Facilitating the Use of TPM Technologies Using the Serenity Framework

Antonio Muñoz and Antonio Maña

Computer Science Department
University of Málaga
{amunoz,amg}@lcc.uma.es

Abstract. Trusted platform modules (TPMs) specification is highly complex and therefore the deployment of TPM –based security solutions is equally complicated and difficult; although they can provide a wide range of security functionalities. In order to make TPM technology available to system engineers without their needing to have in-depth knowledge of trusted computing specifications we propose, in this paper, to develop an approach using security patterns to specify TPM-based security solutions. Ideally suited to producing precise specifications of TPM –based solutions for certain security goals are the refined notions of security patterns developed in the SERENITY research project.

1 Introduction

Security patterns, described informally using either plain text or semi formal languages with graphical visualisations, have successfully been used to describe security solutions in such a way as to make them available to system engineers who are not necessarily experts in security engineering [1,2,5,6,7,8,10]. In the SERENITY research project [9] the notion of security patterns has been extended to concrete specifications of re-usable security mechanisms for AmI (ambient intellligence) systems. Also included is information on the properties satisfied and context conditions. The Trusted Computing Group [3] specifies the trusted computing platform (TPM) and so these types of security solutions have proven to be particularly useful for describing security solutions reliant on TPMs. Usually because of the complexity of their standard, only experts on trusted computing are able to develop TPM-based solutions for non trivial requirements. One way to make TPM-based solutions available for wide-scale use in software development is to describe re-usable solutions in terms of security patterns. As most of the complexity is in the selection of TPM commands and the details of the calls to a particular security service this makes high-level patterns using plain text less suitable. So in this paper we use a fairly simple example of a TPM-based security solution to demonstrate and motivate the refined notion of security patterns developed in SERENITY. Also developed in SERENITY are more complex security patterns are studied for example certified migration keys to control the migration of data between a set of platforms.

© Springer-Verlag Berlin Heidelberg 2011

One area where the need for confidentiality and mechanisms to protect data confidentiality is amply demonstrated is in the field of medicine as the following scenario shows. A patient continues their treatment at home whilst still being monitored by their medical centre. Let us imagine that following a doctor's visit the patient requires a prescription to be filled but cannot go to the pharmacy and therefore the doctor issues an electronic prescription which he sends to the medical centre. Once it reaches the centre it is dealt with by a social worker who is responsible for getting it filled and delivering it to the patient. The prescription is therefore stored in the social worker's PDA (personal digital assistant used to denote any portable device able to perform this task) and sent from there to the pharmacy's PC. Obviously the patient's details are confidential with only those authorised able to access the prescription; in this case the doctor, relevant staff at the medical centre, the social worker and naturally the pharmacist. Imagine a case where the social worker loses their PDA, there are mechanisms in place to protect confidentiality; as there are for data transfer via the Internet. Possible solutions are;

- Access control provided by the device's operating system
- Software encryption
- A device protected by a TPM (Trusted Platform Module) to encrypt data and bind it to the TPM.

At first glance the first two mechanisms would be suitable for preventing attacks "from outside" but imagine the attacker actually has the device in their possestion, they could gain access to the whole device and therefore study the encryption application to find out the decryption key or password and then apply it to all manner of attacks. We therefore find the third solution, relying as it does on a TPM to be the most effective, as the owner of the device does not provide an advantage to a possible attacker. They find themselves powerless to attack a TPM as we will go on to show.

2 Introduction to TPM Technology

A TPM is usually 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 verifier 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 an 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.

To request a TPM to use a key (e.g. for decryption), the key's authorisation 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 authorisation value can use the key.

Non-migratable keys are especially useful for preventing unauthorised access to some data stored on the platform. Binding such a key to specific PCR values and using it to encrypt the 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).

3 An Introduction to the Serenity Framework

This section gives an overview of the Project Framework. The main objective of Serenity is to provide a framework for the automated treatment of security and dependability issues in AmI scenarios. For this purpose the project is two-folded: (i) capturing the specific expertise of the security engineers in order to make it available for automated processing, and (ii) providing run-time support for the use and the monitoring of these security and dependability mechanisms. These two cornerstones have been deployed by means of:

- A set of S\D modeling artefacts (S\D artifacts, for short), used to model security and dependability solutions (S&D solutions) at different levels of abstraction. S&D solutions are isolated components that provide security and/or dependability services to applications. The use of different levels of abstraction responds to the need of different phases of the software development process. These artefacts are supported by an infrastructure created for the development and the validation of S&D solutions. This infrastructure includes concepts, processes and tools used by security experts for the creation of new S&D solutions ready for automatic processing.

- A development framework. Under the name of Serenity Development-time Framework (SDF) there is an infrastructure that supports the development of secure applications. These secure applications, called Serenity-aware applications, are supported by S&D solutions, consequently, they include references to the aforementioned S&D artefacts.
- A run-time framework, called Serenity Run-time Framework (SRF). The SRF provides support to applications at run-time, by managing S&D solutions and by monitoring the systems' context. A further description of the SRF can be found at [4].

Once infrastructural pieces have been described, the rest of this section explains how to use S&D modelling artefacts to bridge the gap between abstract S&D solutions and actual implementations of these S&D solutions. Interested readers could refer to Section 3 in order to find information on how the SRF supports applications at run-time.

Back to the abstractions, five main artefacts are provided to achieve a logical way to represent S&D solutions in the Serenity project: S&DClasses, S&DPatterns, IntegrationSchemes, S&DImplementations and ExecutableComponents. These artefacts, depicted in figure 1, represent S&DSolutions using semantic descriptions at different levels of abstraction. The main reason for using different artefacts, each one addressing an abstraction level, is that, by doing this, it is possible to cover the complete life cycle of secure applications, especially at development and run-time phases.

- S&DClasses represent abstractions of a set of S&DPatterns, characterized for providing the same S&D Properties and complying with a common interface. This is one of the most interesting artefacts to be used at development time by system developers. The main purpose of this artefact is to facilitate the dynamic substitution of S&D solutions at run-time, while facilitating the development process. Applications request S&D Solutions to the SRF to fulfill a set of S&D requirements. Usually, these requirements are hard coded by means of calls to S&DClasses or S&DPatterns interfaces. At run-time all S&DPatterns (and their respective S&DImplementations, described below) belonging to the same S&DClass, will be selectable by the SRF automatically.
- S&DPatterns are precise descriptions of abstract S&D solutions. These descriptions contain all the information necessary for the selection, instantiation, adaptation, and dynamic application of the solution represented by the S&DPattern. S&DPatterns describe the security pattern's functionalities and how to use them in a structured way. The most interesting elements of the S&DPattern structure are: (i) The pattern interface, describing the functionalities provided and how to use them; (ii) references to the S&DClasses the S&DPattern belongs to; and (iii) the ClassAdaptor, describing how to adapt the S&DPattern interface to the S&DClass interface. S&DPatterns represents monolithic isolated S&D solutions, but a special type of S&D artefact called \textit{Integration Scheme (IS)} also exists, which consists on an S&D solution at the same level than S&DPatterns. They represent S&D solutions

that are built by means of combining other S&DPatterns. At Serenity-aware application development time, Integration Schemes are used similarly as S&DPatterns are. However, they differ in their development process, presented in \cite{Antonio2006}. All along this paper we use the notion of S&DPatterns to refer to S&DPatterns and Integration Schemes indistinctly.
- S&DImplementations are specification of the components that realize the S&D solutions. S&DImplementations are not real implementations but their representation/description. An S&DImplementation describes an implementation of an S&DPattern and, thus, a S&DPattern may have more than one S&DImplementation.
- Finally, ExecutableComponents are real implementations of the S&DImplementations. These elements are not used at development time, but they are the realization of the selected S&D solution at run-time. An ExecutableComponent works as a stand-alone executable S&D solution ready to provide its services to applications. They are software, and sometimes hardware, components.

Every S&D solution provides at least one security property. Every S&DPattern (and every Integration Scheme) refer to an S&D solution. On the contrary, every S&D solution can be represented by one or more S&DPatterns and/or Integration Scheme. Each S&DPattern is implemented by means of at least one S&DImplementation. Finally, there is an ExecutableComponent entity for each S&DImplementation. While, S&DClasses are the most abstract level entities to represent S&D solutions, ExecutableComponents, being software components, are the lowest abstraction level way to represent an S&D solution. For the representation of S&D solutions, following the Serenity approach, developers need to count on, at least, one artefact for every level of the hierarchy.

To sum up, S&DClasses, S&DPatterns and S&DImplementations are development-time oriented artefacts, while ExecutableComponents are especially suitable for run-time. Serenity-aware applications include references to development-time artefacts. Depending on the artefact level of abstraction, at run-time, the SRF has more/less flexibility to select S&D S&D S&D S&D solutions. In other words, this approach enables the creation of open architectures where, at run-time, the SRF completes by applying the ExecutableComponents that implements the S&D solutions fixed at development-time. The main purpose of introducing this approach is to facilitate the dynamic substitution of S&D solutions at run-time while facilitating the development process.

4 Using TPM Functionalities to Prevent Unauthorized Access to Data

Returning to our previous example of the electronic prescription. Assuming that the PDA is protected by TPM, our solution uses TPM functionality to prevent unauthorised access to the patient's details (in this case their prescription). This is carried out in three stages. Firstly a public key, with given properties, is requested from the social worker (or rather their TPM) by the medical centre. Among these properties is that the key shall be non-migratable, bound to the TPM of the social worker's PDA, and should have certain PCR values (ensuring it hasn't been tampered with). A TPM generated certificate proves the key's properties. Then the medical centre encrypts the patient's prescription using that key and the resulting ciphertext is sent to the social worker's PDA, where finally the prescription is decrypted and sent to the pharmacy using the key's authorisation data. In the following paragraphs we explain in more detail how the key's properties ensure none other than the social worker can carry out the decryption.

4.1 Phases 1 and 2 – Setup and Encryption

First, the medical centre requires a key from the social worker’s PDA that is non-migratable and bound to specific PCR values. The following message sequence chart (msc) shows the subsequent communication between their PDA and the PDA’s TPM for generating this key. The actions are as follows:

1. The PDA starts an object specific authorisation session OSAP.
2. With TPM CreateWrapKey the social worker’s PDA requires the TPM to generate a command contains the key’s usage authorisation data (we do not discuss here where the key’s authorisation data comes from, it can for example be presented them or by their PDA).
3. The TPM generates keyA and returns the key blob.
4. The 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.

The social worker’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 the patient’s prescription and sends the ciphertext to the social worker’s PDA. This ensures confidentiality of the patient’s data during communication between the centre and the PDA.

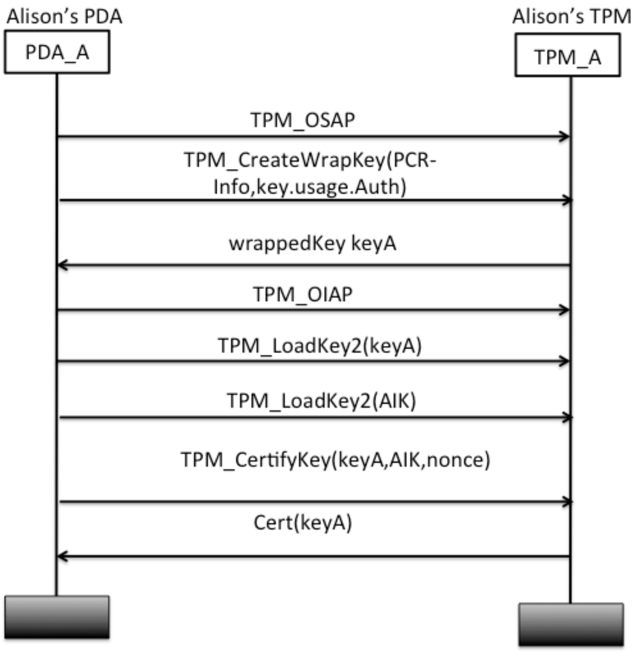

Fig. 1. Phases 1 and 2 setup and encryption

4.2 Phase 3 – Data Retrieval

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

1. Social worker's PDA starts an OIAP session.
2. The PDA then loads keyA into the TPM.
3. With TPM UnBind the 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 that 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 the PDA to the pharmacy.

The key's properties prevent unauthorised access to the patient's data during storage on the 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 being manipulated. Finally, assuming that the social worker does not reveal the key authorisation data to anybody, the key will only be used by the TPM after authorisation is given by the social worker.

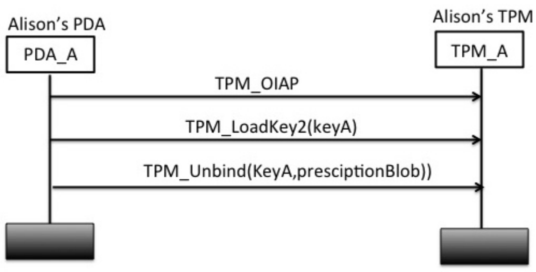

Fig. 2. Phase 3 decryption

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 unauthorised 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 the 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 requirements listed above. These patterns especially support applications that 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 until 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 the 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 realisations 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 realise 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 modelling artefacts [4].

5.2 SERENITY Operation

One interesting aspect to remark on 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 fulfilling 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 realise 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 use to provide them.

6 Conclusions and Future Work

In this paper we have aimed to show how complex technologies such as TPM can be used at both runtime and development and how security patterns, especially SERENITY, can facilitate the use of these technologies and devices. Currently we are focusing on three main lines of research: the development of the SERENITY Runtime Framework, the universality and flexibility of modelling artefacts' structure and contents, and finally, for producing the information contained in the pattern the necessary mechanisms and tools.

Acknowledgment. The work in this paper was partly co-sponsored by the EC Framework Programme as part of the ICT PASSIVE project (http://ict-passive.eu/) and the "Advanced Security Service cERTificate for SOA " ASSERT4SOA project (http://www.assert4soa.eu/).

References

1. Fernandez, E.: Security patterns. In: Procs. of the Eigth International Symposium on System and Information Security, SSI 2006, Keynote talk, Sao Jose dos Campos, Brazil (November 2006)
2. Fernandez, E., Rouyi, P.: A pattern language for security models. In: Pattern Languages of Program Design, PLoP 2001 (2001)
3. T. C. Group. TCG TPM Specification 1.2 (2006), `http://www.trustedcomputing.org`
4. Gallego, B., Serrano, D., Muñoz, A., Maña, A.: Security Patterns, towards a further level. In: The International Conference of Security and Cryptography, SECRYPT 2009, pp. 349–356 (2009)
5. Romanosky, S.: Security design patterns part 1, v1.4 (2001)

6. Armenteros, A., Muñoz, A., Maña, A., Serrano, D.: Security and Dependability in Ambient Intelligence scenarios: The communication prototype. In: International Conference on Enterprise Information Systems (2009)
7. Schumacher, M., Fernandez, E., Hybertson, D., Buschmann, F., Sommerlad, P.: Security Patterns - Integrating Security and Systems Engineering. John Wiley Sons, Chichester (2005)
8. Schumacher, M., Roedig, U.: Security engineering with patterns. Springer, Heidelberg (2001)
9. SERENITY. System engineering for security and dependability. IST project, funded by the EC (2006), `http://www.serenityproject.org/`
10. Wassermann, R., Cheng, B.: Security patterns. Technical Report MSU-CSE-03-23, Department of Computer Science, Michigan State University (August 2003)