Facilitating the Use of TPM Technologies through S&D Patterns

Sigrid Gürgens Carsten Rudolph
Fraunhofer - Institute for Secure Information Technology SIT
Rheinstrasse 75, 64295 Darmstadt, Germany
{guergens,rudolphc}@sit.fraunhofer.de

Antonio Maña Antonio Muñoz
Computer Science Department
University of Málaga, Spain
{amg,anto}@lcc.uma.es

Abstract

Trusted platform modules (TPMs) can provide a variety of security functionalities. However, the TPM specification is highly complex and the deployment of TPM-based solutions is a difficult and delicate task. In this paper we propose the use of security patterns to specify TPM-based security solutions. The refined notion of security patterns developed in the SERENITY research project allows us to produce precise specifications of TPM-based solutions for particular security goals. This approach makes TPM technology available to system engineers without in-depth knowledge of trusted computing specifications.

1 Introduction

Security patterns have successfully been used to describe security solutions in a way that these solutions are made available to system engineers not being experts on security engineering [1, 2, 5, 6, 7, 8, 10]. Such security patterns are usually informally described using either plain text or use semi-formal languages enhanced with graphical visualisations. Recently, in the SERENITY research project [9], the notion of security patterns was extended to exact specifications of re-usable security mechanisms for AmI (ambient intelligence) systems. These security patterns also include information on the properties satisfied by the solution and on the context conditions to be fulfilled.

This type of security patterns has shown to be particularly useful for the description of security solutions relying on the trusted computing platform (TPM) as specified by the Trusted Computing Group [3]. Considering the complexity of the TPM standard it is obvious that only experts on trusted computing are able to develop TPM-based solutions for any non-trivial requirement. Therefore, in order to make TPM-based solutions available for wide-scale use in software development, one possible way is to describe re-usable solutions in terms of security patterns. High-level patterns using plain text are less suitable because most of the complexity lies in the choice of TPM commands and the details of the calls for a particular security service.

This paper uses a relatively simple example of a TPM-based security solution to demonstrate and motivate the refined notion of security patterns developed in SERENITY. More complex security patterns based on TPM technology are developed in SERENITY. Among others, solutions are explored that use certified migration keys to control the migration of data between a set of platforms.

2 An example

Let us consider the following scenario. A Medical Centre takes care of some of their patients who, although not yet completely cured, can live at home. Let us consider patient Bob who, after having been visited by a doctor, needs assistance for getting him some medication the doctor has prescribed. The doctor issues an electronic prescription and sends it to the medical centre. Here the social worker Alison takes over the actual task of getting the medication at a pharmacy and delivering it to Bob. For this, the electronic prescription is stored on Alison's PDA (note: we use the term *personal digital assistant (PDA)* to denote any type of portable device suitable for the tasks described in the scenario), and then the PDA is connected to the pharmacy's PC in order to transfer the prescription.

Patient data needs to be confidential, only authorized persons are allowed to access. In our example only the issuing doctor, some staff of the medical centre, Alison and the pharmacist are allowed access to Bob's prescription. For data transfer via the internet there are various different mechanisms in place today to protect data confidentiality. However, the requirement has to be met even in the case where Alison looses her PDA or gets it stolen along with Bob's prescription.

Possible mechanisms to protect the confidentiality of data stored on a device are

- access control provided by the device's operating system,
- software encryption,
- use a device protected by a Trusted Platform Module (TPM) to encrypt the data and bind it to the TPM.

The first two protection mechanisms are in principal suitable to prevent data access of attackers "from outside". However, an attacker who has got hold of the actual device can for example boot the device with an operating system that allows access to all data, he/she can monitor the encryption application to find out the decryption key or the password protecting the key, and can apply all other kinds of attacks. In this paper we concentrate on the third solution relying on the TPM. In this case, the attacker cannot directly benefit from physical access to the devices. He/she has no means to attack a TPM and data protected by a TPM as we will explain in the next section. Interestingly, this solution additionaly provides confidentiality of the communication line.

3 Introduction to TPM

A TPM usually is implemented as a chip integrated into the hardware of a platform (such as a PC, a laptop, a PDA, a mobile phone). A TPM owns shielded locations (i.e. no other instance but the TPM itself can access the storage inside the TPM) and protected functionality (the functions computed inside the TPM can not be tampered with). The TPM can be accessed directly via TPM commands or via higher layer application interfaces (the Trusted Software Stack, TSS).

The TPM offers two main basic mechanisms: It can be used to prove the configuration of the platform it is integrated in and applications that are running on the platform, and it can protect data on the platform (such as cryptographic keys). For realizing these mechanisms, the TPM contains a crypto co-processor, a hash and an HMAC algorithm, a key generator, etc.

In order to prove a certain platform configuration, all parts that are engaged in the boot process of the platform (BIOS, master boot record, etc) are measured (i.e. some integrity measurement hash value is computed), and the final result of the accumulated hash values is stored inside the TPM in a so-called Platform Configuration Register (PCR). An entity that wants to verify that the platform is in a certain configuration requires the TPM to sign the content of the PCR using a so-called Attestation Identity Key (AIK), a key particularly generated for this purpose. The verifyer checks the signature and compares the PCR values to some reference values. Equality of the values proves that the platform is in the desired state. Finally, in order to verify the trustworthiness of an AIK's signature, the AIK has to be accompanied by a certificate issued by a trusted Certification Authority, a so-called Privacy CA (P-CA). Note that an AIK does **not** prove the identity of the TPM owner.

Keys generated and used by the TPM have different properties: Some (so-called non-migratable keys) can not be used outside the TPM that generated them, some (like AIKs) can only be used for specific functions. Particularly interesting is that keys can be tied to PCR values (by specifying PCR number and value in the key's public data). This has the effect that such a key will only be used by the TPM if the platform (or some application) configuration is in a certain state (i.e. if the PCR the key is tied to contains a specific value). In order to prove the properties of a particular key, for example to prove that a certain key is tied to specific PCR values, the TPM can be used to generate a certificate for this key by signing the key properties using an AIK.

For requesting a TPM to use a key (e.g. for decryption), the key's authorization value has to be presented to the TPM. This together with the fact that the TPM specification requires a TPM to prevent dictionary attacks provides the property that only entities knowing the key's authorization value can use the key.

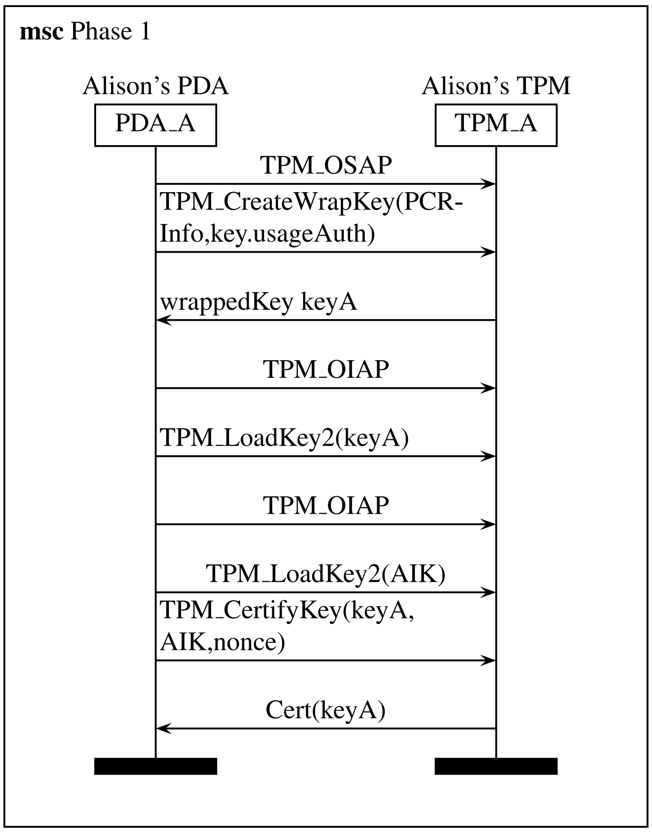
Non-migratable keys are especially useful for preventing unauthorized access to some data stored on the platform. Binding such a key to specific PCR values and using it to encrypt data to be protected achieves two properties: The data can not be decrypted on any other platform (because the key is non-migratable), and the data can only be decrypted when the specified PCR contains the specified value (i.e. when the platform is in a specific secure configuration and is not manipulated).

4 Using TPM functionality to prevent unauthorized access to data

In this section we will explain a solution that both provides confidentiality of the communication line and prevents unauthorized access to Bob's prescription stored on Alison's PDA. We assume that the PDA is protected by a TPM, hence our solution makes use of TPM functionality. It consists of three phases. In the first phase, the medical centre requests from Alison (i.e. from Alison's TPM) a public key with certain properties. In particular, the key shall be bound to the TPM of Alison's PDA (i.e. the key shall be non-migratable) and it shall be bound to specific PCR values (assuring that the PDA has not been tampered with). The key's properties are proven using a TPM generated certificate. In the second phase, the medical centre uses this key to encrypt Bob's prescription and sends the ciphertext to Alison's PDA. The encrypted prescription is then stored on Alison's PDA. Finally, in the third phase Alison presents the key's authorization data in order to decrypt Bob's prescription and transfer it to the pharmacy. The key's properties ensure that nobody else but Alison can perform the decryption. In the following we explain the process in more detail.

4.1 Phases 1 and 2 - setup and encryption

First, the medical centre requires a key from Alison's PDA that is non-migratable and bound to specific PCR values. The following message sequence chart (msc) shows the subsequent communication between Alison's PDA and the PDA's TPM for generating this key.



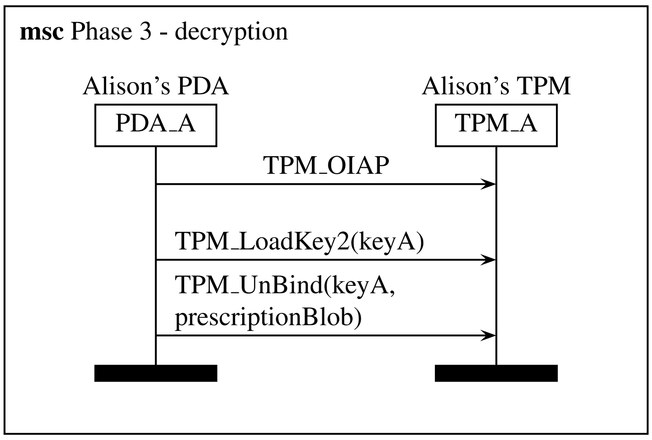
The actions are the following:

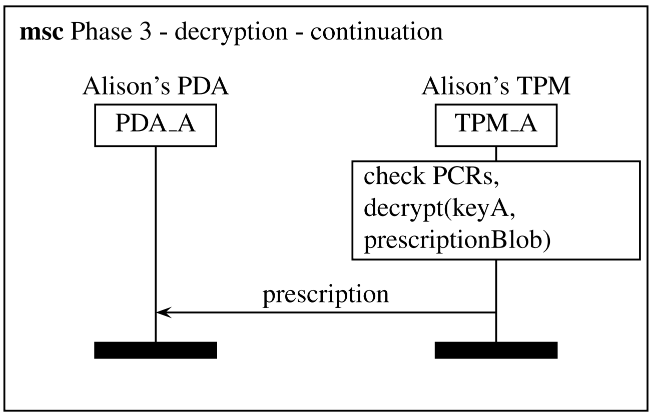
1. Alison's PDA starts an object specific authorization session OSAP.
2. With TPM_CreateWrapKey Alison's PDA requires the TPM to generate a non-migratable key with the PCR values requested by the medical centre. The command contains the key's usage authorization data (we do not discuss here where the key's authorization data comes from, it can for example be presented by Alison or by Alison's PDA).
3. The TPM generates $keyA$ and returns the key blob.
4. Alison's PDA then requests its TPM to generate a certificate for $keyA$:
   - It starts an object independent authorization protocol with TPM_OIAP.
   - Then it loads $keyA$ into the TPM.
   - It starts another OIAP session.
   - It loads an AIK into the TPM.
   - With TPM_CertifyKey it then lets the TPM generate a certificate for $keyA$ using the AIK. (Again we do not discuss where the AIK's authorization data comes from.)
   - The TPM returns the certificate.

Alison's PDA now sends this certificate and the AIK certificate issued by the P-CA to the medical centre which in turn verifies the certificates and checks in particular that the requested key has the required properties (non-migratable, bound to specific PCR values). The medical centre then uses the public part of the key to encrypt Bob's prescription and sends the ciphertext to Alison's PDA. This ensures confidentiality of Bob's data during communication between the centre and the PDA.

4.2 Phase 3 - data retrieval

The encrypted description is stored on Alison's PDA. When Alison wants to transfer the prescription to the pharmacy it needs to be decrypted which has to be done by the TPM. The next figure describes the necessary commands exchanged between Alison's PDA and the PDA's TPM. Again, we do not discuss which entity provides key authorization data necessary for the process.


msc Phase 3 - decryption
Alison's PDA
Alison's TPM
PDA_A
TPM_A
TPM_OIAP
TPM_LoadKey2(keyA)
TPM_UnBind(keyA, prescriptionBlob)


msc Phase 3 - decryption - continuation
Alison's PDA
PDA_A
Alison's TPM
TPM_A
check PCRs, decrypt(keyA, prescriptionBlob)
prescription

1. Alison's PDA starts an OIAP session.
2. The PDA then loads $keyA$ into the TPM.
3. With TPM_UnBind Alison's PDA lets the TPM decrypt the prescription using the private part of $keyA$.
4. TPM_A checks that the PCR values for PDA_A correspond to those the key is tied to and then uses the key to decrypt the prescription blob.
5. TPM_A returns the prescription, which can then be forwarded by Alison's PDA to the pharmacy.

As explained in Section 3, the key's properties prevent unauthorized access to Bob's data during storage on Alison's PDA: Since the key is non-migratable, only the PDA's TPM can decrypt the data. Binding the key to specific PCR values ensures that the TPM only decrypts the data while the PDA is not manipulated. Finally, assuming that Alison does not reveal the key authorization data to anybody, the key will only be used by the TPM after authorization is given by Alison.

5 Capturing the TPM solution

Security patterns represent a suitable means to make TPM functionality available to application developers. However, in order for patterns to serve our purposes, we need them to contain at least the following information:

1. the security requirement it addresses (in our case to prevent unauthorized access to data sent to and then stored on a TPM protected device);
2. the assumptions on the environment that need to hold, both before and during the operation (for example, the TPM has to be active);
3. the roles of the entities involved (in our case these are the medical centre and Alison's PDA);
4. the precise services offered by the pattern for each of the roles;
5. the parameters that must be instantiated;
6. any additional information that may help in the selection and application of the pattern; and
7. information regarding the pattern itself (source, version, certificates, etc).

With this information, application developers can decide whether the security requirements fulfilled by the pattern match the ones needed by the application and whether the environment the application shall run in meets the assumptions specified in the pattern.

5.1 SERENITY patterns

In the project SERENITY, security and dependability (S&D) patterns are described using a specification language that meets the above requirements. These patterns support in particular applications that will run in unpredictable and dynamic contexts. To this end, SERENITY patterns are described using three modelling artefacts:

The first SERENITY artefact is called S&D Class. S&D Classes provide homogeneous mechanisms to access S&D services and allow developers to delay the decision about the most appropriate solution to runtime, when the information required to make a sound decision (about the context, type and capabilities of other parties, etc.) is available. In our case, developers can use the S&D Class ("SimpleConfidentialStorage.cen.eu") that represents confidentiality services and includes a high-level interface (with function calls such as SetupConfidentialStorage, AcceptConfidentialStorage, StoreConfidential, etc) which hides the complexity of the TPM technology.

The second of the SERENITY artefacts, called S&D Pattern, is used to represent abstract solutions such as authentication protocols, encryption algorithms, or TPM functionality. The main purpose of this artefact is to guarantee the interoperability of different implementations of a solution. S&D patterns achieving the same requirement can refer to the same S&D Class. All patterns belonging to one S&D Class provide compatible interfaces which enables their dynamical selection and use. Patterns contain all information necessary to select solutions appropriately addressing certain security requirements. The pattern specifying our example solution contains, among other details, an interface section with all function calls that the solution uses (in particular it contains all TPM command calls). Furthermore a so-called "Interface Adaptor" specifies how each of the Class function calls is translated to a sequence of pattern function calls. In our case, the Interface Adaptor contains for example the following:

```
RetrieveConfidential(d,c) ::= {
TPM_OIAP
TPM-LoadKey2(kA)
TPM_UnBind(kA,Ciphertext,d)}
```

The third artefact provided by SERENITY is the S&D Implementation, which represents specific realizations of an S&D Solution. All S&D Implementations of an S&D Pattern must conform directly to the interface, monitoring capabilities, and any other aspect described in the S&D Pattern. However, they also have differences, such as the specific context conditions that are required, performance, target platform, programming language or any other feature not fixed by the pattern. Implementations that realize the TPM pattern can for example defer in the platform they shall run on and in the platform's operating system.

These three artefacts provide a precise description of S&D Solutions that supports both development time and runtime processes. All SERENITY artefacts are stored in a SERENITY library and thus made available to application developers. Selection of the artefacts and their integration into applications is supported by the SERENITY Runtime Framework (SRF), a suite of tools to support the automated management of S&D Solutions based on our modeling artefacts [4].

5.2 SERENITY operation

One interesting aspect to remark is the use of the previously described artefacts. Let us illustrate it using our scenario. The developers of the healthcare system identify the requirement that patient data needs to be confidential not only during transmissions but also when stored in the social worker's PDA. After searching some available SERENITY online libraries, they identify the S&D Class SimpleConfidentialStorage.cen.eu as addressing this requirement adequately. Hence the class's function calls are integrated into the application. Now the application developers have two choices: to select an appropriate pattern and implementation of the class and integrate it into the application during development time, or to leave this decision open. They decide to delegate the pattern and implementation selection to the SERENITY Runtime Framework (SRF) of the PDA.

At runtime, in order to realize the services of the S&D Class, the SRF identifies the best available S&D Implementation and its corresponding S&D Pattern, according to the current context and the preconditions included in both artefacts. In our case, the best option is an implementation of the "`TPMConfidentialStorage.serenity-project.org`" S&D Pattern, which uses the TPM-based solution described in section 4.

The SRF activates the solution (which may require initialization steps as described in the artefacts) and provides a reference to it to the healthcare application. The SRF uses the Interface Adaptor provided by the S&D Pattern to translate the calls made by the application to the calls provided by the selected solution. In this way the healthcare applications can transparently access the S&D services it needs without knowing in advance which specific solution is used to provide them.

6 Conclusions

In this paper we have shown how security patterns, and in particular SERENITY patterns can contribute to the use of complex technologies and devices such as TPM. Moreover, we have shown how these technologies can be used both at development time and at run time. Our current work is focused on three main lines: the development of mechanisms and tools to analyse solutions and produce the information contained in the pattern, on the generality and flexibility of the structure and contents of the modelling artefacts, and on the development the SERENITY Runtime Framework.

References

[1] E. Fernandez. Security patterns. In *Procs. of the Eigth International Symposium on System and Information Security - SSI2006*, volume Keynote talk, Sao Jose dos Campos, Brazil, November 2006.

[2] E. Fernandez and P. Rouyi. A pattern language for security models. In *Pattern Languages of Program Design. PLoP01.*, 2001.

[3] T. C. Group. TCG TPM Specification 1.2. www.trustedcomputing.org, 2006.

[4] A. Maña, D. Presenza, A. Piñuela, D. Serrano, P. Soria, and D. Sotiriou. Specification of SERENITY architecture. Technical Report Deliverable A6.D3.1, SERENITY Project, 2006.

[5] S. Romanosky. Security design patterns part 1, v1.4, 2001.

[6] F. Sanchez-Cid, A. Mu D. Serrano, and M. Gago. Software engineering techniques applied to ami: Security patterns. In A. Mad V. Lotz, editors, *In Proceedings of the First International Conference on Ambient Intelligence Developments. Developing Ambient Intelligence*, pages 108–124. Springer Verlag, September 2006.

[7] M. Schumacher, E. Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad. *Security Patterns - Integrating Security and Systems Engineering*. John Wiley Sons, 2005.

[8] M. Schumacher and U. Roedig. *Security engineering with patterns*. Springer Verlag, 2001.

[9] SERENITY. System engineering for security and dependability. IST project, funded by the EC. http://www.serenity-project.org/, 2006.

[10] R. Wassermann and B. Cheng. Security patterns. Technical Report MSU-CSE-03-23, Department of Computer Science, Michigan State University, Aug. 2003.