microelectronics	1.2
Infineon	1.2

Table 1: Tested TPM modules.

## 5.1 Security Implications

The analysis of the protocol level integrity test results shows – as expected – that different TPM implementations indeed differ in their behavior, which does not essentially mean that the TPMs are not compliant to the specification. However, several TPMs have non-compliant behavior with respect to the TCG specifications.

We show the possibility that certain compliance deviations and inappropriate implementation may have crucial security impact. We were able to exploit this in case of one widespread TPM (see Appendix A). Some of the security implications are considered in the following.

<sup>12</sup> In particular, measuring time consumption may reveal some information about TPM’s internal behavior (see Section 5.1.2).

### 5.1.1 Dictionary Attack

As mentioned in Section 2.1.3, the TCG specification 1.2 requires countermeasures against dictionary attacks. We have tested the implementations of protection mechanisms against dictionary attacks by performing false authentications to the TPM and analysing the resulting behavior.

In our test, we repeatedly send invalid authorized commands to the TPM. As soon as the TPM starts increasing the response time, we send valid commands with valid authentication. We identified, that the protection mechanism of one TPM implementation resets its locked state immediately after a successful authorization allowing us to continue the dictionary attack but this time without the protection even without executing the “TPM\_ResetlockValue” command.

### 5.1.2 Profiling

Sending a command to the TPM should ideally lead to a result from the TPM chip. This response includes a return code, which can be used to analyze how the TPM processes the incoming data internally. Sending a correct command with valid parameters has to lead to the return code “TPM\_SUCCESS (0)”. If one or more of the parameters are modified, one can figure out, whether the TPM has recognized that manipulation and – depending on the return code – which parameter is checked first by the TPM. Even the time needed to perform the answer might be interesting, but this has not been explored inside our test suite.

Comparing the results of different TPM vendor implementations and comparing the results of different test cases, one may be able to obtain a detailed profile of the TPM behavior. We figured out that some TPM implementations have bugs inside their parameter handling (as listed in the integrity tests table in

Appendix A).

This means that certain TPMs do not check certain parameters inside the command buffer and therefore return “TPM\_SUCCESS” instead of an error when the command has been manipulated. Furthermore, some TPMs return non-specified return codes completely unrelated to the actual command. This may be a useful as an entry point for an attack by analyzing the internals of the TPM.

### 5.1.3 Standard SRK password attack

IBM has defined a standard SRK password<sup>13</sup>, which may be used for storage authorization. This password is settable during taking ownership with the *libtpm*. This might lead to a security risk, since an attacker can very easily test, if the standard SRK password has been taken and then create new keys and load them into the TPM.

### 5.1.4 Handles

The TPM 1.2 specification requires certain quality standards for the randomness of handles (see Section 2.1.2). Those include the key handles returned during a “TPM\_LoadKey” as well as any session handle created with “TPM\_OIAP” or “TPM\_OSAP”. The main problem relies in the missing freshness of the received handle. If an attacker is able to evict a key out of the TPM, the associated key handle is deleted inside the TPM and any further reference to this key handle (e.g., through a key operation) is impossible. By ignoring this requirement, an attacker can easily figure out (due to the profiling property as discussed in Section 5.1.2) how the TPM generates and associates session and key handles and can therefore perform an attack using this information if, for example, he knows that a freed key handle will be immediately assigned to the next

<sup>13</sup>The password is ”SRK\_PWD”

key loaded into the TPM. If the attacker is aware of the SRK password (due to a dictionary attack or due to a standard password), it is possible to evict a key with certain properties, create an own key and load that key into the TPM. Since the TCG specifications do not consider performance issues, an application executing a TPM command has no estimation about the processing time. Therefore, this attack can be extended to a man-in-the-middle attack, where e.g., a malicious device driver may intercept the communication with the TPM, analyse the command and exchange the key, the key handle refers to. An application trying to perform a key operation (e.g., sealing) might not even realize that the key has been exchanged. Therefore, data to be sealed might be encrypted with a key not even in the possession of the regular user of the platform.

## 6 Conclusion

Many vendors currently equip their platforms and devices with Trusted Platform Modules (TPM) claiming to be TCG compliant. However, there is no feasible way for users of these systems to verify this fact.

To face this problem we have developed a prototype test suite for TPM compliance tests according to the TCG specification. The test results show the non-compliance and bugs of several TPM implementations. We present our testing strategy, analysis of results, discuss some issues that may have strong security impact. We point out the corresponding security problems in case of a widespread TPM chip. Although our tests cover only a part of the TCG specification we believe that the results are reasonably representative. We are currently working on a complete test suite specification for TPM implementations.

## References

- [1] TCPA Main Specification, Version 1.1b. [https://www.trustedcomputinggroup.org/specs/TPM/TCPA\\_Main\\_TCG\\_Architecture\\_v1\\_1b.pdf](https://www.trustedcomputinggroup.org/specs/TPM/TCPA_Main_TCG_Architecture_v1_1b.pdf), February 2002.
- [2] ATMEL. AT97SC3201 — The Atmel Trusted Platform Module. [http://www.atmel.com/dyn/resources/prod\\_documents/doc5010.pdf](http://www.atmel.com/dyn/resources/prod_documents/doc5010.pdf), August 2004.
- [3] ATMEL. AT97SC3203 Advanced Information Summary. [http://www.atmel.com/dyn/resources/prod\\_documents/5116s.pdf](http://www.atmel.com/dyn/resources/prod_documents/5116s.pdf), July 2005.
- [4] ATMEL. AT97SC3203S for SMBus Protocol Summary. [http://www.atmel.com/dyn/resources/prod\\_documents/5132s.pdf](http://www.atmel.com/dyn/resources/prod_documents/5132s.pdf), August 2005.
- [5] ATMEL. Trusted Platform Module AT97SC3201 Summary. [http://www.atmel.com/dyn/resources/prod\\_documents/2015s.pdf](http://www.atmel.com/dyn/resources/prod_documents/2015s.pdf), June 2005.
- [6] BEIZER, B. *Software Testing Techniques*, 2nd ed. International Thomson Computer Press, 1990.
- [7] BEIZER, B. *Black Box Testing*. John Wiley & Sons, 1995.
- [8] BROADCOM. Broadcom Revolutionizes LAN Communications by Introducing the World's First PCI Express Gigabit Ethernet Controllers for Server, Desktop and Mobile PCs. <http://www.broadcom.com/press/release.php?id=461159>, October 2003.
- [9] BROADCOM. BCM5752 Product Brief. <http://www.broadcom.com/collateral/pb/5752-PB00-R.pdf>, 2005.
- [10] BROADCOM. BCM5752M Product Brief. <http://www.broadcom.com/collateral/pb/5752M-PB00-R.pdf>, 2005.
- [11] BROADCOM. Broadcom Controllers Integrate TPM 1.2 enabling OEMs to Offer Hardware-Based Security as a Standard Feature on all PCs. <http://www.broadcom.com/press/release.php?id=700509>, April 2005.
- [12] INFINEON TECHNOLOGIES AG. Product Brief — TPM 1.2 Hardware. <http://www.infineon.com/tpm>, May 2005.
- [13] JACOBSON, I., BOOCH, G., AND RUMBAUGH, J. *The Unified Software Development Process*. Addison-Wesley, 1999.
- [14] KRUCHTEN, P. *The Rational Unified Process: An Introduction*, 3rd ed. Addison-Wesley, 2004.
- [15] NATIONAL SEMICONDUCTOR. Product Brief: PC8374T SafeKeeper Desktop TrustedI/O. [http://www.winbond-usa.com/products/winbond\\_products/pdfs/APC/PC8374T.pdf](http://www.winbond-usa.com/products/winbond_products/pdfs/APC/PC8374T.pdf), August 2004.
- [16] Data Brief: ST19WP18-TPM-A Trusted Platform Module (TPM). <http://www.st.com/stonline/products/literature/bd/10425.pdf>, 2004.
- [17] Data Brief: ST19WP18-TPM-B Trusted Platform Module (TPM). <http://www.st.com/stonline/products/literature/bd/10425.pdf>, 2004.
- [18] Data Brief: ST19WP18-TPM-B Trusted Platform Module (TPM).

<http://www.st.com/stonline/products/literature/bd/10425.pdf>, 2004.

- [19] Data Brief: ST19WP18 Trusted Platform Module (TPM). <http://www.st.com/stonline/products/literature/bd/10425.pdf>, 2004.
- [20] TCG. Trusted Computing Group. <http://www.trustedcomputinggroup.org>.
- [21] TRUSTED COMPUTING GROUP (TCG). TPM Main Specification — Part 1: Design Principles. [https://www.trustedcomputinggroup.org/groups/tpm/mainP1DP\\_rev85.zip](https://www.trustedcomputinggroup.org/groups/tpm/mainP1DP_rev85.zip), February 2005.

# Appendix A

## **Sample Test Results**

**Application Level – Functional Tests:**

<b>TPM Chip:</b>	<b>Infineon SLD 9630</b>	<b>Atmel AT97SC3201</b>	<b>National Semiconductor</b>	<b>STM ST 19 WP 18</b>	<b>Infineon SLB 9635</b>
Mainboard	Intel D865 GRH	IBM Thinkpad T41p	IBM Thinkpad T43	Intel D945	Intel D865 GLC
used operating system	Linux 2.6.16-rc1	Linux 2.6.14	Linux 2.6.15-mm4	Linux 2.6.12-rc1	Linux 2.6.16-rc1
<b>Capability Information:</b>					
TPM version	1.1b	1.1b	1.1b	1.2	1.2
TPM vendor	Infineon (IFX)	Atmel (ATML)	National Semiconductor (NSM)	ST Microelectronics (STM)	Infineon (IFX)
TPM firmware version	1.1.1.6	1.1.0.6	1.1.4.22	1.1.0.0	1.1.0.0
Platform Configuration Registers (PCRs)	16	16	16	24	24
Data Integrity Registers (DIRs)	16	2	2	1	1
Available keyslots	4	10	9	9	10
Available monotonic counters	/	/	/	4	8
Maximum counter handles	/	/	/	200	8
Time between counter increments	/	/	/	7 seconds	5 seconds
<b>Capability Algorithms:</b>					
RSA (mandatory)	yes	yes	yes	yes	yes
DES	no	no	no	no	no
3DES	no	no	no	no	no
SHA (mandatory)	yes	yes	yes	yes	yes
HMAC (mandatory)	yes	yes	yes	yes	yes
AES	no	no	no	no	no

<b>TPM Chip:</b>	<b>Infineon SLD 9630</b>	<b>Atmel AT97SC3201</b>	<b>National Semiconductor</b>	<b>STM ST 19 WP 18</b>	<b>Infineon SLB 9635</b>
<b>Capability Protocols (mandatory):</b>					
Object-Independent Authorization Protocol (OIAP)	yes	yes	yes	yes	yes
Object-Specific Authorization Protocol (OSAP)	yes	yes	yes	yes	yes
Authorization-Data Insertion Protocol (ADIP)	yes	yes	yes	yes	yes
AuthData Change Protocol (ADCP)	yes	yes	yes	yes	yes
OWNER	yes	yes	yes	yes	yes
<b>Capability Optional Command Ordinals:</b>					
TPM_ORD_CreateMaintenanceArchive	no	no	no	no	no
TPM_ORD_LoadMaintenanceArchive	no	no	no	no	no
TPM_ORD_KillMaintenanceFeature	no	no	no	no	no
TPM_ORD_LoadManuMaintPub	no	no	no	no	no
TPM_ORD_ReadManuMaintPub	no	no	no	no	no
TPM_ORD_Init	no	no	yes	yes	no
TPM_ORD_SetRedirection	no	no	yes	no	no
TPM_ORD_FieldUpgrade	yes	no	no	no	yes

<b>TPM Chip:</b>	<b>Infineon SLD 9630</b>		<b>Atmel AT97SC3201</b>		<b>National Semiconductor</b>		<b>STM ST 19 WP 18</b>		<b>Infineon SLB 9635</b>	
TPM_ORD_SaveKeyContext	yes		no		no		yes		no	
TPM_ORD_LoadKeyContext	yes		no		no		yes		no	
TPM_ORD_SaveAuthContext	yes		no		no		yes		no	
TPM_ORD_LoadAuthContext	yes		no		no		yes		no	
<b>Random Number Generator:</b>	bytes per request	elapsed time (sec.)	bytes per request	elapsed time (sec.)	bytes per request	elapsed time (sec.)	bytes per request	elapsed time (sec.)	bytes per request	elapsed time (sec.)
TPM_GetRandom	256 Bytes	< 1	768 Bytes	< 1	1050 Bytes	< 1	136 Bytes	< 1	1257 Bytes	< 1
<b>TPM Command: (operation mode)</b>	specification compliant	elapsed time (sec.)	specification compliant	elapsed time (sec.)	specification compliant	elapsed time (sec.)	specification compliant	elapsed time (sec.)	specification compliant	elapsed time (sec.)
TPM_OIAP (s1   s5)	yes	< 1	yes	< 1	yes	< 1	no***	< 1	yes	< 1
TPM_OSAP (s1   s5)	yes	< 1	yes	< 1	yes	< 1	no***	< 1	yes	< 1
TPM_TerminateHandle (s1   s5)	yes	< 1	yes	< 1	yes	< 1	yes	< 1	yes	< 1
TPM_Reset (s1   s5)	yes	< 1	yes	< 1	yes	< 1	yes	< 1	yes	< 1
TPM_GetCapability (s1   s5)	yes	x	yes	x	yes	x	yes	x	yes	x
TPM_GetCapabilityOwner (s1)	yes	x	yes	x	yes	x	yes	x	yes	x
TPM_PcrRead (s1   s5)	yes	< 1	yes	< 1	yes	< 1	yes	< 1	yes	< 1
TPM_TakeOwnership (s5)	yes	3 – 4	no**	30 – 40	yes	5 – 6	yes	20 – 30	yes	1 – 2
TPM_OwnerClear (s1)	yes	4 – 5	yes	< 1	yes	< 1	yes	< 1	yes	< 1
TPM_GetPubKey (s1   s5)	yes	< 1	yes	< 1	yes	< 1	<i>not tested</i>	x	<i>not tested</i>	x

<b>TPM Chip:</b>	<b>Infineon SLD 9630</b>		<b>Atmel AT97SC3201</b>		<b>National Semiconductor</b>		<b>STM ST 19 WP 18</b>		<b>Infineon SLB 9635</b>	
<b>TPM Command: (operation mode)</b>	specification compliant	elapsed time (sec.)	specification compliant	elapsed time (sec.)	specification compliant	elapsed time (sec.)	specification compliant	elapsed time (sec.)	specification compliant	elapsed time (sec.)
TPM_CreateWrapKey (pre-generated keys) (s1)	no*	2 – 3	yes	none	yes	5 – 6	yes	none	yes	1 – 2
TPM_CreateWrapKey (non pre-generated keys) (s1)	no*	40 – 100	yes	10 – 80	yes	40 – 60	yes	10 – 80	yes	10 – 30
TPM_LoadKey (s1)	yes	1 – 2	yes	1 – 2	yes	4 – 5	no***	3	yes	< 1
List Keys (via TPM_GetCapability) (s1)	yes	1 – 2	yes	< 1	yes	< 1	yes	< 1	yes	< 1
TPM_EvictKey (s1)	yes	< 1	yes	< 1	yes	1 – 2	yes	< 1	yes	< 1
TPM_ChangeAuth (s1)	no*	1 – 2	yes	1 – 2	yes	4	yes	1 – 2	yes	< 1
TPM_ChangeAuthOwner (s1)	yes	1 – 2	yes	1 – 2	yes	4	yes	1 – 2	yes	< 1
TPM_Sign (s1)	yes	1 – 2	yes	1 – 2	yes	2	yes	2 – 3	yes	1 – 2
TPM_Unbind (s1)	yes	1 – 2	yes	1 – 2	yes	3	yes	1 – 2	yes	< 1
TPM_Seal (s1)	no*	1 – 2	yes	< 1	yes	3	yes	< 1	yes	< 1
TPM_Unseal (s1)	yes	< 1	yes	1 – 2	yes	3	yes	2 – 3	yes	< 1
TPM_CreateCounter (s1)	/	/	/	/	/	/	yes	x	yes	x
TPM_IncrementCounter (s1)	/	/	/	/	/	/	no****	x	yes	x
TPM_ReadCounter (s1)	/	/	/	/	/	/	yes	x	yes	x
<b>Legend:</b>										
	*	<b>TPM 1.1b ET_SRK bug: Fixed via firmware upgrade or via libtpm patch (refer bugs)</b>								
	**	<b>TPM 1.1b AuthDataUsage bug: Fixed in TrouSerS (refer bugs)</b>								
	***	<b>TPM 1.2 handle bug:(refer bugs)</b>								
	****	<b>TPM 1.2 monotonic counter timing bug: (refer bugs)</b>								

**Application Level – Stress Tests:**

<b>TPM Chip:</b>	<b>Infineon SLD 9630</b>	<b>Atmel AT97SC3201</b>	<b>National Semiconductor</b>	<b>STM ST 19 WP 18</b>	<b>Infineon SLB 9635</b>
Mainboard	Intel D865 GRH	IBM Thinkpad T41p	IBM Thinkpad T43	Intel D945	Intel D865 GLC
used operating system	Linux 2.6.16-rc1	Linux 2.6.14	Linux 2.6.15-mm4	Linux 2.6.12-rc1	Linux 2.6.16-rc1
<b>Key Related Stress Test:</b>					
<b>pre-calculated keys</b>	~ 2	0	~ 6	0	~ 3
<b>Key handle quality description:</b>	The key handles do partly increase in minor or major steps.	Randomized key handles	The keyhandles have an ever-increasing value starting with key handle 0x00000001. After a TPM restart, the key handle still has its last value. Only after some hours of powered down state, the key handles are set back to 0	The key handles always start at 0x00000100. If a key handle is freed, it is immediately assigned to the next key.	The key handles are conform to the TCG specification handle requirement (Ref. TPM Spec 1.2, Rev. 85, Level 2, page 12)
<b>Access / Authorization Related Stress Test:</b>					
<b>Countermeasure against dictionary attack</b>	<b>not present</b>	<b>not present</b>	<b>present</b>	<b>present</b>	<b>present</b>
<b>Dictionary Attack description:</b>	/	/	After 10 invalid attempts, the TPM refuses any further command. After some seconds and a valid authentication, the TPM allows only one command in a timeframe of 5 seconds. After 10-30 minutes, the TPM works normal again	In case of a dictionary attack, the TPM starts increasing its answer time. The first 15 responses are received immediately, then every 2 commands the return time is increased by one second. After about 40 faulty authentication tries, the TPM response time is at about 10 seconds.	In case of a dictionary attack, the TPM cancels the execution after 10 invalid attempts. After further attempts, the TPM increases the time frame, in which a valid command can be executed.
<b>Dictionary Attack countermeasure resettable</b>	/	/	<b>no</b>	<b>yes</b> – the dictionary attack countermeasure resets itself after one successful authentication.	<b>no</b>

**Important Note:** This text explains how to read and understand the results shown in the next table.

The integrity tests on the protocol level are an excerpt upon the tests we performed. The TCG defines certain return codes for most of the possible conditions of the TPM. Those return codes can be found in the TCG specification in chapter „Return Codes“. The TPM, for example, returns „TPM\_AUTHFAIL“, if the verification of the command authorisation failed. But the TCG specification does not always specify, how or in which order the TPMs have to handle their incoming data.

During the development of our testsuite we explored the TCG specification and derived test cases for different test subjects. Any test case has an expected test result, but there might be cases, where more than one test result would be reasonable.

The entries in the table therefore contain either „passed“ if the test result is exactly the expected test result or they contain the received return code of the corresponding TPM command. If the result is not as expected, this does not mean, that the TPM is not compliant, but that the TPM handles the incoming data in a different order.

For example, the first test „TPM\_TEST\_TAG“ modifies the TPM\_TAG-field inside the command parameter structure. The expected result would be „TPM\_BADTAG“.

The results for this test show differences in the TPM behaviour depending on the TPM vendor. The Atmel TPM (second row) returns „TPM\_AUTHFAIL“, which means, that the TPM has first checked, if the authorisation of that command is ok. This still is a correct result, although it does not meet our expected test result (due to the modification of the command buffer).

In case the TPM is obviously not compliant to the TCG specification, we marked that entry bold and added the corresponding bug in the legend below.

**Protocol Level – Integrity Tests: (Sample)**

<b>TPM Command: (operation mode: s1)</b>	<b>Test case:</b>	<b>Test description:</b>				<b>Expected Test result:</b>
	<b>TPM_TEST_TAG</b>	This test modifies the „TPM_TAG“-field inside the parameter structure.				<b>TPM_BADTAG (30)</b>
	<b>Infineon SLD 9630</b>	<b>Atmel AT97SC3201</b>	<b>National Semiconductor</b>	<b>STM ST 19 WP 18</b>	<b>Infineon SLB 9635</b>	
TPM_OIAP	passed	passed	passed	TPM_INVALID_AUTHHANDLE	passed	
TPM_OSAP	passed	passed	passed	TPM_INVALID_AUTHHANDLE	passed	
TPM_TerminateHandle	passed	passed	passed	TPM_INVALID_AUTHHANDLE	/	
TPM_Reset	passed	passed	passed	/	/	
TPM_GetCapability	passed	passed	passed	TPM_INVALID_AUTHHANDLE	passed	
TPM_PcrRead	passed	passed	passed	TPM_INVALID_AUTHHANDLE	passed	
TPM_GetPubKey	passed	passed	passed	/	/	
TPM_CreateWrapKey	passed	TPM_AUTHFAIL	passed	TPM_INVALID_AUTHHANDLE	passed	
TPM_LoadKey / TPM_LoadKey2	passed	TPM_AUTHFAIL	passed	TPM_INVALID_AUTHHANDLE	passed	
TPM_EvictKey	passed	passed	passed	TPM_INVALID_AUTHHANDLE	passed	
TPM_Sign	passed	TPM_AUTHFAIL	passed	TPM_INVALID_AUTHHANDLE	passed	
TPM_Unbind	passed	TPM_AUTHFAIL	passed	TPM_INVALID_AUTHHANDLE	passed	
TPM_Seal	passed	TPM_AUTHFAIL	passed	TPM_INVALID_AUTHHANDLE	passed	
TPM_Unseal	passed	TPM_AUTHFAIL	passed	passed	passed	

<b>TPM Command: (operation mode: s1)</b>	<b>Test case:</b>	<b>Test description:</b>			<b>Expected Test result:</b>
	<b>TPM_TEST_PARAMSIZE_MINOR</b>	This test modifies the „TPM_PARAMSIZE“-field inside the parameter structure by decreasing the size parameter			<b>TPM_BAD_PARAM_SIZE (25)</b>
	<b>Infineon SLD 9630</b>	<b>Atmel AT97SC3201</b>	<b>National Semiconductor</b>	<b>STM ST 19 WP 18</b>	<b>Infineon SLB 9635</b>
TPM_OIAP	TPM_BADTAG	TPM_SIZE	TPM_IOERROR	TPM_INVALID_AUTHHANDLE	passed
TPM_OSAP	passed	passed	passed	<b>XXX</b>	passed
TPM_TerminateHandle	passed	passed	passed	/	/
TPM_Reset	TPM_BADTAG	TPM_SIZE	TPM_IOERROR	/	/
TPM_GetCapability	passed	passed	passed	<b>TPM_SUCCESS *****</b>	passed
TPM_PcrRead	TPM_BAD_INDEX	passed	passed	<b>TPM_SUCCESS *****</b>	TPM_BAD_INDEX
TPM_GetPubKey	passed	passed	passed	/	/
TPM_CreateWrapKey	TPM_INVALID_AUTHHANDLE	TPM_AUTHFAIL	passed	<b>XXX</b>	TPM_INVALID_AUTHHANDLE
TPM_LoadKey / TPM_LoadKey2	TPM_INVALID_AUTHHANDLE	TPM_AUTHFAIL	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE
TPM_EvictKey	passed	passed	passed	TPM_KEYNOTFOUND	/
TPM_Sign	TPM_INVALID_AUTHHANDLE	TPM_AUTHFAIL	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE
TPM_Unbind	TPM_INVALID_AUTHHANDLE	TPM_AUTHFAIL	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE
TPM_Seal	TPM_INVALID_AUTHHANDLE	TPM_AUTHFAIL	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE
TPM_Unseal	TPM_INVALID_AUTHHANDLE	TPM_AUTHFAIL	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE

<b>TPM Command: (operation mode: s1)</b>	<b>Test case:</b>	<b>Test description:</b>			<b>Expected Test result:</b>
	<b>TPM_TEST_PARAMSIZE_MAJOR</b>	This test modifies the „TPM_PARAMSIZE“-field inside the parameter structure by increasing the size parameter			<b>TPM_BAD_PARAM_SIZE (25)</b>
	<b>Infineon SLD 9630</b>	<b>Atmel AT97SC3201</b>	<b>National Semiconductor</b>	<b>STM ST 19 WP 18</b>	<b>Infineon SLB 9635</b>
TPM_OIAP	passed	passed	passed	<b>TPM_SUCCESS *****</b>	passed
TPM_OSAP	passed	passed	passed	<b>XXX</b>	passed
TPM_TerminateHandle	passed	passed	passed	/	/
TPM_Reset	passed	passed	passed	/	/
TPM_GetCapability	passed	passed	passed	<b>TPM_SUCCESS *****</b>	passed
TPM_PcrRead	passed	passed	passed	<b>TPM_SUCCESS *****</b>	passed
TPM_GetPubKey	passed	passed	passed	/	/
TPM_CreateWrapKey	TPM_INVALID_AUTHHANDLE	TPM_INVALID_KEYHANDLE	TPM_INVALID_AUTHHANDLE	<b>XXX</b>	TPM_INVALID_AUTHHANDLE
TPM_LoadKey / TPM_LoadKey2	TPM_INVALID_AUTHHANDLE	TPM_AUTHFAIL	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE
TPM_EvictKey	passed	passed	passed	<b>TPM_SUCCESS *****</b>	/
TPM_Sign	TPM_INVALID_AUTHHANDLE	TPM_AUTHFAIL	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE
TPM_Unbind	TPM_INVALID_AUTHHANDLE	TPM_AUTHFAIL	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE
TPM_Seal	TPM_INVALID_AUTHHANDLE	TPM_INVALID_KEYHANDLE	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE
TPM_Unseal	TPM_INVALID_AUTHHANDLE	TPM_AUTHFAIL	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE	TPM_INVALID_AUTHHANDLE

<b>TPM Command: (operation mode: s1)</b>	<b>Test case:</b>	<b>Test description:</b>				<b>Expected Test result:</b>
	<b>TPM_TEST_AUTHHANDLE</b>	This test modifies the „TPM_AUTHHANDLE“-field inside the parameter structure.				<b>TPM_INVALID_AUTHHANDLE (34)</b>
	<b>Infineon SLD 9630</b>	<b>Atmel AT97SC3201</b>	<b>National Semiconductor</b>	<b>STM ST 19 WP 18</b>	<b>Infineon SLB 9635</b>	
TPM_LoadKey / TPM_LoadKey2	passed	<b>TPM_SUCCESS *****</b>	passed	passed	passed	
TPM_Sign	passed	<b>TPM_SUCCESS *****</b>	passed	passed	passed	
TPM_Unbind	passed	<b>TPM_SUCCESS *****</b>	passed	passed	passed	
TPM_Seal	passed	<b>TPM_SUCCESS *****</b>	passed	passed	passed	
TPM_Unseal	passed	<b>TPM_SUCCESS *****</b>	passed	passed	passed	
<b>TPM Command: (operation mode: s1)</b>	<b>Test case:</b>	<b>Test description:</b>				<b>Expected Test results:</b>
	<b>TPM_TEST_KEYHANDLE</b>	This test modifies the „TPM_KEYHANDLE“-field inside the parameter structure.				<b>TPM_INVALID_KEYHANDLE (12) or TPM_KEYNOTFOUND (13)</b>
	<b>Infineon SLD 9630</b>	<b>Atmel AT97SC3201</b>	<b>National Semiconductor</b>	<b>STM ST 19 WP 18</b>	<b>Infineon SLB 9635</b>	
TPM_LoadKey / TPM_LoadKey2	TPM_KEYNOTFOUND	TPM_KEYNOTFOUND	TPM_KEYNOTFOUND	TPM_KEYNOTFOUND	TPM_INVALID_KEYHANDLE	
TPM_Sign	TPM_INVALID_KEYHANDLE	TPM_INVALID_KEYHANDLE	TPM_KEYNOTFOUND	<b>TPM_WRONG_ENTITYTYPE</b>	TPM_INVALID_KEYHANDLE	
TPM_Unbind	TPM_INVALID_KEYHANDLE	TPM_INVALID_KEYHANDLE	TPM_KEYNOTFOUND	<b>TPM_WRONG_ENTITYTYPE</b>	TPM_INVALID_KEYHANDLE	
TPM_Seal	TPM_INVALID_KEYHANDLE	TPM_INVALID_KEYHANDLE	TPM_KEYNOTFOUND	<b>TPM_WRONG_ENTITYTYPE</b>	TPM_INVALID_KEYHANDLE	
TPM_Unseal	TPM_INVALID_KEYHANDLE	TPM_INVALID_KEYHANDLE	TPM_KEYNOTFOUND	TPM_INVALID_KEYHANDLE	TPM_INVALID_KEYHANDLE	

<b>TPM Command: (operation mode: s1)</b>	<b>Test case:</b>	<b>Test description:</b>			<b>Expected Test result:</b>
	<b>TPM_TEST_AUTHDATA</b>	This test modifies the „TPM_AUTH“-field inside the parameter structure.			<b>TPM_AUTHFAIL (1)</b>
	<b>Infineon SLD 9630</b>	<b>Atmel AT97SC3201</b>	<b>National Semiconductor</b>	<b>STM ST 19 WP 18</b>	<b>Infineon SLB 9635</b>
TPM_LoadKey / TPM_LoadKey2	passed	passed	passed	passed	passed
TPM_Sign	passed	passed	passed	passed	passed
TPM_Unbind	passed	passed	passed	passed	passed
TPM_Seal	passed	passed	passed	passed	passed
<b>TPM Command: (operation mode: s1)</b>	<b>Test case:</b>	<b>Test description:</b>			<b>Expected Test result:</b>
	<b>TPM_TEST_AUTHDATA</b>	This test modifies the „TPM_AUTH“-field inside the parameter structure.			<b>TPM_AUTH2FAIL (29)</b>
	<b>Infineon SLD 9630</b>	<b>Atmel AT97SC3201</b>	<b>National Semiconductor</b>	<b>STM ST 19 WP 18</b>	<b>Infineon SLB 9635</b>
TPM_Unseal	passed	<b>TPM_AUTHFAIL *****</b>	passed	passed	passed
<b>TPM Command: (operation mode: s1)</b>	<b>Test case:</b>	<b>Test description:</b>			<b>Expected Test results:</b>
	<b>TPM_TEST_KEYUSAGE</b>	This test modifies the „TPM_KEY“-field inside the parameter structure.			<b>TPM_INVALID_KEYUSAGE (36) or TPM_INVALID_KEYHANDLE (12) or TPM_KEYNOTFOUND (13)</b>
	<b>Infineon SLD 9630</b>	<b>Atmel AT97SC3201</b>	<b>National Semiconductor</b>	<b>STM ST 19 WP 18</b>	<b>Infineon SLB 9635</b>
TPM_Sign	TPM_INVALID_KEYUSAGE	TPM_KEYNOTFOUND	TPM_INVALID_KEYUSAGE	TPM_INVALID_KEYUSAGE	TPM_INVALID_KEYHANDLE
TPM_Seal	TPM_INVALID_KEYHANDLE	TPM_KEYNOTFOUND	TPM_INVALID_KEYUSAGE	<b>TPM_INVALID_AUTHHANDLE</b>	TPM_INVALID_KEYHANDLE
TPM_Unseal	TPM_INVALID_KEYHANDLE	TPM_KEYNOTFOUND	TPM_INVALID_KEYUSAGE	TPM_INVALID_KEYUSAGE	TPM_INVALID_KEYHANDLE

<b>TPM Command: (operation mode: s1)</b>	<b>Test case:</b>	<b>Test description:</b>			<b>Expected Test result:</b>
	<b>TPM_TEST_BADINDEX</b>	This test modifies the „TPM_RegisterIndex“-field inside the parameter structure.			<b>TPM_BAD_INDEX (2)</b>
	<b>Infineon SLD 9630</b>	<b>Atmel AT97SC3201</b>	<b>National Semiconductor</b>	<b>STM ST 19 WP 18</b>	<b>Infineon SLB 9635</b>
TPM_PcrRead	passed	passed	passed	passed	passed

<b>TPM Command: (operation mode: s1)</b>	<b>Test case:</b>	<b>Test description:</b>			<b>Expected Test result:</b>
	<b>TPM_TEST_COUNTID</b>	This test modifies the „TPM_CountID“-field inside the parameter structure.			<b>TPM_BAD_COUNTER (69)</b>
	<b>Infineon SLD 9630</b>	<b>Atmel AT97SC3201</b>	<b>National Semiconductor</b>	<b>STM ST 19 WP 18</b>	<b>Infineon SLB 9635</b>
TPM_ReadCounter	/	/	/	<b>TPM_BAD_PARAMETER *****</b>	passed

**Legend:**

	passed	<b>passed means, that the received return code complies the expected test result</b>
	other than passed	<b>is the received return code. For details, see TCG specification, chapter „Return Codes“</b>
	*****	<b>TPM 1.2 monotonic counter return bug: (refer bugs)</b>
	*****	<b>TPM success return bug: (refer bugs)</b>
	*****	<b>TPM 1.1b second key return bug: (refer bugs)</b>
	XXX	<b>TPM crashed before finished with all test cases! No further tests were possible.</b>

**Protocol Level – Stress Tests:**

<b>TPM Chip:</b>	<b>Infineon SLD 9630</b>	<b>Atmel AT97SC3201</b>	<b>National Semiconductor</b>	<b>STM ST 19 WP 18</b>	<b>Infineon SLB 9635</b>
Mainboard	Intel D865 GRH	IBM Thinkpad T41p	IBM Thinkpad T43	Intel D945	Intel D865 GLC
used operating system	Linux 2.6.16-rc1	Linux 2.6.14	Linux 2.6.15-mm4	Linux 2.6.12-rc1	Linux 2.6.16-rc1
<b>concurrent OIAP sessions</b>	20	2	8	12	32
<b>concurrent. OSAP sessions</b>	20	2	8	6	32
<b>Session handle quality description:</b>	The session handle do partly increase in minor or major steps. If a session handle is freed, it is immidiately assigned to the next session.	The session handles always start at 0x00000000. If a session handle is freed, it is immidiately assigned to the next session.	The session handles always start at 0x00000000. If a session handle is freed, it is immidiately assigned to the next session.	The session handles always start at 0x00000000. If a session handle is freed, it is immidiately assigned to the next session.	The session handles are conform to the TCG specification handle requirement (Ref. TPM Spec 1.2, Rev. 85, Level 2, page 12)

**Bugs Summary:**

<b>TPM Chip:</b>	<b>Infineon SLD 9630</b>	<b>Atmel AT97SC3201</b>	<b>National Semiconductor</b>	<b>STM ST 19 WP 18</b>	<b>Infineon SLB 9635</b>
Mainboard	Intel D865 GRH	IBM Thinkpad T41p	IBM Thinkpad T43	Intel D945	Intel D865 GLC
used operating system	Linux 2.6.16-rc1	Linux 2.6.14	Linux 2.6.15-mm4	Linux 2.6.12-rc1	Linux 2.6.16-rc1
	<b>TPM 1.1b ET_SRK bug</b>	<b>TPM 1.1b AuthDataUsage bug</b>	/	<b>TPM 1.2 handle bug</b>	/
		<b>TPM 1.1b second key return bug</b>		<b>TPM 1.2 monotonic counter return bug</b>	
		<b>TPM success return bug</b>		<b>TPM 1.2 monotonic counter timing bug</b>	
				<b>TPM success return bug</b>	

**Bug description:**

<b>TPM 1.1b ET_SRK bug:</b>	The TPM does not differ between ET_KEYHANDLE and ET_SRK. Therefore, an authentication error occurs for keytype 0x0004. <i>Fixable via TPM Firmware Update or libtpm-patch</i>
<b>TPM 1.1b AuthDataUsage bug:</b>	The TPM does change the authdata-field inside the SRK-blob after taking ownership. For details refer to trousers-bugfix in <i>trousers/src/tspi/spi_tpm.c</i>
<b>TPM 1.1b second key return bug:</b>	The TPM does not check, whether the authorisation data failes for the first or the second key. If the second key authorisation fails, the TPM has to return „TPM_AUTH2FAIL“ instead of „TPM_AUTHFAIL“
<b>TPM 1.2 handle bug:</b>	TPM Spec 1.2, Rev. 85, Level 2, page 12: The three LSBs of the handle MUST contain the collision resistance values. The TPM MUST provide protection against handle collision. ... The three last LSB of the handle MUST be generated randomly.
<b>TPM 1.2 monotonic counter timing bug:</b>	„The TPM must support an increment rate of once every 5 seconds.“ (as specified in TCG spec 1.2)
<b>TPM 1.2 monotonic counter return bug:</b>	If requesting an invalid countId via TPM_readCounter, the TPM has to return TPM_BAD_COUNTER (69). Here TPM_BAD_PARAMETER (3) is returned
<b>TPM success return bug:</b>	By invalidating the send-buffer by modifying one parameter, the TPM returns TPM_SUCCESS (0) instead of an error. See table „Integrity tests“ for details and the failed testcase.