Chapter 1
Programming Interfaces for the TPM

Ronald Toegl, Thomas Winkler, Mohammad Nauman, Theodore W. Hong, Johannes Winter, and Michael Gissing

Abstract The paradigm of Trusted Computing promises a new approach to improve the security of embedded and mobile systems. The core functionality, based on a hardware component known as Trusted Platform Module (TPM), is widely available. However, integration and application in embedded systems remains limited at present, simply because of the extremely steep learning curve involved in using the programmer-facing interfaces. In this chapter, we describe the current state of the Trusted Computing Group's software architecture and present previous approaches to improve usability. We report on a novel design of a high-level API for Trusted Computing for Java which has been optimized for ease-of-use and clear abstraction of Trusted Computing concepts. We derive requirements and design goals and outline the API design. Finally, we show the application and benchmarks in embedded systems. The result of this effort has been standardized as Java Specification Request 321.

1.1 Introduction

Embedded Systems take many forms, such as mobile phones, industrial control systems, network devices, sensor nodes, and smart cards. Having become nearly ubiquitous, the world Embedded Systems market exceeds 100 billion USD [18]. Often, sensitive information is created, accessed, manipulated, stored, and communicated

R. Toegl (✉) • J. Winter • M. Gissing
Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria
e-mail: rtoegl@iaik.tugraz.at

T. Winkler
Pervasive Computing Group/Institute of Networked and Embedded Systems (NES),
Alpen-Adria Universitaet Klagenfurt, Lakeside Park B02b, A-9020 Klagenfurt, Austria

M. Nauman
Computer Science Research and Development Unit, Peshawar, Pakistan

T.W. Hong
University of Cambridge Computer Laboratory, William Gates Building,
15 J.J. Thomson Ave., Cambridge CB3 0FD, UK

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5_1

on such Embedded Systems. Thus, security needs to be considered throughout the design process [50], including hardware design and software development. Specifically, in the Trusted Computing approach, security is bootstrapped from a small dedicated piece of secure hardware, the Trusted Platform Module (TPM).

While most the major computer manufacturers are shipping servers, desktop and notebook computers containing TPMs with several hundreds of million of machines can be assumed to provide this hardware device [58], its application to Embedded Systems has only been limited. Major obstacles to the development of Trusted Computing enabled software have been the high complexity of the specification of the software stack that is used to manage the TPM and limited support for programming languages that support different hardware platforms [57, 60].

In particular, there has been insufficient support for platform-independent runtime environments like .NET [40], Android or Java. Such environments are particularly useful for implementing modern security solutions on heterogeneous platforms. For instance, several billions of devices support Java, and Oracle claims [43] that the Java developer community, with nine million members, is the largest of its kind. It is of little surprise that there have been a number of attempts to provide TPM libraries that target such a programming languages. Yet there was a lack of an established, generally-accepted Application Programming Interface (API) for TPM access.

In this chapter, which extends on [71], we describe the design of a *high-level* Java API for Trusted Computing, which has been published as an official Java *standard* [68]. Our goal in designing this API is to provide a simpler, high-level interface to the TPM while still adhering to the concepts and standards defined by the Trusted Computing Group. Benchmark results show the suitability in Embedded Systems using the TPM.

1.2 Trusted Computing in the Java Environment

1.2.1 Java for Embedded Systems

At the application layer, the Java programming environment has seen a broad adoption ranging from large-scale business applications hosted in dedicated data centers to resource constrained environments as found in mobile phones or Personal Digital Assistants (PDAs), set-top boxes, industrial control and even smart cards. Java program code [21] is not compiled to native machine code but to a special form of intermediate code, called byte code. This byte code is then executed by a virtual machine (VM) [35] called the Java VM. This characteristic makes Java an excellent choice for development aiming at heterogeneous environments. In contrast to conventional programming languages such as C or C++, Java is equipped with inherent security features supporting the development of more secure software. Among those features are automatic array-bounds checking, garbage collection and access control mechanisms. Additional aspects that distinguish Java from other environments are code-signing mechanisms and the verification of byte code when it is loaded. The class-loading mechanism separates privileged code and creates a sandbox for remotely fetched classes [19].

For Embedded Systems development, Java offers a number of advantages [65]. Its hardware-independent architecture hides specifics of the hardware and operating system, as it is abstracted through the platform-specific, often optimized implementation of the Java VM. The rich libraries of the Java Runtime Environment (JRE) offer much more features than operating system APIs. Java also eases the creation of network inter-operable embedded systems and simplifies the software development. For very small systems, Java ME offers small-footprint configurations of Java, albeit with limited functionality. The full-featured Java SE is found on PCs and medium-to-large (i.e. 32 MB of RAM or more) Embedded Systems.

With Java being a key component, Android [20] is now the first choice for mobile smart-phones. Based on a Linux kernel, it offers a broad application library framework and the Dalvik virtual machine with just-in-time compilation which is optimized for resource-restrained devices. In addition, Android is fully source-code compatible with Java. As of 2013, Android is the most widely used, off-the-shelf operating system in Embedded Systems [77].

Here also, Trusted Computing is very promising to further improve security. While generic cryptography is well supported with the Java Security Architecture, there is currently no established standard API for Trusted Computing available. Still, a large number of Java-based use cases have been demonstrated for Trusted Computing, using several existing approaches for Trusted Computing integration in Java.

1.3 TCG Software Architecture

1.3.1 The TCG Software Stack

The TPM design is intended to allow for cost effective implementations on hardware architectures with restricted resources, such as smart card platforms. Consequently, the functionality of the TPM is restricted to what is offered by its API. The TPM is not able to execute custom code, and even most of the features offered require auxiliary functionality implemented in software.

Of the TCG standards, the *TCG Software Stack (TSS)* [73] is responsible to access and manage the TPM and also to provide a programming interface for TC applications. The standard document is accompanied with C header and Web Services Description Language (WSDL) interface definition files. The target language for the standard is the C programming language [26].

The TSS offers a set of function calls that help perform a number of operations. These functionalities cover the setup and *administration* of the TPM, such as taking ownership, the setting of configurations or the querying of properties. With regards to the chain of trust, it is the task of the TSS to record a *Stored Measurement Log*

(SML) for tracing the measurements that led to the current PCR values. The life cycle of *cryptographic keys* is also controlled through the TSS, starting with the creation of public-private key-pairs. The limited resources of the TPM necessitate external, encrypted storage of the cryptographic material, either at run-time by swapping out keys from limited hardware key slots into main memory or filing in *persistent storage* on the hard disk. The TSS also supports different mechanisms of *key certification* and the backup to other TPMs in a protocol called *migration*.

The TSS is also specified to enable Identity Management. To prevent privacy issues by correlation of the reuse of the same unique key pair for different services, the unique Endorsement Key cannot be used directly. Instead, *Attestation Identity Keys* (AIKs) are created in a process that involves the TPM and a trusted third party called a PrivacyCA. The TSS is the entity that collects all required information and certificates to assemble the appropriate data structures for communication between the local TPM chip and the remote PrivacyCA service. In [46], we outline this protocol in more detail. The alternative protocol of Direct Anonymous Attestation [9], is based on group signatures. This protocol allows a TPM to proof that it is within a group of TCG-compliant TPMs without revealing which member it is. However, the scheme described through the TPM and TSS specifications has been found to be complex to use and very slow in practical implementations [14].

Data can be encrypted by the TPM mainly using two mechanisms, binding and sealing. *Binding*, which is done in software, potentially even on a remote host, is the encryption of a limited amount of data with a public RSA key using either PKCS #1 version 1.5 [30] or OAEP [6] paddings. If the corresponding private key is unique and held by a TPM this implies that only this TPM can decrypt the data. In an even stronger mechanism called *sealing*, the encryption is performed on-chip incorporating a unique secret and a set of PCRs. Sealed data can only be unsealed by precisely the same TPM in the desired PCR configuration.

The TSS also provides interfaces to *sign* user data using TPM-protected keys, which were generated with type information that allows them to perform RSA-signature operations. AIKs are even more restricted and can only be used to sign TPM internal data structures. The most prominent example is the *Quote* operation where a set of PCRs is signed to report the current platform state. Further useful interfaces of the TSS provide access to the random number generator, a tick counter that can potentially be correlated with real-time and a monotonic counter mechanism.

From a software engineering perspective, the TSS specification follows a layered architecture shown in Fig. 1.1. Just below the TSS, and not part of it, is the TPM driver. The TPM driver can be either vendor specific or follow the TPM Interface Specification (TIS) standard [74]. It is the task of the lowest layer of the TSS to abstract this driver and expose an OS and vendor independent set of functions that allows basic interaction with the TPM. This lowest layer is called the *Trusted Device Driver Library* (TDDL). The TDDL serves as a single-instance, single-threaded component and allows for sending commands as byte streams to the TPM and receiving the responses.

The next layer, the TSS Core Services (TCS), should be implemented as a singleton system service or daemon. It is the single instance that manages the

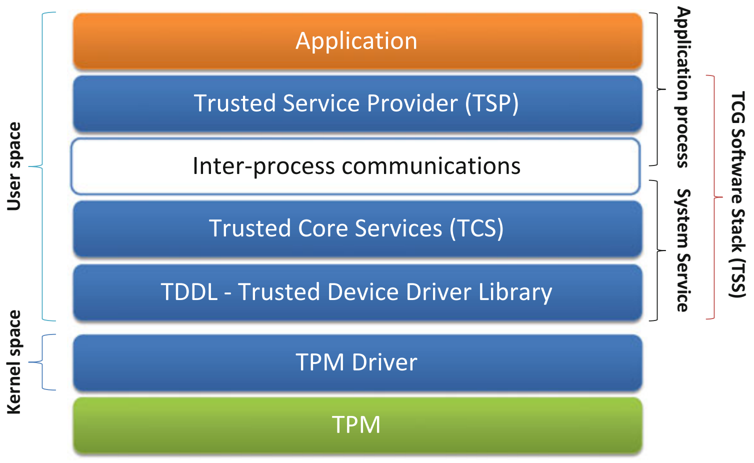
Application
Trusted Service Provider (TSP)
Inter-process communications
Trusted Core Services (TCS)
TDDL - Trusted Device Driver Library
TPM Driver
TPM
User space
Kernel space
Application process
System Service
TCG Software Stack (TSS)

Fig. 1.1 The TCG software stack (TSS) architecture consists of several software layers within a trusted platform

TPM's resources and accesses it. It generates synchronized command streams from concurrent API commands to be transferred through the TDDL. The TCS takes care of the management of TPM key slots as well as permanent storage of TPM key material. Keys are assigned a Universally Unique Identifier (UUID) [34] that is used to identify stored keys. The TCS also maintains the SML where all PCR extend operations are recorded.

The upper layers of the software stack may access the TCS via inter-process communications according to the platform-independent Simple Object Access Protocol (SOAP) [78] interface.

The highest layer, the TSS Service Provider (TSP) provides Trusted Computing services to applications in the form a shared library. The TSP interface is defined as grouped function signatures and data structures in the C programming language. The TSS was also designed to allow partial integration with existing high-level APIs, such as PKCS #11 [52]. This enables the use of the cryptographic primitives provided by the TPM by legacy software. A limitation of this approach is that these legacy cryptographic APIs do not account for advanced Trusted Computing concepts such as sealing. Also the TCG's key typing and padding policies need to be considered [12] and might not match all application areas.

1.3.1.1 TSS in Embedded Systems

On Desktop systems, recent years have seen successful integration of generic TPM 1.2 hardware drivers into major operating systems, i.e. not only Windows, but also Unix derivatives and Linux. Several implementations of TSS exist. One noteworthy open source implementation by IBM is TrouSerS [25].

As we will discuss later in Sect. 1.6.2, there are different sizes and classes of embedded systems. Typically, however, only selected components of the TSS architecture are used. For instance on the TDDL level, drivers are usually derivatives of the Linux implementations. Which driver to use depends on the hardware interface of the TPM.

When drivers have established basic connectivity, higher layers may access the TPM. In many specific use cases, it is sufficient to implement only a small number of instructions, such as writing into a PCR register. This can be achieved by assembling the appropriate instruction byte arrays manually in programs [32] or by including command line tools. This keeps the software overhead on embedded platforms small; however, more complex instructions, for instance creating or certifying fresh keys are a tedious task to write, and most implementors will need a full-scale TSS to achieve such functionality. Very small platforms will note able to support this. One possibly solution can be to perform complex initialization and commissioning tasks on the TPM chip at manufacturing time, before even soldering it onto the target platform [79].

On medium-sized embedded systems, running a full TSS becomes feasibly, although the required XML and cryptographic features require a certain amount of system resources.

1.3.2 Review of Existing Java Libraries

This section presents an overview of existing libraries and APIs that provide first, experimental support for Trusted Computing to Java developers. Additionally, strengths and weaknesses of the individual approaches are discussed.

1.3.2.1 Trusted Computing for the Java Platform and jTSS

The central component of the open source "Trusted Computing for the Java Platform" project is an implementation of the Trusted Software Stack (TSS) for Java programs called jTSS [47]. It is a large library that provides Java programs with the TSS functionality that C programs currently enjoy.

Overall, the project offers two flavors of TSS implementations.

jTSS Wrapper Provides Java programs access to C-based stacks through an object-oriented API, which forwards calls to the native TSS. A thin C-back-end integrates the TSP system library. The Java Native Interface (JNI) maps the functions of the C based TSP into a Java front-end. There, several aspects of the underlying library, such as memory management, the conversion of error codes to exceptions and data-type abstractions, are handled. This wrapping approach results in complex component interactions. Unfortunately, debugging across language barriers is a challenging task. Another drawback is that implementation errors in the C-based components may seriously affect JVM stability.

jTSS Is a native implementation of the TCG Software Stack written completely in the Java language. It offers seamless support for Linux operating systems and all Windows versions starting with Vista, demonstrating platform independence. The Java TCS also synchronizes access from multiple Java applications. Such a full Java TSS implementation clearly reduces the number of involved components and dependencies. Consequently, this approach results in fewer side-effects from incompatible TSS implementations or different interpretations of the TSS specification. Moreover, a pure Java stack can easily be ported to other operating systems and platforms.

The API exposed by both variants is the same, enabling Java application programmers to switch between the two seamlessly, with the choice of the back-end implementation depending on the surrounding platform. It defines data types, exceptions and abstract methods – we refer to it as the *jTSS API* [72]. It closely follows the original TSS C interface, permitting the user to stay close to the originally-intended command flows and providing the complete feature set of the underlying library. jTSS covers almost all of the functions specified by the TCG for communicating with the TPM at the fine granularity of TSS commands. As with every TSS, a complex sequence of commands is required to achieve functionality such as sealing, binding, attestation and key generation for application software. The project also provides `jTpmTools` (jTT), which is a collection of useful sample programs. They are implemented using jTSS API and demonstrate its usage.

The libraries were first created in the course of the Open_TC [42] research project. The first release was authored by Winkler, while the subsequent releases, maintenance and support activities have been done by Toegl. jTSS has since become a popular choice for Trusted Computing related research activities [2–5,7,8,11,13,15–17,22,24,27,29,31,33,37,38,44–46,55,59,63,66,67,69,70,81–84] and even found (quite) limited industrial use. It is one of the most widely used, supported and regularly updated TSSes available today.

While jTSS does allow the use of the TPM from Java, it still involves a significant learning curve for the average Java programmer, who may not be familiar with the procedural programming style that stems from the C-based TSS legacy. Overall, it is still a complicated API that requires a large amount of training and cross-language experience before it can be used in real-world projects.

1.3.2.2 TPM/J

Sarmenta et al. presented TPM/J [53,54], a high-level API that allows Java applications to communicate with the TPM. It is compatible with the Linux, Windows XP and Windows Vista and Mac OS X[1] operating systems thus living up to the promise of platform independence.

[1]At the time of writing, the inclusion of TPMs in Mac OS X compatible platforms has been discontinued.

Strictly speaking, TPM/J is not a TSS since it intentionally deviates from the specifications of TCG's TSS, which seems natural since the TSS specifications provide details specific to the structural programming paradigm and cannot be ported to the object-oriented perspective without major changes to the specifications. A drawback is that the library does not feature a split design as the TSS. Therefore, the JVM must run with elevated privileges to access the TPM hardware resource. Moreover, a major concern for users of TPM/J is that it is not regularly maintained, thus making it unsuitable for large-scale adoption in the community. It has however been used to study monotonic counters [53], and an attack on the TPM [1].

1.3.2.3 TPM4JAVA

TPM4JAVA [23] is a Java library that provides an easy-to-use API to Java programmers for communicating with the TPM. Its design is based on three levels of abstractions:

1. High-level: It provides developers with conveniently usable functionality to execute selected commands such as taking ownership, computing hashes and generate random numbers.
2. Low-level: This is a less user-friendly approach that allows programmers to execute any of the commands supported by the TPM.
3. Back-end: It is used internally for communicating with the TPM device driver library.

While the high-level API makes several functions easily accessible, some operations, such as performing a quote during attestation, require several lines of code and a low-level understanding of the actual functioning of the TPM. This makes 'high-level' a misnomer. The project has not been maintained for several years. Finally, TPM4JAVA shares the limitation of the other approaches of not adhering to the TCG's specifications.

1.3.3 Other Proposed Higher Level Interfaces

From a developer's point of view, the highly complex TSS design suffers from several drawbacks. It is challenging to develop applications with it, as even straightforward mechanisms of the TPM correspond to complicated instruction flows in the TSS API. Implementations of the various TSS layers themselves are often difficult to maintain and suffer from a high risk of introducing security-critical errors. A lot of functionality specified in the API is not relevant for many typical use cases of Trusted Computing. This is especially true for heterogeneous environments or embedded platforms. Based on these insights, individual proposals for higher level interfaces have recently been made for non-Java environments.

Stüble and Zaerin [60] propose a simplified trusted software stack (μTSS) for the C++ language. It mimics the TCG layered architecture (in the form of object-oriented oTDDL, oTCS and oTSP layers), with oTSP providing high level abstractions of selected functionality. It is noteworthy that oTCS offers access to all TPM instructions. Currently, μTSS does not separate user and system processes for device access and does not offer automatic TPM resource management to applications.

Also for C++, Cabbidu et al. [10] present the Trusted Platform Agent, a library that aggregates TSS functions into a higher-level API and also integrates other features missing in the language's standard library like cryptography and network communications. It therefore provides selected building blocks for trusted applications that can be applied with a low learning curve.

Reiter et al. [51] describe an alternative stack design that integrates in an open source cryptography API for Microsoft .NET.

Beneath the TCS layer of the TSS, the TPM Base Services (TBS) [39] in Windows Vista or later, virtualize the TPM for concurrent access and also offer a small set of management features to scripting languages.

1.3.4 Findings

While the aforementioned APIs all share the common goal of providing Trusted Computing functionality to Java developers, to date none of them has seen widespread adoption beyond research and academia.

One of the main reasons is that the interfaces exposed by the libraries often are difficult to learn and understand. This stems from several facts:

1. Trusted Computing by itself is a complex technology. The specifications defining the two major components – the TPM and the TSS [73, 75] – together consist of about 1,500 pages. The concepts are often not well presented for novice users and details have to be looked up in several different places. So it comes as little surprise that very few actual software products make use of the original C-based TSS to access the TPM. Indeed, a 2008 study [57] on the TSS concludes, that, "*it is apparent that, until now, no application exists that makes use of this technology. Even the simplest applications [...] have not been applied yet.*"
2. Implementations like jTSS try to mimic the interface defined by the TSS specification. This interface, however, was developed for procedural programming languages like C. Even though jTSS tries to map the TSS concepts to an object oriented API, it still does not fit well into the Java ecosystem and feels unnatural to developers familiar with other Java APIs. Furthermore, large and complex amounts of code are required to set up and perform basic Trusted Computing functions. This stems from the fact that in the original C-based TSS API, functions take long lists of parameters with many potentially illegal combinations. This makes the API error prone and complex to use for developers without detailed Trusted Computing knowledge.

3. Implementations like TPM/J and TPM4JAVA provide alternative interfaces to Trusted Computing functionality. While in the first case, the interface is at a very low level, the second one offers some higher level abstraction, but is neither consistent nor complete.
4. A full-fledged TSS is a very flexible and powerful library, but practical experience has shown that its full capabilities are not actually required for the vast majority of typical Trusted Computing applications. From our long experience of maintaining jTSS and supporting its users stems the insight that most adopters only follow existing code examples and test code. Few experiments create previously-not-demonstrated functionality, which requires a very steep learning curve.

Therefore, a novel design is needed that improves on the identified shortcomings and provides a programming interface suitable for Java developers while considering the specifics of trusted hardware platforms, legacy software architectures, and, obviously, the Java environment in different deployment scenarios, conventional or embedded.

1.4 API Design

We can now move on to describe the major influences on our specification and our resulting design decisions. Based on defined goals and clear assumptions on the developers that we target, we consider the restrictions imposed by the surrounding environment and discuss how the standardization process has influenced our proposal. From these constrains, we have implemented an agile specification process which enables us to derive the design of the JSR 321 API.

The Java Community Process (JCP) [28] aims to produce specifications using an inclusive, consensus-based approach. The specifications are, throughout their creation and also after release called "Java Specification Request" and assigned a running number. It is controlled by an elected *Executive Committee* (EC), which represents most major players in the Java industry. The central element of the process is to gather a group of industry experts who have a deep understanding of the technology in question and then have a technical lead work with that group to create a first draft. Consensus on the form and content of the draft is then built using an iterative review process that allows an ever-widening audience to review and comment on the document. While the JCP provides a formal framework with different phases and deliverables, an *Expert Group* (EG) may freely decide on its working style.

There are a number of phases in the process. At first, a new specification is *initiated* by a community member and approved for development by the EC of the JCP. Then, a group of experts is formed to develop a preliminary draft of the specification. Feedback from *early reviews* is used to revise and refine the draft. Once considered complete, the draft goes out again for *public review*. Now, the

EC decides if the draft should proceed. If approved by the EC, a proposed final draft of the specification is published and the leader of the expert group sees that the reference implementation and its associated technology compatibility kit are completed. Then the EC will decide on its *final approval*. Completed specifications will be *maintained* and updated.

The process also requires a *reference implementation* (RI). Its purpose is to show that the specified API can be implemented and is indeed viable. With the *Technology Compatibility Kit* (TCK) a suite of tests, tools, and documentation that is used to test implementations for compliance with the specification has to be provided as well.

1.4.1 Goals for a Novel API

From the previously outlined study of different existing Trusted Computing libraries we conclude that none of the proposals fulfills the desirable features an industry specification for applied usage also outside of the academic niche should have. We therefore propose a new API and present a set of goals the Expert Group has decided on.

Integration with Trusted Computing Platforms: As a software interface, the API should be oblivious to the actual hardware it is running on and should not introduce additional limitations on hardware resources. To the OS, the Java Virtual Machine appears just as an ordinary application. Therefore, the TPM access mechanisms need to integrate [69] with the surrounding environment of the OS, be it virtualized or not, and management services.

User-Centric Design. An Application Programming Interface is directed towards the programmer. An Trusted Computing API should therefore be designed to aid developers in writing security applications under the assumption of defined knowledge in the field of application.

Simplified Interface. To make the new API fit into the Java ecosystem, a completely new and fully object-oriented interface is to be designed. For instance, generic entities (e.g., cryptographic keys) in the TSS should be replaced with specific classes that represent the different types (e.g., a dedicated class for each type of key). This allows the set of offered operations to be limited to those actually applicable for a certain object type, thus enhancing usability and reducing the risk of errors.

Reduced Overhead. The TSS API requires a substantial amount of boilerplate code for routine tasks, such as key creation, data encryption or password management. The proposed API should attempt to replace these lengthy code fragments with simple calls using sensible default parameters where required.

Conceptual Consistency. Names in the API should be consistent not only within the API but also with the nomenclature used by the TCG and in Trusted Computing literature. This will allow users to easily switch from other environments to the proposed API. Still, naming conventions of Java must be adhered to.

Testable and Implementable Specifications. The API design should target a small core set of functionality, based on the essential use cases of Trusted Computing. This restriction in size will allow for complete implementations and functional testing thereof. Also, limitations of scope make it possible for all implementations to cover the full proposed API, a key requisite for true platform independence.

Extendibility. The API should allow implementers and vendors to add functionality which is optional or dependent on the capabilities of the surrounding platform.

Standards Compliance. Having an industry-wide standard of accessing the TPM from software is indispensable for widespread use and for enabling code mobility. As the TSS API has shown itself to be unfit for Java environments, the newly proposed API should itself be based on a novel, independent industry standard.

1.4.2 Expected Developer Knowledge

A major goal of the proposed JSR 321 API is to simplify Trusted Computing and make it accessible to a larger group of software developers. To achieve this, it is essential to understand our target audience and their skills before we can move on to create a programming interface for them. In the following we define which skills and knowledge we expect of a developer in order to make full use of the API.

In general, a developer using JSR 321 should be familiar with the cryptographic mechanisms provided in the Java Security Architecture [19, 36]. The concepts of data encryption, decryption and the creation of message digests using hash algorithms should be familiar. The algorithms in particular include SHA-1 and RSA as used by current TPM implementations. Moreover, a general understanding of Trusted Computing concepts and the functionality provided by a TPM is required. This at least should include knowledge about the following topics:

TPM Life-cycle. Starting with its manufacture, a TPM goes through a number of different states. A developer must understand this life-cycle, for instance that the TPM is shipped in an unowned state and its owner must explicitly take ownership, activate, and enable it. When the machine containing the TPM reaches its end of life, the TPM may be cleared to ensure that any TPM protected data can no longer be accessed. To avoid data loss, appropriate mechanisms like key backup or migration must be executed beforehand. Also the implications of a transfer of ownership of a platform need to be considered.

TPM Key Management. A TPM supports a range of different key types, including storage, binding and signature keys. The developer is responsible for building and maintaining a consistent hierarchy. For instance, if certain keys are created as non-migratable this may rule out any backup of them.

Root and Chain of Trust. Ideally a consistent chain of trust would be established by the operating system. However, today's mainstream platforms fail to do so. Developers need to take extra care to consider the security level represented by the PCR values.

Trusted Storage. Care must be taken when the binding and especially sealing mechanisms are applied to data or user supplied key material. Again, the problem of backup arises, especially considering state changes which can render sealed data permanently unaccessible.

Attestation. A number of different protocols have been proposed to perform attestation to a remote verifier [62]. This API supplies the means to create TPM quotes. Since there is no general standard for attestation and public key infrastructures remain limited, we leave the full specification and implementation of communication protocols to the application developer.

1.4.3 API Scope Considerations

JSR 321 aims to be a simplified, compact and user-friendly API, that should integrate in the complex ecosystem of today's Trusted Computing infrastructures. It it therefore imperative to clearly focus the functions offered by the interface so that the resulting design is consistent and viable.

A natural starting point would be to derive a Trusted Computing API from the complete TSS specifications. However, JSR 321 is not planned to fully replace the TSS in all its tasks. Instead, and as required by the nature of the JVM as a user process, it builds on and extends the TSS services offered by the operating system environment.

As a deliberate design decision, JSR 321 will provide functionality focused on applications and middle-ware, rather than providing support for the low level BIOS or OS features of the TPM. This restriction matches the field of use of Java and permits a significant reduction in complexity. JSR 321 will not duplicate elements of the Java Cryptography Architecture, thus fitting into the existing library framework.

Finally, many TSS-specified functions are simply not needed in Java APIs. Management of memory and other resources can and should be hidden from application developers. Object initialization and destruction are natural features of object-oriented languages. Cryptographic primitives like hash functions are already well-supported in the Java Cryptography Extension.

To derive the functional scope of the API, the commented complete list of TCG-specified TSP functions [12] was considered. Based on the criteria and principles laid out previously, those features were selected that are required for core use cases that have high importance for practical applications.

In summary, the design focuses on the most important *core concepts of Trusted Computing*. The second main goal is to provide *high usability*. At the same time, the API is designed to remain modular enough to be extensible with future developments.

1.5 Outline of the API

The unique name-space officially assigned to the JSR 321 API is `javax.trusted computing`. Within this name-space, a number of packages has been specified, each representing a well defined set of functionality in several classes. The relationship between the packages and classes is outlined in Fig. 1.2. These packages comprise the API:

`javax.trustedcomputing.tpm` This package contains all relevant functionality for connecting to a TPM. Within the TPM hardware chip, the concept of a *context* allows the separation of objects such as keys between different users and different connection sessions. In JSR 321, a TPM connection is represented by the central `TPMContext` object that acts as a factory for other objects specified by the API such as the `KeyManager` or the `Sealer`. The `TPM` interface is also defined in this package, which provides general TPM related information such as its version and manufacturer. Additionally, it allows PCR registers to be read and extended, as well as providing the `Quote` operation required for platform attestation.

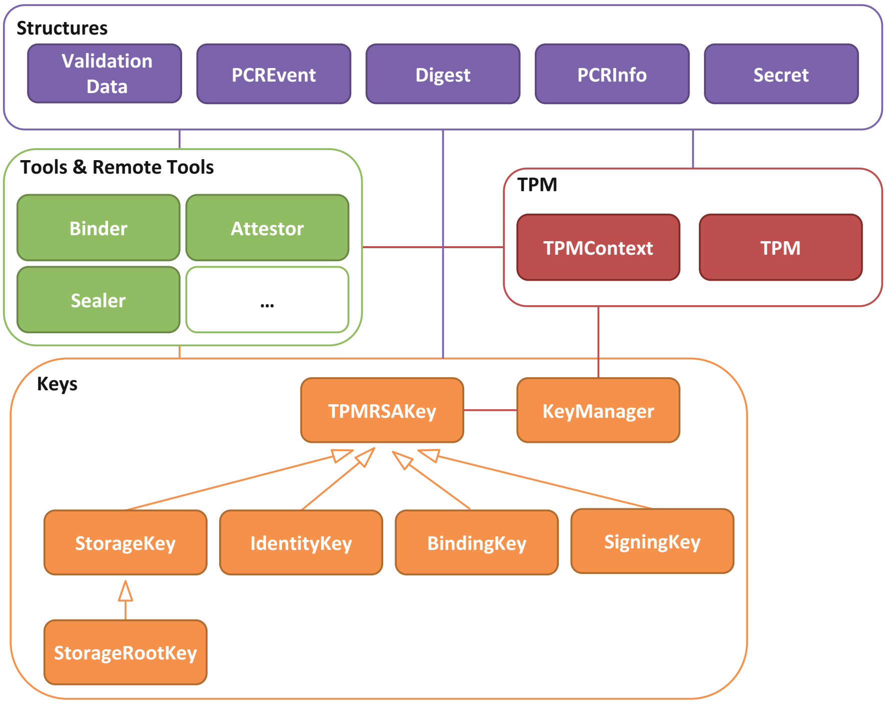

Fig. 1.2 Illustration of the relationship between the core components, including the TPMContext, KeyManager, and Key classes and the Tools

`javax.trustedcomputing.tpm.keys` Contrary to the TSS specification, JSR 321 introduces specific interfaces for the individual key types supported by the TPM. This includes interfaces for storage, sealing and binding keys. Compared to having one generic key object, this approach reduces ambiguities in the API and allows appropriate key usage to be enforced at the interface level. Using strong key types also relates well to results in formal API design and analysis research.

`javax.trustedcomputing.tpm.structures` This package holds data structures required for certain TPM operations. They include the `PCREvent` structure required for operations on the measurement log, `PCRInfo` used as part of platform attestation and `ValidationData` as returned by the TPM quote operation.

`javax.trustedcomputing.tpm.tools` In this package, there are interface definitions for helpers classes to perform TPM operations such as binding, sealing, signing and time stamping. The `javax.trustedcomputing.tpm.tools.remote` sub-package offers abstract classes that allow a remote host without TPM to participate in Trusted Computing protocols. It provides the functionality to validate and verify signatures on TC data types.

For error handling, a single `TrustedComputingException` covers all lower layers. It offers the original TPM/TSS error codes, but also a human readable text representation, which is a great step forward in terms of usability. Despite using only a single exception class, implementations of the API should forward as much error information as possible. For illegal inputs to the JSR 321 API, default Java run-time exceptions are used. Finally, functions offering bit-wise access to status and capability flags are replaced by specific boolean methods that allow access to application relevant flags.

In JSR 321, the `KeyManager` interface defines methods for creating new TPM keys. Upon creation, a secret for key usage and an optional secret for key migration have to be specified. After a key is created, the `KeyManager` allows the key, encrypted by its parent, to be stored in non-volatile storage. As required, the `KeyManager` allows keys to be reloaded into the TPM, provided that the key chain up to the storage root key has been established (i.e., each parent key is already loaded into the TPM). Every time a new key is created or loaded from permanent storage, a usage secret has to be provided. This secret is represented by an instance of a dedicated class `Secret` that is attached to the key object upon construction. `Secret` also encapsulates and handles details such as string encoding, which are often a source of incompatibility between different TPM-based applications.

The extendable `tools` package implements various core concepts of Trusted Computing. As each tool that accesses the TPM is already linked to a `TPMContext` at creation, there are few or no configuration settings required before using the tool. Each tool provides a small group of methods that offer closed functionality. For example, a `Binder` allows the caller to `bind` data under a `BindingKey` and a `Secret`, and returns the encrypted byte array. Usage complexity is minimal as no further parameters need to be configured and the call to `unbind` encrypted

```
try {

    TPMContext context = TPMContext.getInstance();
    context.connect(null);

    KeyManager keyManager = context.getKeyManager();

    StorageRootKey srk = keyManager
            .loadStorageRootKey(Secret.WELL_KNOWN_SECRET);

    Binder binder = context.getBinder();

    Secret keyUsageSecret = context.getSecret("Passphrase for
        using the key.".toCharArray());

    BindingKey bindingKey = keyManager.createBindingKey(srk,
            keyUsageSecret, null, false, true, true, 2048, null);

    byte[] plainData = new String("Data to be encrypted and bound.
        ").getBytes();

    byte[] boundData = binder.bind(plainData,
                    bindingKey.getPublicKey());

    context.close();

} catch (Exception e) {
    // Handle errors..
}
```

Fig. 1.3 Example of JSR 321 code that performs binding of data

data is completely symmetric. Besides the core set of tools (`Signer`, `Binder`, `Sealer`, `Attestor`, `Certifier`, `Signer`), implementers of JSR 321 may add further sets of functionality. An example might be the tool `Initializer` which manages TPM ownership, if the Java library is implemented on an OS without tools for doing so.

In Fig. 1.3 we list source code that demonstrates the API. The example shows Java code that first opens a TPM context session, creates a non-migratable cryptographic key with the following key policy: the key is a child of the Storage Root Key, its usage authenticated with keyUsageSecret; there is no migration secret set as the key is non-migratable; it's volatile, requires authentication, is a 2,048 bit RSA key and is not restricted to a PCR configuration. Finally, the program binds data to the platform where the code is executed. This example also allows us to evaluate the expressiveness and complexity of writing code. In [61], Stüble and Zaerin use the number of Lines of Code (LOC) of code examples as measure to compare different Trusted Computing APIs. They compare implementations of the identical binding use case. According to them, achieving the same functionality requires 146 LOC with TSS, 30 with jTSS and 18 using μTSS. The JSR 321 program we present takes only 15 LOC. Besides this obvious reduction of code size, the naming conventions used throughout the API allow the effective use of code-completion mechanisms found in modern Integrated Development Environments (IDE) such as Eclipse. In many cases, the IDE will automatically suggest a suitable parameter for method calls, thus considerably speeding up the development of trusted computing applications.

1.6 Experience and Outlook

1.6.1 Third Party Implementation and Teaching Experience

Our API design has already been adopted by a third party, indicating the viability of the JSR 321 approach. To improve the security of smart power meters, the TECOM research project [64] has independently created a JSR 321 implementation based on the Early Draft version of the specification in order to satisfy their need for a high-level Trusted Computing API for Java-based embedded systems. Their implementation was built on top of the previously described ţ TSS in C++ and Java. The feedback [56] from this external implementation effort has been very positive and helpful. Aside from minor ambiguities in the specification and small feature requests, no major difficulties were reported. TECOM concluded that JSR 321 "provides most functionality that the majority of users would probably need" and that the single interface layer and low level of background knowledge required gave it an advantage over other APIs.

In summer 2010, JSR 321 was used for teaching the 5th European Trusted Infrastructure Summer School (ETISS), at Royal Holloway, University of London. In the 90-min "TPM Lab" we provided an introduction to the central component of Trusted Computing, the Trusted Platform Module (TPM). The lab explained TPM activation control, basic operations, and high-level programming of the TPM with JSR321. The concept of the chain-of-trust was explored in a practical sealing experiment. On the available HP desktop machines with Infineon TPMs we provided pre-configured USB-memory sticks running Ubuntu Linux and Eclipse as development environment. Although many of the participants had either no experience with TPMs or with Java, about two thirds of them were able to complete the implementation of an unsealing program within less than 1 h. This clearly underlines the low initial threshold for using the JSR 321 API. JSR 321 was also used in two practicals for courses taught at Graz University of Technology. In the third year bachelor-class "Security Aspects in Software Development" of 2011, students were given the choice of implementing the signature component of a CA service either through JavaCard or TPM interfaces. The groups that chose JSR 321 did not require more supervision or advise than those choosing the more conventional technology. In the master-level "Selected Topics IT Security 1" class, in the summer term 2012 students were, among others, given the task to demonstrate remote attestation of an Android environment. To this end, they employed a software emulated TPM and accessed it through JSR231. As the Android environment is source compatible with Java, this caused no additional complexity.

Thus, JSR 321 has proven to be fit for implementation by third parties not involved in the specification. Also, the API can be used in academic teaching and Embedded Systems much like other existing security technologies.

1.6.2 Application in Embedded Systems

In this section we study whether it is (performance-wise) feasible to use Java as a runtime environment for Trusted Computing related applications in Embedded Systems. This section briefly describes the used platforms and the corresponding software configurations.

When considering trusted embedded platforms we have to distinguish two classes of platforms: first, PC like platforms including many "industrial PC" motherboards, typically based on $\times 86$ compatible processors. They have a PC-style Southbridge controller which exposes an LPC bus interface. On this class of trusted embedded platforms the TPM is connected to the system using the standard LPC bus interface. Trusted embedded platforms in this category can be treated exactly in the same manner as desktop PC platforms without any loss of generality. The following sections focus on a second class of trusted embedded platforms, which do not provide a standard LPC bus interface and thus have to resort to alternative methods for connecting to a TPM. We concentrate our discussion on TPMs connected via an I^2C bus. This bus is supported by virtually any embedded microprocessor or micro-controller of interest, either through dedicated hardware blocks or through software-emulation using general purpose input/output pins.

The inter-IC (I^2C) bus [41, 80] was introduced by Phillips Semiconductors (now NXP Semiconductors) in 1982 to provide simple bidirectional 2-wire bus for communication between micro-controllers and other integrated circuits (ICs). Data transfers are 8-bit oriented and can reach speeds of up to 100 kbit/s in standard mode or 400 kbit/s in fast mode. Higher transfer speeds up to 5 Mbit/s can be achieved given that certain hardware constraints are met.

Currently no approved publicly available TCG standard for TPMs with I^2C interface exists, although several vendors have recently started shipping I^2C-enabled TPMs. The TCG's Embedded Systems Working Group has started work on a, currently unreleased, draft for a I^2C TPM interface specification based on the existing PC-centric TPM TIS [74] specification. At the time of this writing no final standard has been publicly available yet.

1.6.2.1 Benchmark Platforms

We chose three different systems to evaluate the performance of Java-implemented Trusted Computing libraries. At the time of performing the analysis, available TPMs were designed for the PC platform only and thus used the *Low Pin Count* (LPC) bus as an interface. To enable Embedded Computing scenarios, we created our own

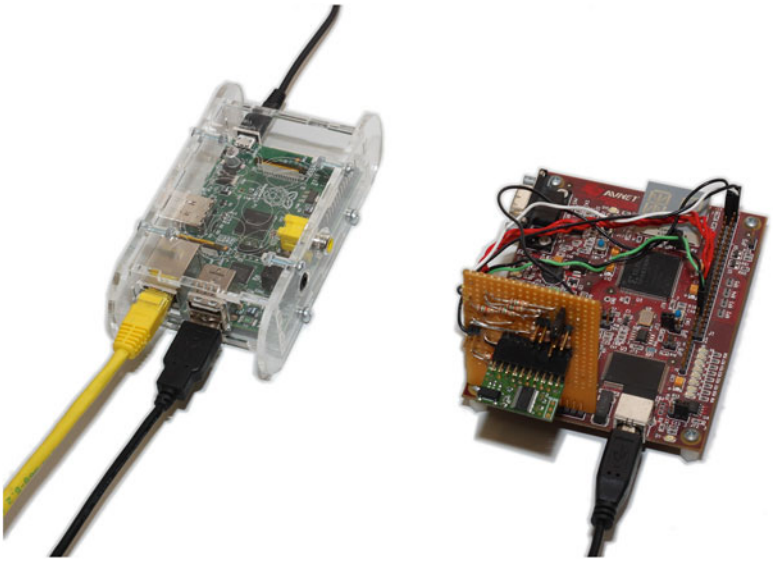

Fig. 1.4 Raspberry Pi with IAIK LPC-over-USB-TPM

adapter board which enables us to connect a LPC TPM to embedded systems. The board is described in more detail in [48] and can be seen in Fig. 1.4. Basically it tunnels LPC packets over an USB interface. In addition two kernel modules are built and loaded. They provide a `/dev/tpm0` device by accessing our LPC-over-USB adapter board. The used TPM is an Infineon TPM version 1.2 with firmware 3.17. This adapter in combination with the PC-TPM is used as substitute since TPMs with I^2C interface were not available at the time of the experiments.

To give a hint on how our adapter performs, a simple method was used. We measured the time the kernel module needs to obtain the content of all PCRs. The used command was `time cat /sys/class/misc/tpm0/device/pcrs`. On the HP dc7900 platform it takes 0.402 s with the built-in TPM and the `tpm_tis` kernel module. With the IAIK LPC-over-USB TPM and our kernel modules the command took 1.399 s. On a Raspberry Pi it finished after 1.520 s.

HP dc7900 We use an HP dc7900 desktop PC as a reference platform. It runs Ubuntu 12.04 x86_64 with OpenJDK 64-Bit Server VM,version 1.6.0_24, as provided by Ubuntu package repositories. The CPU is an Intel Pentium Dual-Core E5200. Together with 4 GB RAM this is not the newest generation of desktop PCs but provides a solid base for comparisons.

Freescale i.MX51 EVK The i.MX51 EVK is an evaluation kit for Freescale's i.MX51 system on a chip (*SoC*). The main component of the SoC is an 800 MHz ARM Cortex A8 CPU core. The platform has 512 MB of RAM, whereof about 400 MB are available for the operating system. The used OS is an Ubuntu 10.04

with a self compiled Linux kernel version 2.6.35. The sources and patches for the kernel are provided by Freescale. The Java environment is provided by several different Java Virtual Machines (JVMs):

- OpenJDK Zero VM, version 1.6.0_18, is the standard Java VM of Ubuntu 10.04. It comes without an JIT compiler, so the Java bytecode is just interpreted.
- Java HotSpot™ Embedded Client VM, version 1.6.0_32. This is a original binary release by Oracle, providing a Java SE 6 runtime environment for ARM 6 and 7 platforms. The JIT compiler is of the 'client' flavor.
- Java HotSpot™ Embedded Client VM, version 1.7.0_04, is an updated release based on the Java SE 7 specifications.
- Java HotSpot™ Embedded Server VM 1.7.0, version 1.7.0_04, offers a 'server' flavor (especially with regards to JIT compilation) VM for ARM platforms. In contrast to the 'client' VMs it is only available for ARMv7 processors.
- CACAO, version 1.6.0_25, this version was built from source to examine the performance of an open source VM with an included JIT compiler.

Raspberry Pi The Raspberry Pi is a very affordable computer targeting education and research. It is powered by an 700 MHz ARMv6 core in a Broadcom BCM283 SoC. On top of the SoC are 256 or 512 MB RAM[2] as package on package. Since the board is very popular and a broad user community has built up, there are many available operating system images available. For this survey the following three where used:

- Raspbian 2012-07-15
- Arch Linux 2012-06-13
- Debian 2012-06-18

While *Raspbian* is a Debian clone optimized for the Raspberry Pi, it is not binary compatible with Oracle's optimized JVMs. An overview of the available and considered JVMs follows here:

- OpenJDK Zero VM on Raspbian, version 1.6.0_24.
- OpenJDK Zero VM on Arch, version 1.6.0_22.
- OpenJDK Zero VM on Debian, version 1.6.0_24.
- Java HotSpot™ Embedded Client VM 1.6.0 This is the same JVM as on i.MX51 EVK.
- Java HotSpot™ Embedded Client VM 1.7.0 This is the same JVM as on i.MX51 EVK.

[2]The 256 MB version was used for benchmarking.

Table 1.1 Results for SciMark benchmark. Higher values are better. The `-large` command line option runs the test with larger matrices

Platform	JVM	Default ≈ *MFlops*	`-large` ≈ *MFlops*
dc7900	OpenJDK	845.45	500.32
i.MX51	OpenJDK	10.21	8.77
	Java6	30.34	21.02
	Java7client	30.05	21.13
	Java7server	31.35	24.78
	CACAO	24.83	17.53
Raspbian	OpenJDK	2.87	2.45
Arch Pi	OpenJDK	1.52	1.50
	Java6	21.66	14.41
	Java7client	21.23	14.13
Debian Pi	OpenJDK	2.86	2.43
	Java6	21.94	14.33
	Java7client	21.79	14.15

1.6.2.2 Benchmarks and Results

To initially assess the performance of generic Java software on the presented platforms, we used the SciMark benchmark suite. SciMark[49] is a rather simple benchmark examining the Java performance for scientific computations. It is provided by NIST and is chosen because its run time is not very long and it gives a brief insight in computational power of a Java environment. We use version 2.0a of the benchmark. The results for the SciMark benchmark are shown in Table 1.1.

The second benchmark was created to obtain performance data specifically relevant for Trusted Computing applications. As a base for the benchmark `jTpmTools` was used. The tasks performed in the benchmark are the following:

- `tpm_version`
  Query the TPM's version information
- `tpm_flags`
  Query the TPM's flag settings
- `pcr_read`
  Read the content of all PCRs
- `pcr_extend`
  Extend a small chunk of data to a PCR (20 byte)
- `pcr_extend_big`
  Extend a huge chunk of data to a PCR (40 MB)
- Sealing
  Seal and unseal 512 bit of zeros. This test consists of three tasks:
  – `create_key`

  Create the needed key
  - `seal`
    Seal the data
  - `unseal`
    Unseal the data

- Quoting
  Obtain a quote from the TPM. This test consists of two tasks:
  - `create_key_legacy`
    Create the needed key
  - `quote`
    Obtain quote
- `stress_block_1` to `stress_block_10`
  Each block queries the TPM's flags 200 times, totaling in a count of 2,000 queries.

  The resulting measurements are given in Table 1.2 and Fig. 1.5.

Table 1.2 Selected results of the Java-based tools benchmark. All values are ms, obtained with local bindings and file system storage. On dc7900 platform OpenJDK is used, on i.MX51 Java6 and on Raspberry Pi Java7client

Benchmark	dc7900	i.MX51	Raspberry Pi
tpm_version	143	161	158
tpm_flags	203	221	220
pcr_read	1,283	1,592	1,591
pcr_extend	278	284	555
pcr_extend_big	1,018	9,070	19,715
create_key	45,818	15,638	2,626
Seal	4,800	4,509	4,604
Unseal	3,239	2,389	2,418
create_key_legacy	18,170	15,135	2,243
Quote	6,000	5,829	6,000
stress_block_1	13,779	13,020	13,020
stress_block_2	13,511	13,019	13,058
stress_block_3	13,503	13,019	13,018
stress_block_4	13,547	13,019	13,028
stress_block_5	13,443	12,999	13,018
stress_block_6	13,695	13,019	13,048
stress_block_7	13,511	14,439	13,228
stress_block_8	13,635	14,999	13,028
stress_block_9	13,439	14,999	13,538
stress_block_10	13,587	14,999	13,898

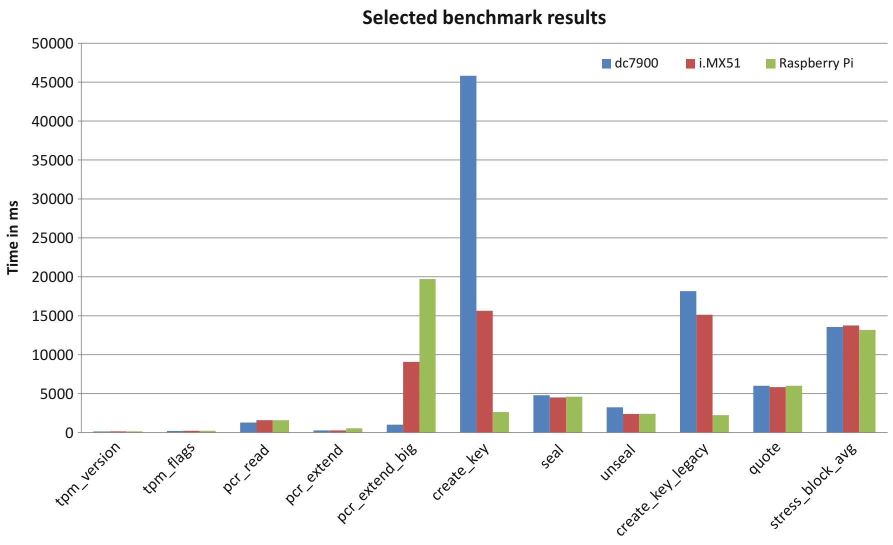

Fig. 1.5 Visualization of results presented in Table 1.2. `stress_block_avg` is the mean value of the 10 runs

1.6.2.3 Performance Discussion

The computational power of the investigated systems is by magnitudes lower than that of standard desktop PCs. Yet, the main insight is that it is possible to use a Java based trusted software stack also on limited platforms such as the i.MX51 EVK board and the Raspberry Pi. The results presented show that the performance difference between a desktop PC platform and embedded systems is minimal for Trusted Computing related tasks. This is due to the fact that TPMs are rather slow devices, so that even the medium-sized Embedded Systems we considered for our benchmarks need to wait for the results of the hardware security component.

Note that the values for `create_key` and `create_key_legacy` might be misleading. The creation of keys depends on entropy generated inside the TPM. The true random number generator's entropy pool is depleted after a few generated keys so any further creation has to wait for new randomness to be available. This leads to high deviations of the values. We show the minimum runtimes measured.

1.6.3 Compatibility with Next Generation TPMs

Throughout the standardization process, 1.2 was the latest version of the TPM specifications [75] officially released and therefore was used as basis for the design of JSR 321. We believe that our basic approach of providing a high-level abstraction of core concepts of Trusted Computing will remain valid for future versions of the specification. Any necessary changes to the API, which could become opportune by the design of the TPM 2.0, which has recently been made available in a preliminary version [76], can be supported by releasing a new revision through the Maintenance phase of the Java Community Process. In addition, we believe that the results of the JSR 321 specification could itself serve to help guide the specification of the next generation of TSS. The authors have received encouraging feedback from the TCG that a high-level Trusted Computing API approach as pioneered in the presented work would be highly desirable for specifications in other languages such as C.

Summary

In this chapter we outlined the current status of software libraries for TPM access and application-level integration of Trusted Computing. To date, several commercial and some free implementations of the TCG Software Stack have been published with varying levels of completeness and standard compliance.

As a basis for the work presented in this chapter, we reviewed and discussed the state of the art of available Trusted Computing software libraries. For native applications, there are ongoing projects which aim to fully implement the TSS standard, and alternative approaches which intentionally provide a reduced and simplified interface.

In managed run-time environments, Java currently is the primary choice for the implementation of Trusted Computing applications. This is emphasized by the existence of several different libraries and frameworks that have been proposed or prototyped for this language. Our review of existing approaches has uncovered a number of drawbacks including high complexity, inconsistent APIs, limited object-orientation or lack of features.

Despite the availability of libraries and tools, Trusted Computing is not yet widely used and has not found its way into commercial applications [57]. We and other designers of TC APIs [60] attribute this fact primarily to the high complexity of and developer expertise required by existing standards and APIs. We believe that a lower learning curve for the software interfaces can attribute to a more widespread use in the future.

Based on these findings, we specify goals for a novel high-level Java API that aims to overcome these limitations. Specifically, we focus on a simple interface for access to commonly used TPM functionality and define the technical knowledge expected of programmers using it. In contrast to the original TSS design, we propose a fully object-oriented approach that hides low-level details and provides additional guidance for developers by providing solid default configurations. Results from the reference implementations are encouraging and demonstrate the feasibility of the proposed approach.

(continued)

Performance evaluations on Embedded Systems underline the applicability of our approach in different usage scenarios, that need not be restricted to the Desktop and Server world.

The aim of our API design was the release as the official Java standard API for Trusted Computing. Therefore, we have adopted an agile and transparent working style within the Java Community Process. The desirable set of API features has been selected based on open discussions. Feedback received from external reviewers and independent implementers has helped to adapt and extend the design. After two publicly announced reviews and several votes within the Java Community Process, the standard was published [68]. The Reference Implementation, the Technology Compatibility Kit and documentation is available under an open-source license from https://jsr321.java.net/.

We believe that this effort towards an open, simple and consistent programming interface can considerably contribute to the future adoption of Trusted Computing. Even thought the proposed JSR 321 API is designed for the Java programming language, we anticipate that the contribution of this work will not be limited to Java. Due to the clear and lightweight design of the API, implementations in other object-oriented programming languages should be possible with only minor adaptations.

References

1. Ables, K.: An alleged attack on key delegation in the trusted platform module. MSc Advanced Computer Science First Semester Mini-Project, University of Birmingham (2009). http://www.computer-science.birmingham.ac.uk/~mdr/research/papers/pdf/09-ables-3.pdf. Website accessed 15 Nov 2012
2. Alam, M., Zhang, X., Nauman, M., Ali, T.: Behavioral attestation for web services (ba4ws). In: Proceedings of the 2008 ACM Workshop on Secure Web Services, Alexandria, pp. 21–28. ACM (2008). doi:10.1145/1456492.1456496
3. Alsouri, S., Dagdelen, O., Katzenbeisser, S.: Group-based attestation: enhancing privacy and management in remote attestation. In: Acquisti, A., Smith, S., Sadeghi A.R. (eds.) Trust and Trustworthy Computing. Lecture Notes in Computer Science, vol. 6101, pp. 63–77. Springer, Berlin/Heidelberg (2010). http://dx.doi.org/10.1007/978-3-642-13869-0_5
4. Baldwin, A., Dalton, C., Shiu, S., Kostienko, K., Rajpoot, Q.: Providing secure services for a virtual infrastructure. SIGOPS Oper. Syst. Rev. **43**(1), 44–51 (2009). doi:10.1145/1496909.1496919
5. Bangerter, E., Djackov, M., Sadeghi, A.R.: A demonstrative ad hoc attestation system. In: Wu, T.C., Lei, C.L., Rijmen, V., Lee D.T. (eds.) Information Security. Lecture Notes in Computer Science, vol. 5222, pp. 17–30. Springer, Berlin/Heidelberg (2008). http://dx.doi.org/10.1007/978-3-540-85886-7_2
6. Bellare, M., Rogaway, P.: Optimal asymmetric encryption – how to encrypt with RSA. In: Santis A.D. (ed.) Eurocrypt 94 Proceedings, Perugia. Lecture Notes in Computer Science, vol. 950. Springer (1995). http://cseweb.ucsd.edu/~mihir/papers/oaep.html

7. Brett, A., Kuntze, N., Schmidt, A.: Trusted watermarks. In: IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, 2009 (BMSB '09), Bilbao, pp. 1–7 (2009)
8. Brett, A., Leicher, A.: Ethemba trusted host environment mainly based on attestation (2009). http://ethemba.novalyst.de/wordpress/wp-content/uploads/2009/11/ethemba1.pdf. Website accessed 15 Nov 2012
9. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, Washington, DC, pp. 132–145. ACM (2004). doi:http://doi.acm.org/10.1145/1030083.1030103
10. Cabiddu, G., Cesena, E., Sassu, R., Vernizzi, D., Ramunno, G., Lioy, A.: The trusted platform agent. IEEE Softw. **28**, 35–41 (2011). doi:http://doi.ieeecomputersociety.org/10.1109/MS.2010.160
11. Celesti, A., Salici, A., Villari, M., Puliafito, A.: A remote attestation approach for a secure virtual machine migration in federated cloud environments. In: 2011 First International Symposium on Network Cloud Computing and Applications (NCCA), Venice, pp. 99–106 (2011)
12. Challener, D., Yoder, K., Catherman, R., Safford, D., Doorn, L.V.: A Practical Guide to Trusted Computing, 1st edn. IBM Press, Upper Saddle River (2008). ISBN-13: 978-0132398428
13. Coppolino, L., Jäger, M., Kuntze, N., Rieke, R.: A trusted information agent for security information and event management. In: Proceedings of the Seventh International Conference on Systems, Saint Gilles (ICONS 2012). Think MInd (2012)
14. Dietrich, K.: Anonymous client authentication for transport layer security. In: De Decker, B., Schaumüller-Bichl I. (eds.) Communications and Multimedia Security. Lecture Notes in Computer Science, vol. 6109, pp. 268–280. Springer, Berlin/Heidelberg (2010). doi:10.1007/978-3-642-13241-4_24
15. Dietrich, K., Pirker, M., Vejda, T., Toegl, R., Winkler, T., Lipp, P.: A practical approach for establishing trust relationships between remote platforms using trusted computing. In: Barthe, G., Fournet, C. (eds.) Trustworthy Global Computing. Lecture Notes in Computer Science, vol. 4912, pp. 156–168. Springer, Berlin/New York (2008)
16. FABBRI, F.: Progetto e realizzazione di un protocollo di verifica dell'affidabilita' di un terminale remoto (In Italian). Tesi di laurea specialistica, Università di Pisa (2007)
17. Gissing, M., Toegl, R., Pirker, M.: Management of integrity-enforced virtual applications. In: Lee, C., Seigneur, J.M., Park, J.J., Wagner, R.R. (eds.) Secure and Trust Computing, Data Management, and Applications. Communications in Computer and Information Science, vol. 187, pp. 138–145. Springer, Berlin/Heidelberg (2011). http://dx.doi.org/10.1007/978-3-642-22365-5_17
18. Global Industry Analysts Inc.: Embedded Systems: Market Research Report. http://marketpublishers.com/ (2013)
19. Gong, L., Mueller, M., Prafullch, H.: Going beyond the sandbox: an overview of the new security architecture in the java development kit 1.2. In: Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, pp. 103–112 (1997)
20. Google Inc.: Android OS. Available online at: http://www.android.com/ (2013)
21. Gosling, J., Joy, B., Steele, G., Bracha, G., Buckley, A.: The Java Language Specification Java SE 7 Edition. JSR 901 (2011). http://docs.oracle.com/javase/specs/index.html. Website accessed 2 Nov 2012
22. Hein, D.M., Toegl, R., Kraxberger, S.: An autonomous attestation token to secure mobile agents in disaster response. Secur. Commun. Netw. **3**(5), 421–438 (2010). doi:10.1002/sec.196. http://dx.doi.org/10.1002/sec.196
23. Hermanowski, M., Tews, E.: Tpm4java. Currently only available through http://web.archive.org/web/20090510093615/http://tpm4java.datenzone.de/trac (2009). Website accessed 6 Nov 2012
24. Huh, J.H.: Trustworthy logging for virtual organisations. Ph.D. thesis, University of Oxford (2009)
25. IBM Corp.: Trousers – an open-source TCG software stack implementation. http://trousers.sourceforge.net/. Website accessed 30 Oct 2012

26. ISO: ISO/IEC 9899:2011 Information technology – Programming languages – C. International Organization for Standardization, Geneva (2011). http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57853
27. Jang, J., Nepal, S., Zic, J.: A trust enhanced email application using trusted computing. In: Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing, 2009 (UIC-ATC '09), Maiden, pp. 502–507 (2009)
28. Java Community Process: JCP procedures overview. http://jcp.org/en/procedures/overview. For JSR 321, version 2.6 applied. Website accessed 12 Nov 2012
29. Jianhong, Y., Xinguang, P.: Protocol for dynamic component-property attestation in trusted computing. In: 2010 Second International Conference on Networks Security Wireless Communications and Trusted Computing (NSWCTC), Wuhan, vol. 2, pp. 369–372 (2010)
30. Jonsson, J., Kaliski, B.: Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447 (Informational) (2003). http://www.ietf.org/rfc/rfc3447.txt
31. Khattak, Z., Sulaiman, S., Manan, J.: Security, trust and privacy (stp) framework for federated single sign-on environment. In: 2011 International Conference on Information Technology and Multimedia (ICIM), Kuala Lumpur, pp. 1–6 (2011)
32. Kinney, S.: Trusted Platform Module Basics: Using TPM in Embedded Systems, 1st edn. Newnes, Oxford (2006). ISBN 13:978-0-7506-7960-2
33. Korn, R., Kuntze, N., Repp, J.: Performance evaluation in trust enhanced decentralised content distribution networks. In: 2011 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR), Naples, pp. 1–6 (2011)
34. Leach, P., Mealling, M., Salz, R.: A Universally Unique IDentifier (UUID) URN Namespace. RFC 4122 (Proposed Standard) (2005). http://www.ietf.org/rfc/rfc4122.txt
35. Lindholm, T., Yellin, F., Bracha, G., Buckley, A.: The Java Virtual Machine Specification Java SE 7 Edition. JSR 924 (2011). http://docs.oracle.com/javase/specs/index.html. Website accessed 2 Nov 2012
36. Lipp, P., Farmer, J., Bratko, D., Platzer, W., Sterbenz, A.: Sicherheit und Kryptographie in Java (In German). Addison-Wesley, München/Boston (2000). ISBN 3827315670
37. Lyle, J.: Trustworthy services through attestation. Ph.D. thesis, University of Oxford (2009)
38. Lyle, J., Martin, A.: On the feasibility of remote attestation for web services. In: Proceedings of the 2009 International Conference on Computational Science and Engineering, Vancouver, vol. 03, pp. 283–288. IEEE Computer Society (2009). doi:10.1109/CSE.2009.213
39. Microsoft: TPM Base Services. Microsoft Developer Network. http://msdn.microsoft.com/en-us/library/aa446796(VS.85).aspx. Website accessed 30 Oct 2012.
40. Microsoft Developer Network: Overview of the .net framework. http://msdn.microsoft.com/en-us/library/zw4w595w.aspx. Website accessed 1 Nov 2012
41. NXP semiconductors: I2C-Bus Specification and User Manual (2012). Available online at: http://www.nxp.com/documents/user_manual/UM10204.pdf
42. Open_TC Consortium: The Open Trusted Computing Project (Open_TC) (2005–2009). Currently available only through http://web.archive.org/web/20110723233118/http://www.opentc.net/. Archived website accessed 30 Oct 2012.
43. Oracle: About Java (2012). http://www.java.com/en/about/. Website accessed 14 Nov 2012
44. Parno, B., Lorch, J., Douceur, J., Mickens, J., McCune, J.: Memoir: practical state continuity for protected modules. In: 2011 IEEE Symposium on Security and Privacy (SP), Berkeley, pp. 379–394 (2011)
45. Parno, B., McCune, J.M., Perrig, A.: Bootstrapping Trust in Modern Computers. Springer, New York (2011)
46. Pirker, M., Toegl, R., Hein, D., Danner, P.: A PrivacyCA for anonymity and trust. In: Chen, L., Mitchell, C.J., Martin, A. (eds.) Proceedings of the 2nd International Conference on Trusted Computing (TRUST 2009), Oxford. Lecture Notes in Computer Science, vol. 5471, pp. 101–119. Springer, Berlin/Heidelberg (2009)

47. Pirker, M., Toegl, R., Winkler, T., Vejda, T.: Trusted computing for the Java™platform (2009). http://trustedjava.sourceforge.net/. Website accessed 29 Jan 2013
48. Pirker, M., Winter, J., Toegl, R.: Lightweight distributed heterogeneous attested android clouds. In: Katzenbeisser, S., Weippl, E., Camp, L., Volkamer, M., Reiter, M., Zhang, X. (eds.) Trust and Trustworthy Computing. Lecture Notes in Computer Science, vol. 7344, pp. 122–141. Springer, Berlin/Heidelberg (2012). http://dx.doi.org/10.1007/978-3-642-30921-2_8.
49. Pozo, R., Miller, B.: SciMark 2.0(2000). http://math.nist.gov/scimark2/.
50. Ravi, S., Raghunathan, A., Kocher, P., Hattangady, S.: Security in embedded systems: design challenges. ACM Trans. Embed. Comput. Syst. **3**(3), 461–491 (2004). doi:10.1145/1015047.1015049
51. Reiter, A., Neubauer, G., Kapfenberger, M., Winter, J., Dietrich, K.: Seamless integration of trusted computing into standard cryptographic frameworks. In: Proceedings of the Second International Conference on Trusted Systems, Beijing, pp. 1–25. Springer (2011). doi:10.1007/978-3-642-25283-9_1
52. RSA Laboratories: PKCS #11 v2.20: Cryptographic Token Interface Standard. RSA Security Inc. Public-Key Cryptography Standards (PKCS) (2004). ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf Website accessed 29 Jan 2013
53. Sarmenta, L., van Dijk, M., O'Donnell, C., Rhodes, J., Devadas, S.: Virtual monotonic counters and count-limited objects using a TPM without a trusted OS. In: Proceedings of the First ACM Workshop on Scalable Trusted Computing (STC '06), Alexandria, 1-59593-548-7, pp. 27–42. ACM (2006). doi:http://doi.acm.org/10.1145/1179474.1179485
54. Sarmenta, L., Rhodes, J., Müller, T.: TPM/J Java-based API for the trusted platform module (2007). http://projects.csail.mit.edu/tc/tpmj/. Website accessed 30 Oct 2012
55. Schlüter, M.: Realisierung einer mobilen, vertrauenswürdigen GeschÃd'ftsplattform auf Basis von Trusted Computing zur gesicherten Datenerfassung (In German). Master's thesis, Technischen Hochschule Mittelhessen (2012).
56. Schnepp, I., Panenka, S., Richard-Foy, M.: JSR321 feed-back from TECOM-FP7's implementation. Technical report, Atego (2010). Review 2.1
57. Selhorst, M., Stueble, C., Teerkorn, F.: TSS Study. Study on behalf of the german federal office for information security (BSI), Sirrix AG security technologies (2008). http://www.sirrix.com/media/downloads/57653.pdf,download. Website accessed 1 Nov 2012.
58. Shim, R., Mainelli, T., O'Donnell, B., Chute, C., Pulskamp, F., Rau, S.: Worldwide interfaces and technologies embedded in PCs 2010–2014 forecast. Technical report, IDC (2010)
59. Strasser, M., Stamer, H.: A software-based trusted platform module emulator. In: Lipp, P., Sadeghi, A.R., Koch, K.M. (eds.) Trusted Computing – Challenges and Applications. Lecture Notes in Computer Science, vol. 4968, pp. 33–47. Springer, Berlin/Heidelberg (2008). http://dx.doi.org/10.1007/978-3-540-68979-9_3
60. Stueble, C., Zaerin, A.: µTSS – a simplified trusted software stack. In: Proceedings of the 3rd International Conference on Trust and Trustworthy Computing (TRUST 2010), Berlin. Lecture Notes in Computer Science, vol. 6101. Springer (2010)
61. Stueble, C., Zaerin, A.: µTSS – a simplified trusted software stack. Technical report, Sirrix AG (2010)
62. Stumpf, F., Tafreschi, O., Röder, P., Eckert, C.: A robust integrity reporting protocol for remote attestation. In: Proceedings of the Second Workshop on Advances in Trusted Computing (WATC'06 Fall), Tokyo, Japan (2006). http://www.research.ibm.com/trl/projects/watc/FredericStumpfPaper.pdf
63. Tanveer, T., Alam, M., Nauman, M.: Scalable remote attestation with privacy protection. In: Chen, L., Yung, M. (eds.) Trusted Systems. Lecture Notes in Computer Science, vol. 6163, pp. 73–87. Springer, Berlin/Heidelberg (2010). http://dx.doi.org/10.1007/978-3-642-14597-1_5
64. TECOM Consortium: Trusted Embedded Computing project (TECOM) (2008–2010). Currently available only through http://web.archive.org/web/20100625044259/http://www.tecom-project.eu/. Website accessed 9 Nov 2012

65. Petazzoni, T. Opdenacker, M.: Java in embedded linux systems (2009). http://free-electrons.com/doc/embedded_linux_java.pdf
66. Toegl, R.: Tagging the turtle: local attestation for kiosk computing. In: Park, J.H., Chen, H.H., Atiquzzaman, M., Lee, C., Kim, T.H., Yeo, S.S. (eds.) Advances in Information Security and Assurance. Lecture Notes in Computer Science, vol. 5576, pp. 60–69. Springer, Berlin/Heidelberg (2009). doi:http://dx.doi.org/10.1007/978-3-642-02617-1_7
67. Toegl, R., Hutter, M.: An approach to introducing locality in remote attestation using near field communications. J. Supercomput. **55**(2), 207–227 (2011). doi:10.1007/s11227-010-0407-1. http://dx.doi.org/10.1007/s11227-010-0407-1
68. Toegl, R., Lipp, P., Nisewanger, J., Rao, D.D., Winkler, T., Keil, W., Hong, T., Nauman, M., Gungoren, B., Graf, K.M.: JSR321 Trusted Computing API for Java. Java Community Process Specification Final Release http://jcp.org/en/jsr/detail?id=321 (2011). Java Specification Request # 321. Website accessed 31 Oct 2012
69. Toegl, R., Pirker, M.: An ongoing game of tetris: integrating trusted computing in java, block-by-block. In: Gawrock, D., Reimer, H., Sadeghi, A.R., Vishik, C. (eds.) Future of Trust in Computing, pp. 60–67. Vieweg+Teubner, Wiesbaden (2009). http://dx.doi.org/10.1007/978-3-8348-9324-6_7
70. Toegl, R., Pirker, M., Gissing, M.: acTvSM: a dynamic virtualization platform for enforcement of application integrity. In: Chen, L., Yung, M. (eds.) Trusted Systems. Lecture Notes in Computer Science, vol. 6802, pp. 326–345. Springer, Berlin/Heidelberg (2011). http://dx.doi.org/10.1007/978-3-642-25283-9_22
71. Toegl, R., Winkler, T., Nauman, M., Hong, T.W.: Specification and standardization of a java trusted computing api. Softw. Pract. Exp. **42**(8), 945–965 (2012). http://dx.doi.org/10.1002/spe.1095
72. Toegl, R., Winkler, T., Pirker, M., Steurer, M., Stoegbuchner, R.: IAIK Java TCG Software Stack – jTSS API Tutorial (2011). http://trustedjava.sf.net. Website accessed 14 Nov 2012
73. Trusted Computing Group: TCG Software Stack (TSS) Specification Version 1.2 Level 1 Errata A (2007). http://www.trustedcomputinggroup.org/resources/tcg_software_stack_tss_specification. Website accessed 29 Jan 2013
74. Trusted Computing Group: TCG PC Client Specific TPM Interface Specification (TIS) specification version 1.21 revision 1.00 (2011). http://www.trustedcomputinggroup.org/resources/pc_client_work_group_pc_client_specific_tpm_interface_specification_tis. URL http://www.trustedcomputinggroup.org. Website accessed 29 Jan 2013
75. Trusted Computing Group: TCG TPM specification version 1.2 revision 116 (2011). http://www.trustedcomputinggroup.org/resources/tpm_main_specification. Website accessed 29 Jan 2013
76. Trusted Computing Group: Trusted Platform Module Library part 1: Architecture – Familiy "2.0" Level 00 Revision 00.96 (2013). http://www.trustedcomputinggroup.org/resources/tpm_main_specification. Website accessed 1 July 2013
77. UBM Tech: 2013 embedded market study (2013). http://e.ubmelectronics.com/2013EmbeddedStudy/index.html
78. W3C XML Protocol Working Group: SOAP Version 1.2 Part 1: Messaging Framework. W3C Recommendation, W3C (2007). http://www.w3.org/TR/soap12-part1/
79. Weiser, S., Tögl, R., Winter, J.: Measured firmware deployment for embedded microcontroller platforms. In: MeSeCCS Proceedings, Lisbon. SCITEPRESS (2014)
80. Winter, J., Dietrich, K.: A hijacker's guide to communication interfaces of the trusted platform module. Comput. Math. Appl. **65**(5), 748–761 (2013). http://www.sciencedirect.com/science/article/pii/S0898122112004634
81. Xingkui, W., Xinguang, P.: The trusted computing environment construction based on jtss. In: 2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC), Jilin, pp. 2252–2256 (2011)

82. Xinguang, P., Wei, J.: Filter-based trusted remote attestation for web services. In: 2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), Beijing, vol. 3, pp. 5–9 (2010). doi:10.1109/ICCSIT.2010.5564906
83. Yan, J., Peng, X.: Security strategy of DRM based on trusted computing. J. Comput. Inf. Syst. **9**(7), 3226–3234 (2011)
84. Zic, J., Nepal, S.: Implementing a portable trusted environment. In: Gawrock, D., Reimer, H., Sadeghi, A.R., Vishik, C. (eds.) Future of Trust in Computing, pp. 17–29. Vieweg+Teubner, Wiesbaden (2009). http://dx.doi.org/10.1007/978-3-8348-9324-6_2