Usability and Security of Trusted Platform Module (TPM) Library APIs

Siddharth Prakash Rao and Gabriela Limonta, *Nokia Bell Labs;*
Janne Lindqvist, *Aalto University*

https://www.usenix.org/conference/soups2022/presentation/rao

This paper is included in the Proceedings of the Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022).

August 8–9, 2022 • Boston, MA, USA

978-1-939133-30-4

Usability and Security of Trusted Platform Module (TPM) Library APIs

Siddharth Prakash Rao
Nokia Bell Labs, Finland

Gabriela Limonta
Nokia Bell Labs, Finland

Janne Lindqvist
Aalto University, Finland

Abstract

Trusted Platform Modules (TPMs) provide a hardware-based root of trust and secure storage and help verify their host's integrity. Software developers can interact with a TPM and utilize its functionalities using standardized APIs that various libraries have implemented. We present a qualitative study (n=9) involving task analysis and cognitive interviews that uncovered several usability and security issues with `tpm2-tools`, one of the widely used TPM library APIs. Towards this end, we implemented a study environment that we will release as open source to support further studies.

Our results support two major conclusions: 1) `tpm2-tools` APIs, as designed, are not designed to be developer-friendly, and 2) One of the major causes for these usability issues is in the TPM specifications. Since other libraries also mirror the specifications and provide no significant usability improvements, our results are likely to indicate similar issues with all current TPM library APIs. We provide recommendations for improving the TPM library APIs documentation and software, and we highlight the need for HCI experts to review TPM specifications to preemptively address usability pitfalls.

1 Introduction

A Trusted Platform Module (TPM) [51] is a tamper-resistant chip that is used as a hardware-based root of trust in many modern applications [34, 61]. TPMs can carry out common cryptographic operations, such as secure key generation, encryption, hashing, and signing. Furthermore, since the TPM is physically isolated from the processing system of its host, it can be used for securely storing a small amount of sensitive data (e.g., keys and certificates), which can further be utilized for verifying the integrity of its host. TPMs also provide various non-cryptographic security features for imposing access control restrictions on the objects created or stored in the TPM. Such restrictions play a crucial role in hardening the security of applications built using TPMs. The Trusted Computing Group (TCG) defines standard specifications that cover TPM architecture and implementation [56], and several high-level Application Programming Interfaces (APIs) to interact with the TPM hardware [52, 54, 55]. The latter is implemented by various software libraries, and is the scope of our study.

APIs play a crucial role in modern software development because they provide reusable components for developers to build applications efficiently and in less time. Nevertheless, APIs tend to be complex, and making them usable (or developer-friendly) has been an ongoing research theme. Previous works have analyzed various APIs that offer security using cryptographic features to understand and improve their usability [13, 40]. In this work, we extend this research theme for TPM library APIs, which offer additional non-cryptographic features (e.g., access control). We believe that the combination of these security features adds more to the complexity of TPM concepts and hinders the APIs' usability and security. Our work explores them by systematically analyzing the implementation of `tpm2-tools`, a widely used TPM library API with 85830 downloads (refer to Appendix A).

Our main goals are to understand the usability and security pitfalls of TPM developers and review the current API implementation to provide concrete design guidelines for usably secure API development. In this realm, we conduct a qualitative study with TPM developers using mixed methods (task analysis and cognitive interviews). We identify common use cases of TPM, conduct a thorough review of the available APIs, literature survey of the prior art, and combine them to design tasks and questionnaires for our participants. We also involve the participants in a follow-up interview to understand their experiences, perceptions, and opinions about the APIs. We conduct thematic analysis and code analysis to identify themes and common coding patterns that give an overview of the usability and security pitfalls of `tpm2-tools` library APIs. Based on these results, we provide concrete recommendations for the library documentation and software.

Contributions: First, we built a study environment that supports all major TPM libraries and works right out of a browser [6]. It is the only available platform for studying TPM-related tasks to our best knowledge.

Second, we show that the `tpm2-tools` APIs are not user-friendly based on a systematic study involving an analysis of developers' inputs collected through tasks and interviews. Although various guidelines are available to design usably secure APIs, we find that the `tpm2-tools` library have not seriously considered or implemented them. Consequently, developers struggle to use the APIs efficiently and are prone to make trivial mistakes that undermine security. The complexity of the topics and lack of developer-friendly APIs and supporting materials could pose a major barrier for developers to fully utilize TPM's capabilities. Our work identifies and highlights various usability pitfalls that impact security and provides concrete recommendations to address them.

Third, we highlight that standard specifications are a potential venue to influence the usability of technologies. We observed that the TPM API implementation strictly follows the standards and found many instances where the pitfalls could be traced back to them. Thus, we believe that there is a need for HCI experts to be involved in the design and review of standards to preempt any possible usability pitfalls that otherwise would be propagated to the software implementation.

2 Background

This section covers the background of TPM security features. We explain the following components: cryptographic operations, hierarchies, TPM-specific restrictions, platform configuration registers, authorizations, and sessions. The TPM uses these components to provide security-related functions, such as key generation, the hardware-based root of trust, device identity, remote attestation, and secure storage.

The TPM supports common cryptographic operations such as **encryption**, **signing**, and **hashing**. Additionally, it provides secure key generation functionalities, where the TPM-generated keys can be used internally for cryptographic operations, or they can be exported for external applications. The keys that reside within the TPM are protected by a logical abstraction (i.e., collection of objects) called a **hierarchy**. The TPM provides four hierarchies: *owner*, *endorsement*, *platform*, and *NULL*. The first three are meant to be used by the TPM's owner, manufacturer, and host platform. The *NULL* hierarchy is reserved for short-lived objects that are expected to be lost on reboot. In addition to keys, a hierarchy can include another kind of object: a sealed data blob, a structure for protecting small amounts of user data in the TPM.

Each hierarchy is associated with a random seed, which is used to generate primary keys that can serve as the root in a tree of other (child) objects inside the hierarchy. Primary keys are stored inside the TPM and cannot be exported or read externally, whereas child objects (such as sealed data blobs and non-primary keys) are generated on the TPM but stored in the disk until loaded back for actual use. Since the child objects leave the trust boundary of the TPM when exported, they can be misused. In such cases, the child's security is guaranteed by *wrapping*, a mechanism where the child's sensitive part is encrypted by its parent key and can only be decrypted upon loading into the TPM.

The TPM gives users control over the usage of TPM-generated keys by allowing them to impose different **restrictions** over the key's *purpose*, *duplication*, and *usage*. The *purpose restriction* implies restricting the key for encryption/signing or only for decryption by setting the attributes `sign` and `decrypt`, respectively. The term *duplication* in the TPM context refers to the possibility of a key having more than one parent. The *duplication restriction* includes setting the `fixedParent` attribute to allow the key's parent to change within the same TPM or `fixedTPM` attribute to allow the key duplication for using it on a different TPM. Finally, the *usage restriction* refers to restricting the key, by setting the `restricted` attribute, to sign/decrypt only TPM-internal data. Please note that these attributes can only be set during the key generation; hence, the restrictions cannot be updated later.

TPM manufacturers utilize and predefine the key restrictions to provide two special keys: the endorsement key (EK) and the attestation key (AK). The EK is a restricted encryption primary key generated from the endorsement hierarchy's seed; it is certified by the TPM manufacturer as proof of its authenticity. The AK is a restricted signing key protected by the EK as its parent. The TPM can use the AK to prove its unique identity, e.g., during remote attestation. It guarantees that a specific, legitimate TPM has produced the signed message.

TPMs provide **Platform Configuration Registers (PCRs)** as an option for secure storage. PCRs are a set of 24 registers used to store hashes of different system components (e.g., firmware, kernel, hypervisor, operating system and files in the filesystem) usually during the boot sequence. PCRs are considered secure storage because a user cannot directly write hash values into the PCRs. Instead, they can provide the hash, and the TPM will extend one of the PCRs by concatenating it with the preexisting PCR content. The concatenated result is fed to the hashing algorithm to compute the digest, which is then stored in the PCR as the new value (referred to as *measurements*) to represent the state of a TPM's host platform at that given time. The hashing algorithm used determines the size of the measurement. The TPM supports multiple hashing algorithms and the group of PCRs associated with the same algorithm are referred to as a *PCR bank*.

PCR measurements can be used in remote attestation to verify that the integrity of a device has not been tampered with. TPM uses *quotes* – the measurements and other information (e.g., clock and number of reboots and suspends)– and signs it with a restricted key (e.g., AK). This process of obtaining a TPM-equivalent of message authentication code is called *quoting*. A verifier uses the quotes to detect tampering by checking the contents against reference known values. Furthermore, since an AK signs the quote, it guarantees the verifier that the measurements are trustworthy and generated by a legitimate TPM. If tampering (e.g., changes to firmware or kernel) is detected, the verifier can also identify the exact component that is tampered with because each PCR contains a measurement that represents a specific host component.

The TPM provides a limited amount of Non-Volatile Random Access Memory (NVRAM) that can be used to store persistent data. Users may use NVRAM for secure storage because access to NVRAM can be restricted with TPM security mechanisms such as *sealing*. When an NVRAM area is sealed against a particular state of the host, its content can only be read if the state is unaltered.

Similar to sealing, TPM provides other types of restrictions for TPM objects (e.g., keys, sealed data blobs, or NVRAM areas). For example, a user can create policies to define restrictions on how an object can be used. Furthermore, it can also impose **restrictions against TPM-internal and TPM-external states**. The TPM-internal states include the PCR state, NVRAM contents, and TPM counters, and the TPM-external states may be passwords, state of external hardware (e.g., biometrics or GPS information), and signatures from smart cards. These restrictions are set when an object is created, and they will be checked before the object is used.

A user may need to use **session-based authorizations** to comply with the restrictions imposed on objects in the TPM or to authorize commands. Sessions are a way to communicate authorizations to the TPM since they carry the information needed to prove that the user can perform the intended action. The authorization contained in a session may be reused to execute several commands repeatedly, as sessions preserve the state between commands. For example, HMAC sessions may be used to communicate a password more securely, as the password only needs to be specified once during the creation time of an object (e.g., a key). In such sessions, the password is used as an input to calculate the HMAC of each command and response from the TPM, which allows authorizing an action without actually sending the password.

3 Related work

TPMs have been used in a wide range of security-critical applications [26, 27, 37, 38, 48]. Security flaws in the TPM could undermine the security of the applications that use it, and finding such flaws is attractive to researchers. Prior research works have analyzed the TPM architecture and specifications to verify the security guarantees of TPM [22–24, 49, 62]. These existing studies mainly on formal methods for analyzing TPM specifications. To our best knowledge, analysis of TPM software libraries and their API implementation has not been studied. In our study, we explore this topic by involving the TPM developers to understand how they use the APIs and investigate how the API implementation could pose security and usability barriers.

We build upon the prior research that identified various human factors of secure application development. In particular, we draw insights from the studies that evaluated cryptographic libraries [13, 17, 28, 35, 40, 44] due to an overlap of cryptographic features offered by TPM library APIs. These studies have found that lack of usable APIs and documentation are major barriers to developers. Our study evaluates whether these issues are replicated in the TPM ecosystem.

Similarly, previous studies have identified that developer's background [19, 43, 63], information sources [14–16], and workplaces [18, 32] are some of the other factors that affect secure application development. We are particularly interested in the information sources referred to by TPM developers as supporting materials. They may have to refer to a broader set of sources, e.g., library documentation, TPM specifications, and other security guidelines, and their effect on the way developers use TPM library APIs is yet unknown. In our study, we explore the role of such sources in TPM development.

4 Research Methodology

We first created a pool of potential participants and sent them a preliminary survey (refer to Appendix B) inviting them to participate in the study. This survey helped us filter suitable participants and identify the common use cases for designing the tasks. In parallel, we conducted an independent analysis of the TPM ecosystem to scrutinize the intrinsic details of concepts, software libraries, and supporting resources (e.g., standards and documentation) available for the developers. This helped us define evaluation criteria for the tasks. Similarly, we conducted a literature review of previous usable security research works (e.g., [13, 47]) , that served as a reference when preparing our questionnaires. Next, we designed TPM-related tasks and questionnaires to collect practical and conceptual barriers (mental models) while coding with TPM libraries. We built our study environment (refer to §4.1) using open source modules and hosted it on our servers. After participants completed the tasks, we conducted cognitive interviews to understand their experiences while dealing with TPMs (refer to §4.3). During the interview, we allowed them to introspect about their experience with the TPM ecosystem and provide suggestions for improvement. Finally, we conducted a two-stage analysis (refer to §4.4) of the data we gathered from the study environment and the interviews.

Participant demography. We targeted experienced developers with TPM experience such that we could design realistic tasks leveraging TPM-specific security features. We aimed to evaluate usability and security pitfalls closer to real-life TPM usage, as opposed to struggles with its steep learning curve. We used purposive sampling (refer to Appendix C for more details). Our results are based on the responses from the 9 participants who matched the criteria for our target population and went through the entire study protocol. We offered them a compensation worth 100 €. Participants were male (aged 18–59) with a security background, <1–5 years of TPM, >=2 years of coding experience and a bachelor's degree.

Ethical and privacy considerations: We followed best practice guidelines throughout our study [47]. Our institution's review team approved our study and confirmed that it meets the ethics and privacy standards. After a careful review of our methodology, the review team drafted a GDPR-compliant *privacy policy* and *participation consent form*. We presented both of them to the participants before the study began. Accordingly, the collected personally identifiable information (e.g., email and names) was solely used for contacting the participants. We excluded such data from the analysis and discarded it immediately by the end of the study. Furthermore, we used open-source components to build our study environment, hosted it on our servers, and ensured that the participants' data was entirely under our control.

4.1 Task and questionnaire design

Tasks. We identified four common use cases of TPM from our preliminary survey as follows: *encryption* (symmetric and asymmetric), *storing measurements* of files on PCRs, *securing secrets* on the TPM, and *remote attestation*. We also identified various cryptographic and non-cryptographic security features. We designed simple tasks around the use cases and added conditions with a combination of security features to evaluate functional correctness and the participant's security choices. Such conditions allowed us to understand whether a knowledgeable developer (from a coding and security point of view) would be able to choose suitable cryptographic parameters and impose TPM-based security restrictions while dealing with common use cases. The tasks required the participant to use well-known TPM commands. Each task was divided into simple steps (refer to Appendix D) to ensure better understanding and obtain higher completion rates. We ran a pilot study to evaluate this. We improved the text and adjusted the tasks' complexity based on the feedback.

Table 1 summarizes the mapping of use cases and security features of our tasks. We assigned four tasks to the participants and did not impose time restrictions for completion. As the first task, five of them got asymmetric encryption and the rest got symmetric encryption. The remaining three tasks were common to all. We assigned a specific library to each participant based on their preliminary survey responses: one was assigned `IBMTSS` and the rest `tpm2-tools`. However, switching libraries was allowed, and the participant whom we assigned `IBMTSS` switched to `tpm2-tools`.

Questionnaires. We collected data about the participants' backgrounds and understand their perceptions and opinions while using TPMs (refer to Appendix F). After showing the general instructions, we asked basic questions about the participant's *demographics*, *TPM background* and *contact details*.

After each task, we presented *task-specific* questions where the participants had a chance to report their perceptions and opinions about the task. In particular, we wanted to understand their familiarity and complexity perception about the task; also their security and correctness perception about their response. We also asked questions about the type of resources they used, the reason for not completing the task (if applicable), and their opinion on the usefulness of the error messages in fixing their mistakes or making secure choices. The questionnaire used a 5-point Likert scale.

After attempting all the tasks, we presented an exit questionnaire to ask about their use of supporting materials for TPM-related activities. This question was asked to compare with a similar question asked in the *task-specific* questionnaire and to understand whether they had to refer to new types of materials for our study. We also asked them why they referred to supporting materials outside the library documentation.

4.2 Study environment

We built our study environment as an online Integrated Development Environment (IDE) and hosted it on our servers. Refer to Appendix G for the technical details. We designed the study environment using our personal experiences to mimic the real-world TPM development conditions with minimal participant effort, and our participants confirmed its ecological validity during the interviews. Each participant got a dedicated server accessible over a unique URL and only needed a browser to participate. The IDE allowed interacting with the TPM emulator using function calls provided by each supported library (see Appendix A for details of supported libraries). The participants were free to switch between libraries at any point during the study by choosing from a dropdown menu in the IDE. We also added two additional features to the IDE to *reboot* and *reset* the TPM to its initial state.

Figure G.1 shows the interface of our study environment. The environment started with a *Welcome* page, which contained the study's purpose and logistics, an instructional video that covered basic features and navigation of the environment, and links to FAQs. This page was followed by the *Demographics* page. Then, every task was shown on two pages: one for the task description and IDE and the other for the task-specific questionnaire. In the end, the *Final questions* page included the exit questionnaire. Participants were free to move between tasks and attempt them any number of times. But we stored everything they executed in the IDE.

Task	Security features		Description
	Crypto	Non-Crypto	
Asymmetric encryption	C2	NC1, NC2	• Create secure encryption keys using the TPM • Perform asymmetric encryption • Impose restrictions on key attributes to allow duplication of the key, which makes it exportable to other devices
Symmetric encryption	C1	NC4, NC5	• Create password-protected symmetric encryption keys using the TPM • Perform symmetric encryption • (Optional) Use TPM sessions to authorize the use of a password protected key
Storing measurements	C4	NC6	• Perform secure hashing • Identify suitable PCRs for storing measurements • Extend measurements to all available PCR banks in the TPM
Securing secrets	—	NC1, NC3	• Create NVRAM index to securely store information in the TPM • Use the correct parameters for NVRAM index creation • Seal the reading operation against a PCR state • Lock the NVRAM index against future writes
Remote attestation	C3	NC1, NC2, NC6	• Create secure keys using the TPM • Impose restrictions on the key used for signing a quote • Create a quote with the TPM including the state of the kernel (stored in PCR2) • Verify if a given quote is valid for remote attestation

Table 1: Mapping of use cases and security features into tasks (also refer to Appendix E and D)

4.3 Interview

We conducted semi-structured interviews with 9 participants to extract in-depth qualitative insights beyond the task analysis. In particular, we were interested in understanding participants' mental models and experiences while working with TPMs. We also wanted to know how they would resolve the usability and secure programming barriers that we identified. Two of the authors facilitated the interviews, one for leading the conversation and the other for observing and note-taking. We prepared the main questions we wanted to cover with our participants. We analyzed each participant's responses to pick the most suitable version of their code snippets and relevant observations from their questionnaire responses. During the interviews, we showed these responses and used appropriate probes to obtain in-depth information about our main question. We invited the participants for a one-hour online interview, and asked for consent to record and auto-transcribe the call. The interviews were loosely structured into three segments. Refer to Appendix H for more details.

(1) Introduction. We first reminded the participants about the study details and explained the purpose and structure of the interview. Next, we probed them with open-ended questions about the working mechanism of TPMs, their use cases, and their relevance to participants' work. We also confirmed that the participants were comfortable with the study environment and had a positive overall experience with the logistics.

(2) Task- and questionnaire-specific observations. We were interested in understanding the participant's motivation behind specific choices, as well as their mental model and problem-solving approach when completing the tasks. We showed them their responses and asked them to outline their approach. We used necessary probes (e.g., *"how did you pick the cryptographic algorithms for creating the keys?"*) to understand further their thoughts on the security and correctness of their approach. We also probed to check their awareness of alternative approaches and discussed their advantages and disadvantages. In addition, we tried to understand how they search for relevant information about TPM-related topics.

(3) General discussions. We probed the participants on general topics outside the tasks and questionnaires. For example, we asked the participants for their usual approach to solving TPM programming tasks. Also, we collected their suggestions on improving the TPM ecosystem and offered them a chance to address anything that was not directly part of our questions.

4.4 Overview of analysis

Our analysis included two phases. In the first phase, we analyzed the data collected from the study environment, such as the code snippets executed in the IDE and the responses to the questionnaires. While we analyzed the code snippets for correctness and security, our primary goal was to understand the typical solutions and coding patterns of TPM developers. Furthermore, we identified interesting code snippets and responses to use as probes for the interview. We present our observations about common coding patterns in §5.2.

In the second phase, we conducted independent and iterative analysis of the interview transcripts for thematic coding (refer to §5.1). First, we identified three broad categories under which we could explore themes. Then, two of the authors independently coded the transcripts (using an inductive approach) to generate a list of all potential codes that would suit these categories. We noticed that no new concepts emerged from the last two interviews, which indicates saturation. This process was repeated several times, until all concepts mentioned by participants were reflected, to refine the codebook. We consolidated the codebooks with refined code names and guidelines. The codes represented concrete roadblocks or influencing factors for the TPM developers. We do not report any measure of inter-rater reliability (IRR), because we report no quantitative results and the iterative review of the codes was the process to yield the themes we identify in our work [36]. Furthermore, the codes themselves are not the product of our work. We identified eighteen codes under seven themes from our thematic analysis (refer to Table 2).

5 Results

In this section, we present our findings of themes from the thematic analysis and common coding patterns of TPM developers. The results from thematic analysis (in §5.1) are presented loosely as a tuple of *observations* and *evidences* along with a brief *discussion* pertaining to each theme. Each theme's main observation is aggregated from analyzing the developer's perceptions and opinions of the questionnaire data and thematic analysis of the interview transcripts. Relevant snippets from the participant's code submissions or quotes from interviews are used for justifying the observations. The discussion reflects our observation from the literature analysis and conversation with the participants about the potential treatment to some of the roadblocks they usually face while working with TPMs. We have followed a similar approach for presenting the common coding patterns in §5.2.

5.1 Themes emerged

Our thematic analysis yielded three categories: library, supporting materials and user themes, as summarized in Table 2.

5.1.1 Library Themes

This category captures the reasons why something went wrong when completing the study, which were related to the library itself. We identified three themes as follows:

Naming conventions and usage. Participants P1–P3, P5, P8, and P9 expressed difficulties with the naming conventions used in the libraries and deciding when to use each command. This theme includes codes around three areas: *confusion between two available options* with overlapping functionality, *names do not convey the functionality or cause confusion*, and *inconsistency* in syntax when specifying similar parameters but for different commands.

For example, P1 was confused between the commands `tpm2_createprimary` and `tpm2_create`. The former is used to create primary objects (such as *primary keys*), and the latter is used to create child objects (such as *child keys* and *sealed data blobs*) that are protected by a primary key in the hierarchy. P1 was confused because the difference in functionality was not clear from the names of the commands.

While the library implementation follows the specifications [52], they can alleviate the confusion by providing abstraction functions with more appropriate names. We suggest that the functionalities of the original commands can be split into two new commands: e.g., `tpm2_createkey` for creating parent or child (with `-P` and `-C` as flags, respectively) and `tpm2_createblob` for creating sealed data blobs. Nevertheless, such abstractions must retain the parameters used in the original commands and implement secure defaults.

Output formats. P1–P4, P6, P8, and P9 had difficulties interpreting the outputs of different commands they used for completing the tasks. The problems include *insufficient information* and a *lack of clarity on how to interpret* the output.

In the remote attestation task, the participants were asked to verify a quote, which required using `tpm2_checkquote` to verify the contents and signature of a quote. The output lacks a success message, making developers rely on ad hoc methods to confirm the verification. For instance, P9 said that "I think I was doing just a minimal verification (...) I was executing this command twice, one with the correct files and one with false files to see if the end result was different". The above example is also the case where the output lacks information about what is verified. Altogether, developers may wrongly interpret the output and assume that the verification succeeded as the command did not return any errors.

We also noticed a lack of details in the meta-information shown to developers. For example, when a key is created, the attributes of the key are printed as `fixedtpm|fixedparent|restricted|decrypt`. This meta-information works as a reminder of the capabilities and restrictions of the key. Although our participants found this meta-information helpful, and it seemed important, they admitted their reluctance to check each meta-information even if they did not fully understand it. While the brevity of such meta-information suffices for seasoned developers, the less-experienced ones still have to refer to the documentation to understand their meaning.

Clear description, success, and error messages are important for the developers [29]. Hence, a possible treatment for this situation is to return more verbose outputs that clearly confirm a successful command or indicate what went wrong.

Error handling. All participants except P6 expressed difficulties when interpreting error and warning messages provided by the library. They found the messages to be unclear and hence did not fully understand them. Also, the messages did not provide any valuable feedback (i.e., lack pointers on resolving the errors), and the participants did not have enough domain-specific knowledge to fix it by themselves.

For example, all participants of the symmetric encryption task failed to specify an initialization vector (IV). Although a warning message indicated that the IV was weak, everyone ignored it because there was no feedback on how to specify the IV. P5 commented that,

> I didn't really find a way how I could specify a better IV and, I don't know, **I find it's kind of destructive criticism when the**

> **program just tells me "well, you used the wrong IV", but doesn't make any comments on how to do it better.** So, if it finds out hey, you are using a weak IV, it could suggest the use of the appropriate flag to specify a better one.

We traced the cause of this problem to the example in the documentation, which lacks an IV [10]. Therefore, we believe that participants copied this example and ignored the importance of an IV. One solution to this problem is to update the code in the documentation. Also, the command could return a descriptive error message that suggests how to specify the IV and the correct flag to use. The library developers can refer to the RUST compiler [60] as a good example of suggestions to include in the error messages.

Themes		Codes	Participants
Library	Naming conventions and usage	Confusion between two available options	6/9
		Names do not convey functionality or cause confusion	
		Inconsistency	
	Output formats	Insufficient information	7/9
		Lack of clarity on how to interpret	
	Error handling	Error message is unclear	8/9
		Lack of pointers on how to resolve	
Supporting materials	Documentation shortcomings	Lack of examples	7/9
		Lack of background information	
		Incorrect or missing explanations	
		Complicated presentation	
		Not easily accessible	
User	Mental models	Misunderstanding by user (documentation is correct)	5/9
		Misunderstanding due to missing or incorrect documentation	
		Misunderstanding due to unknown sources	
	Trust factors	Past experience or code	7/9
		Defaults and documentation examples are secure	
		TPM's ability to handle security	

Table 2: Identified Themes

5.1.2 Supporting Materials Themes

We asked participants what supporting materials they used to complete our study's tasks and in general when doing TPM-related coding. Supporting materials refer to the library documentation, TCG standards, and additional resources (e.g., blogs, personal notes, forums). Although all participants reported using the library documentation as the primary resource, we found that participants with less TPM experience reported relying on the TCG standards to get background information and on third-party forums when they faced issues. On the other hand, experienced TPM developers said using only the library documentation to obtain ready-made code examples that they can tweak. They know what to look for based on previous experiences.

We now present the common themes around supporting materials, especially their shortcomings. We limit the results in this section to library documentation because the participants did not provide details about the exact resource they used, and also, we could not trace them. Nevertheless, we believe that the shortcomings we report in this category may also exist in all types of supporting materials.

Documentation shortcomings. Participants P1–P3, P5, P7 and P9 indicated a *lack of examples*, as well as a *lack of background information* about related TPM or security concepts. They emphasized the need for customizable examples and descriptive background information, primarily aimed at beginners. In particular, P5 stated that

> What I would really love would be example code. (...) I mean, there is example code for the simple problems, **but as soon as you want to do something that goes away from the simple problems, it gets a little difficult.**

Participants P1, P5, and P7 highlighted that the documentation had *incorrect or missing explanations* and that they would prefer clear explanations for using one approach over another. An example of this theme arose in the securing secrets task, where we asked participants to store a secret string "in the TPM" and to impose restrictions on reading and writing that secret. We expected them to create an area in the NVRAM with the appropriate security controls for accessing the memory area. Instead, most participants (6/9) created a sealed data blob, which is stored outside of the TPM. During the interviews, we learned that the participants were aware of both approaches and the latter being a less secure one. However, most participants mentioned that they avoided the NVRAM approach because the documentation lacked a good explanation. Also, it would be too time-consuming for them to figure out how to complete our task using NVRAM. Hence, they settled for the less-secure but better-documented approach.

Finally, P1, P3, P7–P9 indicated that supporting materials had a *complicated presentation* or were *not easily accessible*. In particular, P9 said it was time-consuming to find the command they would need from the library documentation:

> The first task was taking a lot of time because I couldn't find the correct command. It was like, I know what I'm going to do, but none of these commands seems to be relevant. (...) And then it was almost by accident, that I found out the correct option.

All the aspects mentioned in this section could be tackled by improving the documentation quality and adding more background information about TPMs. Additionally, the documentation could include use case-based example tutorials. The `tpm2-tools` library started adding these kinds of comprehensive guides [58] with code examples and background information, but it has been defunct since February 2021.

5.1.3 User Themes

Mental Models of TPM Concepts. We identified different mental models formed by the participants that led to misconceptions of TPM concepts. We observed an example of a *misunderstanding by the user* in the asymmetric encryption task, where the participants had to create a key (using `tpm2_create`) that is exportable to another TPM. P3 misunderstood the documentation and formed an incorrect mental model about the trust boundaries of the TPM, believing that any key created with a TPM can be loaded into other TPMs. They stated that,

> When I used the `tpm2_create` command, you can see that I have given two files with `-u` and `-r`. So the keys are already outside. So, I know that if I can load this key in another TPM, it should work because normally any key can be loadable on TPM as long as it is cryptographically valid.

It is true that the flags `-u` and `-r` return the public and private keys in two separate files. However, the private part remains protected by its parent key in the TPM, and it cannot be used outside unless its restrictions are relaxed. This can only be achieved by setting the `fixedTPM` and `fixedParent` attributes of the key to `False`, which P3 missed. The man page for `tpm2_create` [9] covers all flags and includes a link to a separate page [4] that covers the flag `-r` in detail. The latter page mentions the private key (i.e., the child key) being protected by its parent and further links to the TPM standard [51], which explains these concepts in depth. Although the documentation is correct, we believe that its nested presentation could have caused P3 to form a wrong mental model about the trust boundaries of the TPM. A potential solution would be to simplify the documentation by covering all necessary details (e.g., command usage, TPM concepts, and concrete examples) in a single page.

Another example from P3 is an incorrect mental model about remote attestation, due to a lack of background information in the documentation. They had not done any remote attestation tasks before our study and only read the `tpm2_checkquote` [8] man page. This man page does not include a high level of detail about all the requirements that need to be verified to trust a quote; it only discusses the quote's signature and reference PCR values. This, along with the insufficient information in the output of the command, implied to the participant that verifying a quote meant just checking the signature. However, this is only a part of the verification, since the verifier must also check the signing key's properties, e.g. that it is `restricted` and comes from a legitimate TPM. Such incorrect mental models may lead a developer to implement remote attestation insecurely. The library can address this issue by improving the already existing tutorial for remote attestation, which lives outside the documentation [57]. The tutorial can be improved by adding concrete code examples and moving it to the `tpm2-tools` documentation.

We also found several cases of incorrect mental models and misconceptions, but we could not trace the exact source that confused the participants. In such cases, we could only confirm that the confusion did not originate from the documentation and speculate or attribute it to *unknown sources*. An example of such a case emerged in the discussions about the asymmetric encryption task. Many participants were unsure of why they had to create parent keys before creating child keys, but they blindly followed the documentation examples. On the other hand, one participant (P8) had selected cryptographic parameters only for the parent key but not for the child key, as they believed the parameters would be inherited. We believe that P8's security background and the use of the words parent/child appealed to them that the child key inherits security properties from its parent. However, in reality, the child key will only be as secure as the defaults defined for the library, with no impact from its parent's cryptographic parameters. A possible treatment would be to simplify the key creation process, e.g., by offering abstraction functions to create both the parent and the child key.

Trust factors. We observed that the participants relied on their trust in different factors when making secure choices. For example, P1, P3–P5, P7, and P8 trusted and relied on their *past experience* of completing similar tasks. As a result, they preferred to use their old code snippets instead of figuring out how to approach our study's tasks from scratch. However, one pitfall of this method is that outdated code snippets could lead to trivial errors, e.g., due to software version mismatch. One participant encountered such a situation and struggled to complete a task before realizing that they wrote their code using an outdated API version.

Similarly, we observed that developers tend to trust that the *library defaults and documentation examples are secure*. This was prevalent in situations where the lack of a relevant security cryptographic background prevented the participants (e.g., P1–P3) from feeling confident in making an informed choice. Therefore, the library needs to provide secure default values and examples to support less experienced developers.

Finally, participants P3 and P8 had implicit trust in the *TPM's ability to handle security* details, so they did not have to make any explicit choices. This subtheme was prevalent in the tasks where participants had to create parent and child keys, where they would trust the TPM to take care of security aspects of those keys. P3 expected that, when using the endorsement hierarchy, the TPM would prevent the user from creating signing keys that would allow a third party to correlate a set of signatures and determine that they came from the same TPM; whereas P8 expected the TPM to assign child keys the same cryptographic parameters as their parent key. We suggest that the library should clarify in the documentation which security aspects are covered by the TPM to inform users about the security guarantees it can provide.

5.2 Common coding patterns

This section highlights the common coding patterns followed by participants. We limit our discussion to the features for which we found interesting patterns.

5.2.1 While using non-cryptographic security features

Here, we cover the common patterns observed when using the following non-cryptographic features of the TPM: *use of hierarchies* (NC1), *key restrictions* (NC2), *session-based authorizations* (NC5) and *PCR usage* (NC6). We evaluated feature NC1 when creating TPM objects in the asymmetric encryption, securing secrets, and remote attestation tasks. We wanted participants to avoid using the NULL hierarchy since objects in this hierarchy are lost upon reboot. Most participants selected the correct hierarchy for the use case but relied on the defaults. This pattern highlights the importance of providing relevant and secure defaults, as discussed in the trust factors theme in §5.1.3. The exception to this pattern occurred in the remote attestation task, where most participants explicitly selected the endorsement hierarchy due to its privacy protections. We noticed that some participants were reluctant to rely on defaults if the use case demanded a specific type of protection from the hierarchy and would instead explicitly specify the hierarchy. We speculate that developers hesitate to rely on defaults and tend to be extra cautious when specifying parameters if the use case's security requirements are well understood. Hence, the library must create awareness by highlighting such requirements in the documentation.

Then, feature NC2 was evaluated when participants set restrictions on the keys created for the remote attestation task, where they were asked to create a `restricted` key for signing a quote. A signing key without the `restricted` attribute could be misused during remote attestation to sign any data, including a forged quote. Although the library offers a convenience function `tpm2_createak`, which takes care of setting the `restricted` attribute automatically, only two participants used this function. Moreover, only 3/9 participants created restricted signing keys, whereas the rest relied on the default key attributes set at creation time. Again, we found that most developers rely on key's default attributes instead of setting them explicitly and that convenience functions are rarely used.

We evaluated NC5 in the symmetric encryption task, where participants had to encrypt and decrypt both a file and a string multiple times using a password-protected key. When password authorization is used, the password is sent in plaintext between the user and the TPM every time the participants perform an operation. A local attacker can eavesdrop on such communication to capture the password. A secure way to use password protection is to utilize TPM session utilities, e.g., HMAC or policy sessions (refer to §2); however, none of the participants used them. During the interviews, we found that many participants had theoretical knowledge of sessions but lacked the hands-on experience to use them.

Finally, feature NC6 was evaluated in the storing measurements tasks, where participants had to pick suitable PCRs for storing the measurements of a configuration file. We expected them to avoid PCRs 16 or 23, which can be reset and repopulated with arbitrary measurements during run-time by an attacker. We found that 2/9 participants used PCR 23. During discussions, they blamed the TPM specification [53], which states that PCR 23 is meant for "Application support" and deceived them into thinking PCR 23 is reserved for any application needing to store measurements. However, the specification also mentions that the operating system dictates its usage, and it may be reset and used at any time. Unfortunately, both participants had ignored the latter part and formed incorrect mental models. This highlights the importance of the specifications in shaping users' mental models and misconceptions.

5.2.2 While using cryptographic security features

We now report the common patterns observed while performing *symmetric encryption* (C1), *asymmetric encryption* (C2), *signing* (C3) and *hashing* (C4). For cryptographic security features C1, C2, and C3, we noticed that when participants were asked to create keys for encryption and signing, they first created a parent key in one of the hierarchies and then a child key. Also, most participants specified the cryptographic attributes (e.g., algorithm, key length, purpose, and duplication restrictions) only for the child key, whereas they relied on the library defaults for the parent. We traced back the origin of such patterns to the documentation examples in the `tpm2-tools` library that miss out on specifying the attributes for the parent. Child keys are stored outside of the TPM (e.g., on the disk) until they are loaded onto the TPM for use; therefore, it is good that developers are cautious and explicitly specify their attributes. Nevertheless, attributes of the parent are equally crucial because child keys may be compromised if their parent has weak or insecure attributes. We can confirm from our analysis that the defaults of the parent keys are secure. Nevertheless, the library needs to be aware that the developers have a high degree of trust in defaults, and it has a responsibility always to keep the defaults secure and updated.

Another common mistake appeared in the symmetric encryption task (feature C1). All four participants asked to complete the task failed to specify an IV, which lowers the security of the encryption process.

C4 was evaluated in the storing measurements task. We noticed that 8/9 participants relied on the hashing functions provided by the TPM to obtain the hash of the file they would later extend to a PCR, where one participant used an external library for hashing. TPM library developers should be aware of such patterns and restrict the use of external libraries that lie outside their control and may contain vulnerabilities.

On top of the above features, we observed an interesting pattern in the remote attestation task where the participants were asked to generate a quote. Including a random nonce with the quote allows the attestation server to defend against replay attacks. Unfortunately, only 2/9 participants' quotes included a nonce, where one used a random nonce, and the other had blindly copied the nonce from the man page example of the `tpm2_checkquote` command. During the interviews, we learned that most participants ignored using nonce because the documentation is inconsistent and does not emphasize its importance. Another reason for not using a nonce was the lack of a threat model explicitly indicating a replay attack. Some participants also noted that the library uses a relatively unfamiliar term (`-qualification`) to refer to a nonce, which does not immediately evoke the concept of a nonce.

The library can address the concerns presented in this section by revising the documentation to be consistent, providing secure defaults, clear explanations and examples, and using familiar terminologies that could remind the developers about common threats they should consider.

6 Recommendations

This section provides explicit recommendations, in the form of concrete action points, for improving the TPM library API documentation and software. Some of our recommendations overlap with Green and Smith's usable and secure API design principles [30]. These include abstracted and readable API design that does not go against developers' habits and mental models, non-ambiguous and safe defaults, and detailed and visible errors and outputs. We observed that `tpm2-tools` does not follow these existing guidelines; therefore, we reiterate and emphasize some of these principles in the form of actionable recommendations for the library developers. The emphasis of [30] is on making APIs easy to learn and use such that the developers have no need to understand the complexities of cryptography and minimal reliance on the documentation. While this standpoint is valid for mature ecosystems, it is crucial to understand complex concepts in niche ecosystems like TPM, as the knowledge is not widespread. We argue that it is still the library's responsibility to educate and assist the developers. Thus, our recommendations also focus on improving the documentation with missing details of the TPM background as the first step towards more usably secure APIs.

6.1 For library documentation

Technical specifications and standards are meant to be detailed and comprehensive. Despite that, many developers refer to software documentation or tutorials written by other developers. Our study confirms this phenomenon. In particular, we found that the participants' primary information source was the `tpm2-tools` documentation, and it influenced the developers' decisions. Although questionnaire responses deemed the `tpm2-tools` documentation satisfactory, our participants expressed frustration during the interviews while referring to them. Similarly, our analysis of the ecosystem revealed several shortcomings. We now present them collectively as concrete pointers to improve the documentation.

Include background information. Developers need to clearly understand TPM concepts and how they can be leveraged for security functionalities. We argue that the current documentation lacks background information about TPM concepts. As some of the experienced participants in our study pointed out, their theoretical knowledge and experience can only give them a sense of familiarity. However, given the complexity of concepts and the abundance of options available, providing more information would help them confidently utilize the needed options. Furthermore, we found several examples of misconceptions and incorrect mental models by less-experienced developers, which could also be addressed by providing additional background information.

Provide code snippets for common use cases. Most developers use code examples from the documentation as their starting point and repurpose them as per their needs. Although the `tpm2-tools` documentation contains simple code snippets, it lacks concrete examples for common use cases that require the use of multiple commands (e.g., remote attestation). Our participants also confirmed that they could not benefit much from the simple snippets. In this realm, we recommend that the TPM library uses the common use cases that we have identified (refer to §4.1) as a starting point and include comprehensive examples around them in the documentation.

Improve entry-level documentation. Despite their prior coding and security experience, all our participants shared similar struggles when they started developing with TPMs. They found the TPM documentation to not be beginner-friendly. Many beginners seem to have faced difficulties setting up their development environment on their local machine and setting up a communication interface with a TPM. We speculate that these difficulties, along with the complexity of TPM concepts, would discourage the developers and may be one of the reasons why there are few TPM developers. To this end, our recommendation would be to improve documentation with carefully curated content for entry-level developers. Our study environment provides an online coding experience that could be leveraged to teach how to code with TPM without installing anything on a local machine.

Include guidelines for picking security attributes. We found that developers selected security attributes and cryptographic primitives based on their prior experience rather than explicitly looking for existing guidelines (e.g., [41]). In the case of TPMs, developers have additional non-cryptographic security attributes to choose from, and a strong cryptographic background would not be enough to help with their decision. We recommend including brief guidelines within the documentation about both cryptographic and TPM-specific security attributes. Having all required information in the documentation would help developers formulate a security rationale and encourage secure choices. Any such guidelines should cover the various attributes available but explicitly promote the most secure ones (e.g., by using them in the code examples). At the very least, the documentation should include external links to any relevant guidelines.

Improve the documentation to address incoherence. We observed that developers were confused by incoherent aspects of the documentation on four occasions. The first occasion is with the confusing naming conventions (refer to §5.1.1) — i.e. when referring to commands or functions that share a common prefix (such as `tpm2_create` vs. `tpm2_createprimary`). The documentation could address this by alerting and reminding the developer about the other commands. Secondly, many commands are to be used in conjunction with suitable flags and parameters, and the developers doubted of their choices. Instead of just listing the available options, the documentation should provide verbose descriptions and concrete examples.

The third occasion is when there are multiple approaches for a task. For example, when to choose NVRAM over sealed blobs for storage is trivial for experienced TPM developers, whereas newcomers struggle to make the distinction. In such cases, the documentation could compare the approaches and help developers make the correct choice for their use case. Finally, several instances of confusion exist due to inconsistencies in the naming conventions across software versions. For example, a participant complained about discrepancies in flag usage between different versions of `tpm2-tools`. This inconsistency breaks Green and Smith's "ensure that APIs are easy to read and update" principle, confuse the participant and make them gullible to commit trivial mistakes. Along with clear documentation, these confusions can be prevented by using common naming conventions that align with developers' habits and mental models.

All our recommendations above provide essential components that should reflect in the documentation. However, the libraries must pay attention to balancing the depth and presentation of information such that it neither overwhelms the new developers nor bores the seasoned developers. One potential solution would be to use documentation tools with rich text features to improve readability and presentation.

6.2 For library software

Provide developer-friendly error messages. Designing effective error messages to provide feedback and assist developers is a longstanding research theme [33, 50]. However, many libraries, including `tpm2-tools`, produce incomprehensible error messages that are least useful and frustrate the developers. We have highlighted some of the examples in §5.1.3. A common concern among our participants was that the error messages lacked clarity and did not help resolve the problem. Despite that, seasoned developers have adapted their mental models of associating the ambiguous error messages with something more concrete based on their experience. Since new developers do not have this advantage, we suggest the library review existing error messages carefully. There is a wide range of design guidelines on developing human-centric error messages [20, 25, 46, 59], and we recommend the TPM community to adopt them. Similar to Green and Smith's principles "APIs should interact with the end user" and "APIs should be hard to misuse", we suggest revising the error messages to be more specific and provide constructive feedback, e.g., that suggests resolution or guides towards the right resources.

Provide concise output messages. Similar to error messages, our participants were unsatisfied with the output messages of `tpm2-tools` as they lacked clear feedback. Our recommendation is to review the output message formats to include the following necessary components. The output should show a *success message* that not only assures the developer that the commands have been executed without errors but also provides them *feedback* on whether the command has achieved its goal. Also, any interpretation and obvious additional steps (e.g., executing another command) to be taken must be clearly communicated. Additionally, the output should include a *concise description of meta-information* (e.g., of the objects created) if applicable. This recommendation is one way to make the API self-explanatory and ensure that principle "APIs should be easy to use" from Green and Smith is followed.

Utilize abstractions for sequential command execution. There are several occasions where multiple commands have to be executed sequentially. For example, for storing measurements of a file in a PCR (in task 2), one can take the hash of a file and then extend it to a PCR using `tpm2_hash` and `tpm2_pcrextend` commands, respectively. Alternatively, `tpm2-tools` offers an abstraction function called `tpm2_pcrevent` which combines hashing and extending in one go. We strongly believe that such abstractions provide convenience to the developers and make their code less error-prone by triggering a sequential execution of functions that might be missed otherwise. Unfortunately, while there are several abstraction functions available in the `tpm2-tools` library, they seem to be underutilized. We recommend that libraries promote and highlight the advantages of abstraction functions whenever available. Also, they should identify occasions where the order of command execution has to be preserved and provide abstractions for them. Even better would be to provide abstraction functions for common use cases.

Promote secure cryptographic primitives. We recommended in §6.1 that the documentation should include guidelines for picking security attributes. While security-conscious developers may benefit from that, it is common for developers to rely on the default options, especially while picking cryptographic primitives provided by the library. This finding is also reflected in Green and Smith's "Make defaults safe and unambiguous" principle. Thus, libraries must avoid supporting algorithms with known security vulnerabilities and set the defaults to the most secure primitive. If the insecure algorithms have to be supported for legacy reasons, their use should be discouraged, e.g., via warnings. When we examined cryptographic algorithms supported by the `tpm2-tools` library, we found that the library does not support insecure algorithms. However, the default options are set to the primitive with bare minimum security in most cases. We suggest that the library should consider updating defaults to the most secure option available. We also remind that support for cryptographic algorithms should be regularly audited, and support for insecure ones (if found) should be discontinued.

7 Discussion

Our results hint at the importance of threat models in the API ecosystem. We found that developers do not always feel forced to make security choices unless a threat model is given or they understand the threats to defend against. Documentation could bridge this gap by including common threat models, extensive background topics and secure coding examples built around common use cases. We believe that including threat models in the documentation of security-critical technologies would help developers cultivate intuitive security thinking and form correct mental models.

Similarly, naming conventions, along with text and format of errors and outputs, are crucial in invoking security thinking among the developers. For example, using familiar terminologies (such as `nonce`) as part of the commands could do the trick. However, in practice, software implementations often blindly borrow the names and texts defined in the standard specifications. This leaves minimal scope for improvement in the later stages. In this realm, we recommend considering usability and human factors already in the creation of the specifications. In particular, HCI usability experts should review the function names, text, and error and output formats. We believe API design principles [30] that are typically recommended at the implementation stage can also be applied while designing the specification.

Roadblocks faced by developers could easily make them gullible to commit mistakes, which could have serious consequences in the context of security-critical technologies. Libraries can learn from the common mistakes and address them in the software development life cycle. Modern code repositories provide an easy way of tracking end-user issues and creating automated workflows for integrating their solutions (e.g., as a feature) in software updates.

Our interaction with the developers drew attention to the importance of communities, as they often seek help from sources outside the software documentation. In particular, they rely on peer support and prefer immediate help (e.g., ready-made code and error resolution tips). Unfortunately, the generic venues for such help (e.g., Stack Overflow) lack useful content for niche technologies, such as TPM. In such cases, community forums of these technologies play a vital role. For example, we observed that *Tpm.dev* is one such community forum for TPM developers to discuss concerns and learn from each other, irrespective of experience or library preference. *Tpm.dev* has also recently started to share beginner-friendly resources [11], written by seasoned developers. We have discussed our results with *Tpm.dev* to encourage their initiative and hope they benefit from this study.

Limitations. One of the limitations of this work is the small number of participants. Despite that, we believe our results are generalizable due to the participants' expertise. Similar to Nielsen-like heuristic evaluations [42], we argue that the usability issue discovered for one participant is likely to indicate a general usability issue [31, 39, 45]. The other limitation is that our results are drawn only by observing the `tpm2-tools` library. Despite being allowed to choose a different TPM library supported by our study environment, all participants, including those familiar with multiple libraries, used `tpm2-tools`. However, we consider our results generalizable because the other libraries, similar to `tpm2-tools`, closely mirror the specifications. A comparison is provided in Appendix A. Our independent analysis confirmed that they do not provide more user-friendly function abstractions or naming. While the author of the `IBMTSS` library claims to provide a simpler interface , there is no empirical evidence to support their usability claims. Future studies can utilize our study environment and insights from this study to validate such claims or, even better, to conduct a quantitative and comparative analysis between different TPM libraries.

8 Conclusions

We conducted the first qualitative study targeting TPM library APIs and found that they are not developer-friendly. In particular, we identified specific areas where the TPM library APIs contain usability and security pitfalls and provided recommendations to fix them. Our contributions also include an open-source environment for TPM usability studies.

Our findings support those of past API usability and security studies. Additionally, we found new insights by studying the interesting combination of cryptographic and non-cryptographic features of TPM that is rarely seen in previously studied security APIs. Some of the identified pitfalls can be traced back to the TPM specification that forms the design basis for software implementation. Based on this observation, we highlight an important issue: any technology that follows standard specifications tends to accumulate usability pitfalls, well before its implementation, in the standards design. This is an opportunity for standardization bodies to prioritize usability by involving HCI experts in the design process. Previous studies have not highlighted this issue, and no usability frameworks have been explicitly created for the standards.We hope that our work inspires and steers future research in this direction.

Acknowledgments

We thank our anonymous reviewers for their insightful reviews and feedback that helped us improve the paper. We also thank our participants, without whom this study would not have been possible. We are grateful to Yoan Miche from Nokia Bell Labs for his support and discussions throughout this research project.

References

[1] Go-TPM. [Online]. https://github.com/google/go-tpm.

[2] IBM's TPM 2.0 TSS. [Online]. https://sourceforge.net/projects/ibmtpm20tss/.

[3] Judge0. [Online]. https://github.com/judge0/judge0.

[4] Protection details. [Online]. https://github.com/tpm2-software/tpm2-tools/blob/5.0/man/common/protection-details.md.

[5] SurveyJS. [Online]. https://github.com/surveyjs/survey-library.

[6] TPM study environment. [Online]. https://github.com/nokia/tpm-study-environment.

[7] tpm2-tools. [Online]. https://github.com/tpm2-software/tpm2-tools.

[8] tpm2_checkquote(1) tpm2-tools | General Commands Manual. [Online]. https://github.com/tpm2-software/tpm2-tools/blob/5.0/man/tpm2_checkquote.1.md.

[9] tpm2_create(1) tpm2-tools | General Commands Manual. [Online]. https://github.com/tpm2-software/tpm2-tools/blob/5.0/man/tpm2_create.1.md.

[10] tpm2_encryptdecrypt(1) tpm2-tools | General Commands Manual. [Online]. https://github.com/tpm2-software/tpm2-tools/blob/5.0/man/tpm2_encryptdecrypt.1.md.

[11] TPM.dev tutorials. [Online]. https://github.com/tpm2dev/tpm.dev.tutorials.

[12] wolfTPM. [Online]. https://github.com/wolfSSL/wolfTPM.

[13] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L Mazurek, and Christian Stransky. Comparing the usability of cryptographic apis. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 154–171. IEEE, 2017.

[14] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305. IEEE, 2016.

[15] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. How internet resources might be helping you develop faster but less securely. *IEEE Security & Privacy*, 15(2):50–60, 2017.

[16] Yasemin Acar, Christian Stransky, Dominik Wermke, Charles Weir, Michelle L Mazurek, and Sascha Fahl. Developers need support, too: A survey of security advice for software developers. In *2017 IEEE Cybersecurity Development (SecDev)*, pages 22–26. IEEE, 2017.

[17] Steven Arzt, Sarah Nadi, Karim Ali, Eric Bodden, Sebastian Erdweg, and Mira Mezini. Towards secure integration of cryptographic software. In *2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*, pages 1–13, 2015.

[18] Hala Assal and Sonia Chiasson. 'think secure from the beginning' a survey with software developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2019.

[19] Dejan Baca, Kai Petersen, Bengt Carlsson, and Lars Lundberg. Static code analysis to detect software security vulnerabilities-does experience matter? In *2009 International Conference on Availability, Reliability and Security*, pages 804–810. IEEE, 2009.

[20] Brett A Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, et al. Compiler error messages considered unhelpful: The landscape of text-based programming error message research. In *Proceedings of the working group reports on innovation and technology in computer science education*, pages 177–210. 2019.

[21] Stefan Berger and David Safford. SWTPM - software TPM emulator. [Online]. https://github.com/stefanberger/swtpm.

[22] Jan Camenisch, Liqun Chen, Manu Drijvers, Anja Lehmann, David Novick, and Rainer Urian. One tpm to bind them all: Fixing tpm 2.0 for provably secure anonymous attestation. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 901–920. IEEE, 2017.

[23] Stéphanie Delaune, Steve Kremer, Mark D Ryan, and Graham Steel. A formal analysis of authentication in

the tpm. In *International Workshop on Formal Aspects in Security and Trust*, pages 111–125. Springer, 2010.

[24] Stéphanie Delaune, Steve Kremer, Mark D Ryan, and Graham Steel. Formal analysis of protocols based on tpm state registers. In *2011 IEEE 24th Computer Security Foundations Symposium*, pages 66–80. IEEE, 2011.

[25] Paul Denny, James Prather, Brett A Becker, Catherine Mooney, John Homer, Zachary C Albrecht, and Garrett B Powell. On designing programming error messages for novices: Readability and its constituent factors. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–15, 2021.

[26] Andreas Fuchs, Christoph Krauß, and Jürgen Repp. Advanced Remote Firmware Upgrades Using TPM 2.0. In *ICT Systems Security and Privacy Protection*, pages 276–289. Springer, Cham, May 2016.

[27] William Futral and James Greene. *Intel® Trusted Execution Technology for Server Platforms: A Guide to More Secure Datacenters*. Apress, Berkeley, CA, 2013.

[28] Peter Leo Gorski and Luigi Lo Iacono. Towards the usability evaluation of security apis. In *HAISA*, pages 252–265, 2016.

[29] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Moeller, Yasemin Acar, and Sascha Fahl. Developers Deserve Security Warnings, Too: On the Effect of Integrated Security Advice on Cryptographic API Misuse. In *Proceedings of the Fourteenth Symposium on Usable Privacy and Security*, page 17, Baltimore, MD, USA, August 2018.

[30] Matthew Green and Matthew Smith. Developers are not the enemy!: The need for usable security apis. *IEEE Security & Privacy*, 14(5):40–46, 2016.

[31] Thomas Grill, Ondrej Polacek, and Manfred Tscheligi. Methods towards api usability: A structural analysis of usability problem categories. In *International conference on human-centred software engineering*, pages 164–180. Springer, 2012.

[32] Julie M Haney, Mary Theofanos, Yasemin Acar, and Sandra Spickard Prettyman. " we make it a big deal in the company": Security mindsets in organizations that develop cryptographic products. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 357–373, 2018.

[33] James J Horning. What the compiler should tell the user. In *Compiler Construction*, pages 525–548. Springer, 1974.

[34] Shohreh Hosseinzadeh, Bernardo Sequeiros, Pedro RM Inácio, and Ville Leppänen. Recent trends in applying tpm to cloud computing. *Security and Privacy*, 3(1):e93, 2020.

[35] Luigi Lo Iacono and Peter Leo Gorski. I do and i understand. not yet true for security apis. so sad. In *Proc. of the 2nd European Workshop on Usable Security, ser. EuroUSEC*, volume 17, 2017.

[36] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. Reliability and inter-rater reliability in qualitative research: Norms and guidelines for CSCW and HCI practice. *Proceedings of ACM Human-Computer Interaction*, 3(CSCW), November 2019.

[37] Microsoft. BitLocker (Windows 10) - Windows security. [Online]. https://docs.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-overview, December 2021.

[38] Microsoft. Secure the Windows boot process - Windows security. [Online]. https://docs.microsoft.com/en-us/windows/security/information-protection/secure-the-windows-10-boot-process, December 2021.

[39] Eduardo Mosqueira-Rey, David Alonso-Ríos, Vicente Moret-Bonillo, Isaac Fernández-Varela, and Diego Álvarez-Estévez. A systematic approach to api usability: Taxonomy-derived criteria and a case study. *Information and Software Technology*, 97:46–63, 2018.

[40] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. Jumping through hoops: Why do java developers struggle with cryptography apis? In *Proceedings of the 38th International Conference on Software Engineering*, pages 935–946, 2016.

[41] National Institute for Standards and Technology (NIST). Cryptographic standards and guidelines. [Online]. https://csrc.nist.gov/Projects/cryptographic-standards-and-guidelines, 2017. Accessed on: Nov 3, 2021.

[42] Jakob Nielsen. How to conduct a heuristic evaluation. [Online]. https://www.ingenieriasimple.com/usabilidad/HeuristicEvaluation.pdf, 1995.

[43] Daniela Oliveira, Marissa Rosenthal, Nicole Morin, Kuo-Chuan Yeh, Justin Cappos, and Yanyan Zhuang. It's the psychology stupid: how heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 296–305, 2014.

[44] Nikhil Patnaik, Joseph Hallett, and Awais Rashid. Usability smells: An analysis of developers' struggle with crypto libraries. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 245–257, 2019.

[45] Helen Petrie and Christopher Power. What do users really care about? a comparison of usability problems found by users and experts on highly interactive websites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2107–2116, 2012.

[46] James Prather, Raymond Pettit, Kayla Holcomb McMurry, Alani Peters, John Homer, Nevan Simone, and Maxine Cohen. On novices' interaction with compiler error messages: A human factors approach. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, pages 74–82, 2017.

[47] Elissa M Redmiles, Yasemin Acar, Sascha Fahl, and Michelle L Mazurek. A summary of survey methodology best practices for security and privacy researchers. Technical report, 2017.

[48] Nabil Schear, Patrick T. Cable, Thomas M. Moyer, Bryan Richard, and Robert Rudd. Bootstrapping and maintaining trust in the cloud. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 65–77, Los Angeles California USA, December 2016. ACM.

[49] Jianxiong Shao, Yu Qin, Dengguo Feng, and Weijin Wang. Formal analysis of enhanced authorization in the tpm 2.0. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pages 273–284, 2015.

[50] Ben Shneiderman. Designing computer system messages. *Communications of the ACM*, 25(9):610–611, 1982.

[51] TCG. Trusted Platform Module Library Part 1: Architecture. Trusted Platform Module Library Specification, Family 2.0 Level 00 Revision 01.59, The Trusted Computing Group, November 2019.

[52] TCG. TCG Feature API (FAPI) Specification. Technical Report Version 0.94 Revision 09, June 2020.

[53] TCG. TCG PC Client Platform Firmware Profile Specification. TCG PC Client Platform Firmware Profile Specification, Family 2.0 Level 00, Version 1.05 Revision 23, May 2021.

[54] TCG. TCG TSS 2.0 Enhanced System API (ESAPI) Specification. Technical Report Version 1.00 Revision 14, October 2021.

[55] TCG. TCG TSS 2.0 System Level API (SAPI) Specification. Technical Report Version 1.1 Revision 36, October 2021.

[56] The Trusted Computing Group (TCG). Tpm 2.0 library specification. [Online]. `https://trustedcomputinggroup.org/resource/tpm-library-specification/`, 2019. Accessed on: Nov 27, 2021.

[57] tpm2-software community. Remote attestation. [Online]. `https://tpm2-software.github.io/tpm2-tss/getting-started/2019/12/18/Remote-Attestation.html`.

[58] tpm2-software community. Tutorials. [Online]. `https://tpm2-software.github.io/tutorials/`.

[59] V Javier Traver. On compiler error messages: what they say and what they mean. *Advances in Human-Computer Interaction*, 2010, 2010.

[60] Jonathan Turner. Shape of errors to come. [Online]. `https://blog.rust-lang.org/2016/08/10/Shape-of-errors-to-come.html`.

[61] Juan Wang, Yuan Shi, Guojun Peng, Huanguo Zhang, Bo Zhao, Fei Yan, Fajiang Yu, and Liqiang Zhang. Survey on key technology development and application in trusted computing. *China Communications*, 13(11):70–90, 2016.

[62] Stephan Wesemeyer, Christopher JP Newton, Helen Treharne, Liqun Chen, Ralf Sasse, and Jorden Whitefield. Formal analysis and implementation of a tpm 2.0-based direct anonymous attestation scheme. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 784–798, 2020.

[63] Glenn Wurster and Paul C Van Oorschot. The developer is the enemy. In *Proceedings of the 2008 New Security Paradigms Workshop*, pages 89–97, 2008.

Appendices

A TPM library comparison

Table 3: Comparison between TPM Library APIs

	tpm2-tools [7]	**IBMTSS** [2]	**go-tpm** [1]	**wolfTPM** [12]
Programming language	Shell	C/Shell	Go	C
Date of creation	August 2015	May 2015	February 2018	January 2018
Usage statistics $^+$	507 GitHub stars 85830 downloads	11 Source-forge reviews 22136 downloads	391 GitHub stars dependency of 58 Go projects	124 GitHub Stars 75 downloads
Version used in our study *	v5.0	v1.5.0	v0.3.2	v2.0.0
Supports functions needed for study tasks	Yes	Yes	Yes	Yes
Follows the standards closely	Yes	Yes	Yes	Yes
Usability claims	No	Yes	No	No

* This was the latest release of the library at the time of the study design.
$^+$ As of May 19, 2022.

B Preliminary survey

- Have you worked on hardware-based security (trusted computing)?
- Have you developed using the Trusted Platform Module (TPM)? Development of any kind that requires hands-on skills
- What are you using/have you used TPM for? (As part of work; Hobby projects; School/University projects; Other (please specify))
- How many years of experience do you have with TPM development (Less than a year; 1-2 years; More than 2 years)
- Which version of TPM have you used? (TPM v1.x; TPM v2.0)
- Have you used any of the TPM simulators?
- Which TPM software stack (TSS) have you used or use? [All that apply] (tpm2-software (tpm2-tss/tpm2-tools); TPM 2.0 TSS by IBM, TPM TSS by Microsoft; go-tpm by Google; WolfTPM; Other (please specify))
- What have you used TPM for? Feel free to describe it in detail. If you have any open source projects, blog posts, or products, we would love to have a look at them. (free text)
- Are you interested in taking part in our study about the usability of TPM libraries? The study will include some simple tasks that involve using TPM libraries. Your efforts will be fairly compensated.
- Please provide us your email. We will use your email only for contacting you for the next phases of our study. If not, your email will be deleted immediately and permanently. The email provided by you is not used for any other purposes. (free text)
- Please tell us your name. We will use it only for addressing you when we send further communication about our study. (free text)

C Participant sampling and recruitment

First, we created a participant pool of the target population from personal contacts, mailing lists, forums, and code repositories. Then, we contacted them to take part in our preliminary survey via emails and social media (Twitter and LinkedIn). This survey collected participants' contact and demographic details, information about prior experience, TPM software libraries, their use of TPM and willingness to participate in the next part of the study. 36 out of 48 people expressed their interest in participating in the second part. We prioritized 34 participants who matched the criteria for our target population. We then invited them to participate via an email that contained: a reminder of their preliminary survey, a brief description of the goals, a link to their study environment, assigned library, compensation details (worth 100 €), and the approximate time needed to complete the study. 13 participants completed all the tasks, and the majority (N=11) used the `tpm2-tools` library. Our results are from a qualitative study of 9 of them whom we interviewed.

D Task descriptions

Task 1 (Track A): Asymmetric Encryption

Step 1a: Create Key. Your task is to create a secure asymmetric key of your choice for encryption purposes (e.g., for encrypting a file on your disk) using the TPM.
While creating the key, make sure the following conditions are met:

- The key should be exportable to other devices
- The key should be available across system reboots

Step 1b: Encrypt. Use the key you created in **Step 1a** to encrypt `path/to/file`

Step 1c: Reboot and decrypt

- First, reboot the environment by clicking the "*Reboot TPM*" button in the IDE.
- Then, decrypt the file you encrypted in step 1b.

Task 1 (Track B): Symmetric Encryption

Step 1a: Create Key. Your task is to create a password-protected symmetric key of your choice using the TPM.

While creating the key, keep in mind the following:

- You may have to repeatedly use this key for encryption and decryption

Step 1b: Encrypt. Use the key you created in **Step 1a** to encrypt the following:

- The string "TPMMakesMeFeelGreat"
- `path/to/file`

Step 1c: Decrypt

- Now, using the same key, decrypt the string <`enc.string`> from step 1b.
- Also, decrypt the file <`enc.file`>.

Step 1d: Cleaning the environment (Optional step) Other users may be using this environment in the future. If there is any code cleanup you would like to add, you can do so now.

Task 2: Storing Measurements

Step 2a: Measure and store in PCRs. Your task is to measure the file `path/to/file` and store the measurements using suitable PCRs.

The measurements stored in the PCRs are used in remote attestation to validate that the host machine has the correct configuration (based on `config.json`).

When storing the measurements, keep in mind the following:

- The attestation server may request the measurements from any PCR bank.

Step 2b: Read measurements. Read the contents of the PCRs (extended in step 2a). This is done to ensure that the measurements are recorded correctly.

Please make a note if you encounter any error(s).

Task 3: Securing Secrets

Step 3a: Store secret in the TPM. Your task is to store the secret string "workingWithTPMisAwesome" securely in the TPM.

The PCR allocation is as follows.
`PCR 0`: Core Root of Trust for Measurement,
`PCR 1`: Firmware, `PCR 2`: Kernel,
`PCR 3`: Config, and `PCR 4-23`: Unused.

While storing in the TPM, make sure that the following conditions are met:

- The secret should only be readable when the firmware has not been modified.
- The secret should not be modifiable after it has been written into the TPM.

Step 3b: Read secret. Read the secret (stored in step 3a) from the TPM. This is done to ensure the secret is stored correctly.

Please make a note if you encounter any error(s).

Task 4: Remote Attestation

Step 4a: Get quote. Your task is to get a quote of a machine for remote attestation.

The PCR allocation is as follows.
`PCR 0`: Core Root of Trust for Measurement,
`PCR 1`: Firmware, `PCR 2`: Kernel,
`PCR 3`: Config, and `PCR 4-23`: Unused.

While getting the quote, make sure that the following conditions are met:

- The quote should include the state of the kernel.
- The quote should be signed using an appropriate key to prove that the quote contents were generated by a legitimate TPM.

Step 4b: Verify quote. You are provided with the following files in your environment under the directory `path/to/directory`.
`q1.msg`: Quote file (signed by `key1`),
`q1.sig`: Signature file for `q1.msg`,
`key1.pub.pem`: Public part of `key1` in PEM format,
`key1.pub.tss`: Public part of `key1` in TSS format.

Your task is to verify that the quote contents of `q1.msg` can be used for remote attestation.

While verifying the quote, make sure that the following conditions are met:

- The content of the quote was generated by a TPM.
- The content of the quote has not been tampered with.

E TPM security features

Table E.4: List of cryptographic security features

Code	Description
C1	Symmetric $\longrightarrow$ Encryption
C2	Asymmetric $\longrightarrow$ Encryption
C3	Asymmetric $\longrightarrow$ Signing
C4	Hashing

Table E.5: List of non-cryptographic security features

Code	Description
NC1	Use of the TPM hierarchies
NC2	TPM key restrictions
NC3	Restrictions against TPM-internal states (e.g. PCRs, NVRAM, counters)
NC4	Restrictions against TPM-external states (e.g. password, signature, smart cards)
NC5	Session-based command or object authorization
NC6	PCR usage

F Questionnaires

Demographics

- How long have you been programming? (Less than a year; 1-2 years; 2-5 years; More than 5 years)
- How long have you been programming with TPMs? (Less than a year; 1-2 years; 2-5 years; More than 5 years)
- In which context do you usually deal with TPM related topics? (Big company (>250 employees); Small and medium enterprise (including startups); Academic institution; On my own free time after work; Other (Please specify))
- What is your occupation?
- Are you associated with <*library*> in any of the following capacities? [All that apply] (Creator; Maintainer; Regular contributor; I might have contributed something minor; End user; Other (Please specify))
- Do you have a computer security background?
- What is your highest level of education? (No formal education; Some high school; High school or equivalent; Technical or occupational certification; Some college course work completed/Associate degree; Bachelor's (or undergraduate) degree; Master's degree; Doctorate degree)
- Please tell us your gender. (Female; Male; I prefer not to say; Other (Please specify))
- Where are you from? (dropdown)
- What is your age (in years)? (<18; 18-29; 30-39; 40-49; 50-59; >60)
- We will contact you again with respect to compensation once the survey is over. Please leave your email address below. (free text)

Task-specific questionnaire

- How familiar are you with the task that you have just attempted? (Not at all familiar; Slightly familiar; Somewhat familiar; Moderately familiar; Extremely familiar)
- How frequently have you done tasks like this one? (Never; Rarely; Sometimes; Often; Frequently)
- How difficult was this task? (Very difficult; Difficult; Neutral; Easy; Very easy)
- Did you encounter any error messages? If yes:
  Please rate your agreement to the following statements on a scale from 'strongly disagree' to 'strongly agree.' (Strongly disagree; Disagree; Neutral; Agree; Strongly agree)
  - The error or warning messages were helpful in improving my answers.
  - The error or warning messages were helpful in making secure choices, e.g. while selecting parameters for specific library functions.
- Did you manage to complete all the steps in this task?
  If yes:
  - I think my code snippet for this task is correct. (Strongly disagree; Disagree; Neutral; Agree; Strongly agree)
  - I think my code snippet for this task is secure. (Strongly disagree; Disagree; Neutral; Agree; Strongly agree)
  - Did you refer to any of the following resources while completing the task? [All that apply]
    - Official resources (Official library documentation; TCG Technical standards; I did not use official resources)
    - Additional resources (Mailing lists or community forums of the library that you used; Third-party/generic TPM forums (e.g., stack overflow, social media groups); Blogs, walk-through and hands-on guides; Training and workshop materials; Personal notes; I did not use additional resources; Others (please specify))
  - Did you observe anything interesting when completing the task? If yes, please describe it. (free text)

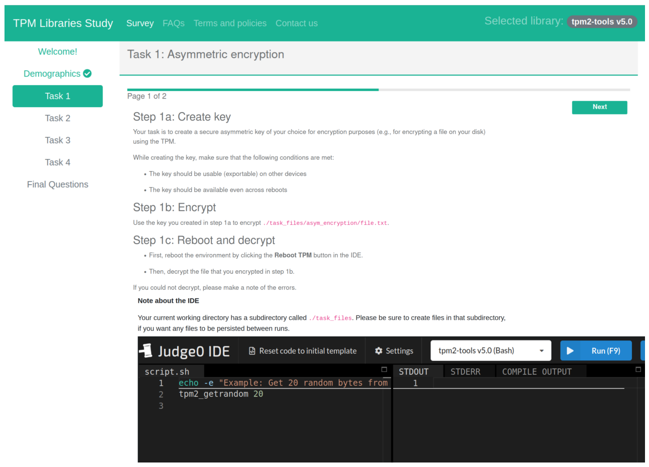
  If no:
  - Why do you think you could not complete all the steps? [All that apply] (I did not know how to do it; I could not find suitable resources to help me complete the task; I tried and gave up midway because the steps were too difficult or time-consuming; The description was not understandable; The task did not interest me; Other (please describe))
    If "I did not know how to do it" or "I could not find suitable resources to help me complete the task:
  - Did you refer to any of the following resources while completing the task? [All that apply]
    - Official resources (Official library documentation; TCG Technical standards; I did not use official resources)
    - Additional resources (Mailing lists or community forums of the library that you used; Third-party/generic TPM forums (e.g., stack

overflow, social media groups); Blogs, walk-through and hands-on guides; Training and workshop materials; Personal notes; I did not use additional resources; Others (please specify))

Exit questionnaire

- In general, which of the following do you refer to for your regular TPM-related activities? (Official resources only; Additional resources only; Mostly official resources, but sometimes additional resources; Mostly additional resources, but sometimes official resources)
- Overall, I would rate the user-friendliness of <*library*> as (Worst imaginable; Awful; Poor; Fair; Good; Excellent; Best imaginable)
- How satisfied are you with the <*library*> documentation? (Not at all satisfied; Slightly satisfied; Moderately satisfied; Very satisfied; Extremely satisfied)
- How do you rate the quality of the <*library*> documentation? (Very poor; Poor; Acceptable; Good; Very good)
- How frequently do you refer to additional resources? (Never; Rarely; Sometimes; Often; Frequently)
- If you refer to additional resources, what do you think is the reason? [All that apply] (<*library*> documentation is not clear; <*library*> documentation is incomplete/work-in-progress; <*library*> documentation does not add much beyond what is already there in the standards; There are no examples (code snippets or pseudo-code) of common use cases; Background information (e.g., TPM or programming concepts) is missing)
- Is there anything else you want to tell us about <*library*>? (free text)

G Technical details of the study environment


TPM Libraries Study | Survey | FAQs | Terms and policies | Contact us | Selected library: tpm2-tools v5.0

Welcome! | Demographics ✓ | Task 1 | Task 2 | Task 3 | Task 4 | Final Questions

Task 1: Asymmetric encryption

Page 1 of 2 | Next

Step 1a: Create key

Your task is to create a secure asymmetric key of your choice for encryption purposes (e.g., for encrypting a file on your disk) using the TPM.

While creating the key, make sure that the following conditions are met:

- The key should be usable (exportable) on other devices
- The key should be available even across reboots

Step 1b: Encrypt

Use the key you created in step 1a to encrypt `./task_files/asym_encryption/file.txt`.

Step 1c: Reboot and decrypt

- First, reboot the environment by clicking the **Reboot TPM** button in the IDE.
- Then, decrypt the file that you encrypted in step 1b.

If you could not decrypt, please make a note of the errors.

Note about the IDE

Your current working directory has a subdirectory called `./task_files`. Please be sure to create files in that subdirectory, if you want any files to be persisted between runs.

```
echo -e "Example: Get 20 random bytes from
tpm2_getrandom 20
```

Figure G.1: Interface of the study environment

Our main goal for the study environment was to present tasks and questionnaires in a single platform, as an online Integrated Development Environment (IDE), to give a seamless user experience. Also, we wanted to control both the frontend and backend to tweak the user interfaces and capture crucial details, such as partial and intermediate submissions. The off-the-shelf solutions neither satisfied all our requirements nor supported TPM functionalities. Hence, we built the study environment from scratch using open-source components.

We used the `SurveyJS` library [5] to build a survey app for the questionnaires. We then integrated the self-hosted version of the `Judge0` online IDE [3] with a *TPM emulator* [21] in the backend. Please note that, unlike real TPMs, the emulators do not include manufacturer-certified keys in the endorsement hierarchy. However, they are a helpful utility to test TPM functionality without having the hardware and in remote studies. We supported coding with widely used TPM libraries: tpm2-tools [7], IBMTSS [2], go-tpm [1], and wolfTPM [12].

The backend also contained a `MongoDB` database that collected responses to the tasks and questionnaires. We bundled the survey app, IDE, TPM emulator, and the database into a docker image and hosted it on our servers.

H Interview script

We present a skeletal structure of our interviews along with some of the questions in this section. Please note that, for each participant, we had created prompts using their code snippets and questionnaire responses that we found to be worth discussing in-depth. We used probes based on them, which differ for each participant and cannot be generalized; hence, we excluded them from the script presented below.

H.1 Introduction

Mutual introduction and reminder about the study

Ice breaker

- What do you use TPMs for?
- What are the common use cases of TPMs for you?
- How technology like TPM is contributing the field of security?
- Where is TPM useful and where is TPM not useful?

Confirming ecological validity

- How did you feel about the study environment and logistics?
- What about your prior experience or familiarity with such studies involving coding tasks, mainly the IDE?
- Are there any troubling components in our study that you want to highlight?
- Do you have any suggestions for improvement?

H.2 Task- and questionnaire-specific observations

Task-specific

General approach to TPM programming:

- How did you figure out what commands to use?
- Describe your process when starting a task?
- How did you search for relevant information?
- What resources helped you get started with the tasks?

Task 1: Symmetric/Asymmetric encryption

- How did you pick the cryptographic algorithms for creating the keys?
- Could you describe any examples that might have helped you when choosing other parameters? e.g., key length.

Task 2 :Storing measurements

- How did you select which PCR to extend with the measurements of the configuration file?

Task 3: Securing secrets

- Could you describe how and where is this secret stored?
- Could you describe any other ways to complete this task?
- Knowing now that there are other approaches, do you see any advantages or disadvantages of your approach over others?

Task 4: Remote attestation

- How do you verify the quotes are good to be used for remote attestation?
- Can you describe quote verification process?
- How do you verify the quote was generated by a TPM and it has not been tampered with?
- How do you confirm that the quote is valid?
- How do you think the verification could be simplified?

Questionnaire-specific

Correctness and security

- How did you verify that the task conditions were met?
- Why do you think your answer is secure?
- How did you verify you answer is secure?
- How do you know the defaults (or chosen parameters) are secure?
- Did you do any extra checks or referred somewhere?

H.3 General discussion

- Why do you think you had to look for help outside the official documentation?
- Why do you think the non-official resources are more reliable and useful than official resources?
- How do you think TPM library documentation and TPM standards could be written to compliment each other?
- How do you think the library can be improved?
- How do you think developers can contribute to improve the library further?
- Is there anything that you want to tell us regarding your experience about the library?