An Architectural Framework for Analyzing Tradeoffs between Software Security and Performance

Vittorio Cortellessa[1], Catia Trubiani[1], Leonardo Mostarda[2], Naranker Dulay[2]

[1] Università degli Studi dell'Aquila, L'Aquila, Italy
{vittorio.cortellessa, catia.trubiani}@univaq.it
[2] Imperial College London, London, United Kingdom
{lmostard, nd}@doc.ic.ac.uk

Abstract. The increasing complexity of software systems entails large effort to jointly analyze their non-functional attributes in order to identify potential tradeoffs among them (e.g. increased availability can lead to performance degradation). In this paper we propose a framework for the architectural analysis of software performance degradation induced by security solutions. We introduce a library of UML models representing security mechanisms that can be composed with performance annotated UML application models for architecting security and performance critical systems. Composability of models allows to introduce different security solutions on the same software architecture, thus supporting software architects to find appropriate security solutions while meeting performance requirements. We report experimental results that validate our approach by comparing a model-based evaluation of a software architecture for management of cultural assets with values observed on the real implementation of the system.

Key words: performance, security, UML, GSPN, tradeoff analysis

1 Introduction

The problem of modeling and analyzing software architectures for critical systems is usually addressed through the introduction of sophisticated modeling notations and powerful tools to solve such models and provide feedback to software engineers.

However, non-functional attributes are often analyzed in isolation. For example, performance models do not usually take into account the safety of a system, as well as availability models do not consider security aspects, and so on. An early but relevant exception in this domain has been the definition of performability [19] that combines performance and availability aspects into the same class of models. With the increasing variety and complexity of computing platforms, we believe that the task of jointly analyzing non-functional attributes to study possible dependencies is becoming a critical task for the successful development of software architectures.

This paper works towards this goal. It presents a framework to jointly model and analyze the security and performance attributes of software architectures. The critical aspect that we tackle is to quantify the performance degradation incurred to achieve the security requirements. The basic idea is that the solution of a performance model that embeds security aspects provides values of indices that can be compared to the ones obtained for the same model (i) without security solutions, (ii) with different security mechanisms and (iii) with different implementations of the same mechanism. Such comparisons help software architects to decide if it is feasible to introduce/modify/remove security strategies on the basis of (possibly new) requirements.

Security is, in general, a complex and cross-cutting concern, and several mechanisms can be used to impact on it. In this paper we focus on two common mechanisms that are: encryption and digital signature.

In order to easily introduce security and performance aspects into software models we have built a library of models that represent security mechanisms ready to be composed. Once an application model is built, in order to conduct a joint analysis of security and performance with our approach it is necessary for the software designer: (i) to specify the appropriate security annotations (e.g. the confidentiality of some data), and (ii) to annotate the model with performance related data (e.g. the system operational profile). Thereafter, such an annotated model can be automatically transformed into a performance model whose solution quantifies the tradeoff between security and performance in the architecture under design. The setting where our approach works is Unified Modeling Language (UML) [1] for software modeling and Generalized Stochastic Petri Nets (GSPN) [15] for performance analysis.

The starting point of this work can be found in [5], where we have introduced an earlier version of this approach and a preliminary security library of models expressed as performance models. As envisaged in [5], thereafter we have applied the approach to real world case studies, and the experimentation phase led us to modify the approach as well as the security library.

An important aspect lacking in [5] that we tackle in this paper is that the choice and the localization of the appropriate security mechanisms should be driven from the system architecture (e.g. features of the communication channels, physical environment, etc.). For example, a message exchanged between two components might require encryption depending on whether the communication channel between the components is a wireless one or not (see Section 5). Hence, the main progress, in comparison to [5], is that here we express security mechanisms as UML architectural models, and the model composition is moved on the designer's side. Thus, our approach allows designers to explicity explore architectural alternatives for balancing security/performance conflicting concerns in the architectural phase of the development process.

To validate our approach we have conducted extensive experiments where we compare the results of our models with the data monitored on a real system. The promising numerical results that we have obtained significantly support the prediction capabilities of our approach.

The paper is organized as follows: in Section 2 we present the related work and describe the novelty of the approach with respect to the existing literature; in Section 3 we introduce our approach and the types of analysis that it can support; in Section 4 we discuss the library of security models and how it is used at the application level; in Section 5 we present the experimental results that we have obtained on a real case study; finally in Section 6 we give concluding remarks and future directions.

2 Related work

The literature offers a wide variety of proposals and studies on the performance aspects of security, but most of them analyze the performance of existing standards such as IPsec and SSL. Therefore the analysis conducted in these approaches remains confined to very specific problems and solutions. However, noteworthy examples in this direction can be found in [4] [9] [14].

A tradeoff analysis of security and scalability can be found in [16], which stresses the importance of understanding the security required while minimizing performance penalties. Our concern is similar to this one because we also target an analysis of how security solutions impact on system performance. However, in [16] the analysis is conducted using a specific security protocol (i.e. SSL) and a limited set of cryptographic algorithms, whereas our framework is intended to model and analyze more general solutions.

Estimating the performance of a system with different security properties is a difficult task, as demonstrated in [12], where they emphasize how difficult is to choose between secure and non-secure Web services, RMI and RMI with SSL. The authors solve the problem by performing different measurements on different platforms to elicit guidelines for security setting selections.

There are also several works that use aspect-oriented modeling (AOM) to specify and integrate security risks and mechanisms into a system model, such as the one in [8]. An interesting performance study, based on AOM, can be found in [22] where security solutions are modeled as aspects in UML that are composed with the application logics, and the obtained model is transformed into a performance model. This work uses an approach to the problem that is similar to ours, in that they are both based on model annotations and transformations. Our work, at some extent, refines the approach in [22] because we target the problem of representing elementary security mechanisms aimed at guaranteeing certain security properties, whereas the analysis in [22] is performed only on the SSL protocol and a set of properties embedded in it. Furthermore this paper differs from [22] because for each security mechanism we additionally consider the reasonable implementation options (e.g. the key length of an encryption algorithm) that significantly impact on the software performance.

The lack of a model-based architectural solution to this problem is the major motivation behind our work. This paper aims at overcoming the limitations of ad hoc solutions that estimate the performance of specific security technologies. To achieve this goal, we propose a framework that manages and composes platformindependent models of security mechanisms, thus allowing designers to estimate and compare the performance of different architectural security solutions for critical systems.

3 Our approach

In this section we present a framework that allows us to quantify the tradeoff between the security solutions introduced to cope with the application requirements and the consequent performance degradation.

In Figure 1 the process that we propose is reported. The process has been partitioned in two sides: on the left side all models that can be represented with a software modeling notation (e.g. UML) appear; on the right side all models represented with a performance modeling notation (e.g. GSPN) appear.

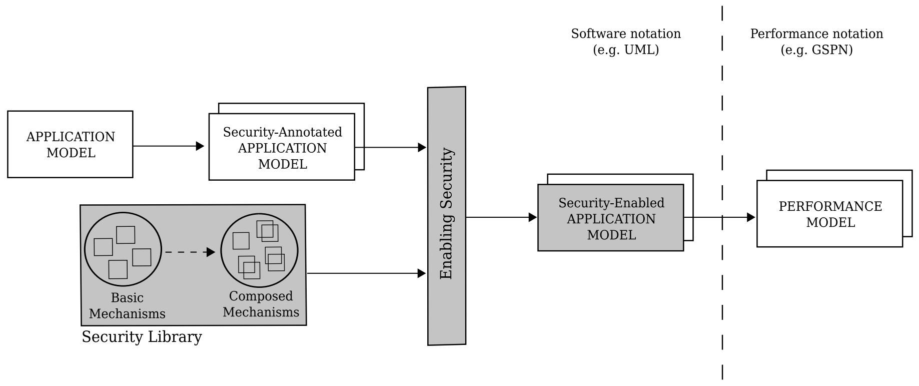


Fig. 1. A joint process for security and performance goals.

The starting point of the process is an *Application Model* that is a static and dynamic representation of a software architecture. For sake of simplification we assume that such model is annotated with the performance parameters related to the application (e.g. the system operational profile) ([1]).

A *Security-Annotated* model is obtained by introducing security annotations in the former. Such annotations specify *where* security properties have to be inserted, namely which software services have to be protected and how (e.g. the entity providing a certain service must be authenticated before using it).

The contribution of this paper can be located among the shaded boxes of Figure 1. A *Security Library* of UML models is provided; in particular, models of *Basic Mechanisms* are combined to build *Composed Mechanisms*.

[1] Note that the standard MARTE profile [2] has been adopted to specify performance parameters in our UML models. However, it is out of the scope of this paper to provide details of performance annotations, because well assessed techniques exist for this goal [20].

The task of *Enabling Security* consists in embedding the appropriate security mechanisms in the software architecture. This step is driven by the security annotations specified in the application model, and a *Security-Enabled Application Model* is finally obtained. As an example, if a security annotation specifies that data integrity must be guaranteed for a certain service, an additional pattern with the steps needed for the data integrity mechanism must be introduced in the architectural model wherever the service is invoked. Such a pattern is one of the mechanisms modeled in our security library.

A key aspect of this task is the composability of models, and this is achieved in our approach through two features: (i) entry points for security mechanisms are unambiguously defined by security annotations, and (ii) mechanism models in the security library have been designed to be easily composable with application models (see Section 4.2).

The security-enabled application model is finally transformed into a GSPN-based *Performance Model*. This step involves not only a transformation between modeling notations ([2]), but an additional task is necessary to appropriately instrument the target performance model, because security mechanisms inevitably introduce additional performance parameters to be set in the model.

The definition of such parameters is embedded in the security library where they are defined in an application-independent way. For example, the encryption mechanism introduces additional parameters affecting system performance, such as the complexity and resource requirements of the encryption algorithm, its mode of operation (e.g. CBC), the lengths of the keys, etc. However, the task of enabling security implies the usage of such mechanisms at the application level, thus they can be influenced by further application-dependent characteristics. For example, the encryption mechanism efficiency is influenced by the speed of the CPU executing the encryption algorithm, the length of the message to be encrypted, etc.

Hence, the GSPN performance model finally generated has to be carefully parameterized with proper performance data.

The GSPN performance models are solved by SHARPE [11] [21], and the model evaluation provides performance indices that jointly take into account both the security and the performance features required for a critical system.

Note that such tradeoff analysis can be conducted on multiple security settings by only modifying the security annotations and re-running the following steps of our approach. In fact, in Figure 1 we can define a certain multiplicity in the security annotations to emphasize that different strategies can be adopted for the same architecture according to different system settings (see Section 5.1).

Finally we observe that several types of analysis can be conducted on the models built with this approach: (i) a performance model with a set of security requirements can be compared with one without security to simply study the performance degradation introduced from certain security settings; (ii) the per-

[2] Well consolidated techniques have been exploited to transform software models (e.g. UML diagrams) into performance models (e.g. GSPN), and readers interested can refer to [3] for an extensive survey on this topic.

formance estimates from different performance models can be compared to each other to study the tradeoff between security and performance across different architectural configurations.

Note that the latter analysis can be hierarchically conducted by assuming configurations that are ever more secure. This scenario leads to continuously raising the security settings, thus allowing us to quantify the amount of system performance degradation at each increase of security.

4 Enabling Security

The OSI (Open Systems Interconnection) Security Architecture standard [18] aims at defining, through basic mechanisms and their composition, various properties of a system belonging to a secure environment.

Based on OSI, in [5] we have introduced a set of performance models for Basic and Composed Security Mechanisms. An open issue of that prelimary work was the usage of those models on real applications. After experimenting on real case studies, we realized that the directives in [18] led us to produce, in some cases, models that were too abstract to be useful in practice. This consideration has brought to substantially modify our security models.

In Table 1 a refined set of Basic and Composed Mechanisms, and their dependencies, is illustrated. Each row refers to a Composed Mechanism and each column to a Basic one. An X symbol in a (i, j) cell means that Basic Mechanism j is used to build Composed Mechanism i.

Basic / Composed	Encryption	Digital Signature Generation	Digital Signature Verification	Decryption
Data Confidentiality	X			X
Data Integrity		X	X	
Peer Entity Authentication		X	X	
Data Origin Authentication		X		

Table 1. Dependencies between Basic and Composed Mechanisms.

The Basic Mechanisms that we consider are: *Encryption*, which refers to the usage of mathematical algorithms to transform data into a form that is unreadable without knowledge of a secret (e.g. a key); *Decryption*, which is the inverse operation of Encryption and makes the encrypted information readable again; the *Digital Signature* is a well-known security mechanism that has been split into *Generation* and *Verification*, in order to express finer grained dependencies among mechanisms.

The Composed Mechanisms have been defined as follows. *Data Confidentiality* refers to the protection of data such that only the authorised entity can read it. *Data Integrity* assures that data has not been altered without authorisation.

Peer Entity Authentication is an identity proof between communicating entities. *Data Origin Authentication* supports the ability to identify and validate the origin of a message; it has been defined as a Composed Mechanism although it depends on only one Basic Mechanism (see Table 1). This choice allows to interpret the generation of a digital signature as an high level mechanism that can be used by itself to enable the (possibly future) verification of data origin.

Models of Basic and Composed Mechanisms have been expressed as UML Sequence Diagrams (see Section 4.1).

In [13] an UML profile, called UMLsec, is presented for secure system development. Security properties (i.e. secrecy, integrity, authenticity and freshness) are specified as tagged values of a common stereotype (i.e. data security). We do not use the UMLsec profile because the set of models we consider act at a lower level of abstraction to provide a higher degree of freedom to architectural designers, e.g. about the encryption algorithm and the key lengths. Ultimately we intend to provide instruments for architecting security and performance critical systems on the basis of quantitative estimates.

4.1 Security Library

In this section we concentrate on the security mechanisms identified in Table 1. Some preliminary operations, such as the generation of public and secret keys and the process of obtaining a certificate from a certification authority, are executed once by all software entities involved in the security annotations.

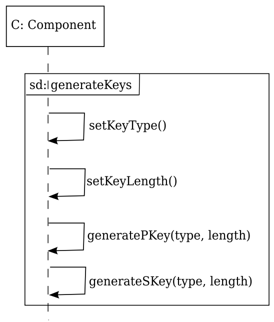
In Figure 2(a) the generation of public and private keys is illustrated: a component sets the key type (*setKeyType*) and the key length (*setKeyLenght*) and generates the public (*generatePKey*) and the private (*generateSKey*) keys.

In Figure 2(b) the process of obtaining a certificate from a certification authority is shown: a component requiring a certificate (*reqCertificate*) sends its information and the public key; the certification authority checks the credentials (*checkEntityInfo*) and, if trusted, generates the certificate (*generateCertificate*) and sends it back (*sendCertificate*) to the software entity.

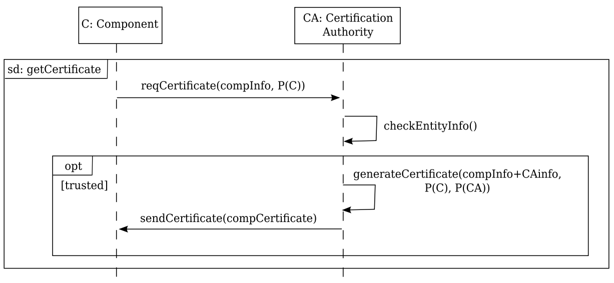
Figure 3 shows the UML Sequence Diagram modeling Encryption. Firstly the sender of the message decides the type of algorithm to use (*setAlgorithmType*) and the key length (*setKeyLength*). The encryption can be of two different types: *asymEncrypt* means asymmetric encryption (i.e. by public key), whereas *symEncrypt* indicates symmetric encryption (i.e. by a shared secret key).

For asymmetric encryption the sender sets the padding scheme it requires (*setPaddingScheme*) and verifies the receiver's certificate if it is not already known. Finally, the encryption algorithm (*encryptAlgorithm*) is executed on the message (*msg*) with the public key of the receiver (*P(R)*).

For symmetric encryption the sender sets the algorithm mode (*setAlgorithmMode*), performs a *key-exchange* protocol if a shared key is not already exchanged, and requires the exchange of certificates. We have modelled the ISO/IEC 11770-3 key exchange protocol that achieves mutual authentication and key exchange. This requires both parties to generate a key that can be combined to form a single session key. Three messages are exchanged. The first one is sent



(a) *generateKeys*


C: Component
CA: Certification Authority
sd: getCertificate
reqCertificate(compInfo, P(C))
checkEntityInfo()
opt
[trusted]
generateCertificate(compInfo+CAinfo, P(C), P(CA))
sendCertificate(compCertificate)

(b) *getCertificate*

Fig. 2. UML Sequence Diagram of some preliminary operations.

by the sender S and contains the sender information, the key generated by it, and a nonce; the message containing all this information is encrypted with the public key of the receiver. The second one is sent by the receiver R and contains the receiver information, the key generated by it, the nonce previously sent by S and a new nonce generated by R; the message containing all this information is encrypted with the public key of the sender. The third one is sent by the sender and contains the nonce sent by the receiver. Finally, the encryption algorithm (i.e. *encryptAlgorithm*) is executed on the message (*msg*) with a session key obtained combining the keys generated by the sender and the receiver ($K(K_S, K_R)$).

Figure 4(a) shows the the UML Sequence Diagram modeling the Digital Signature Generation. First, the hash function (*setHashFunction*) algorithm must be specified, then the digest generated (*generateDigest*) and finally the encryption algorithm (*encryptAlgorithm*), by using the entity private key, applied on the digest.

Figure 4(b) shows the UML Sequence Diagram modeling the Digital Signature Verification. A message (*msg*) and the digital signature (*digitSign*) are received as inputs. Two operations are performed: the first one is to calculate the digest (*execHashFunc*); the second one is the actual execution of the encryption algorithm applied on the input digital signature producing a forecast of the real signature (*encryptAlgorithm*). The last computation involves the verification of the digital signature (*verifyDigitSign*) which compares the forecast digital signature with the received one, in order to confirm the verification.

For sake of space the UML Sequence Diagram modeling the Decryption is not shown but it can be summarized as follows: after receiving the encrypted message, the algorithm type and the key length are extracted, and the decryption algorithm is executed to obtain the plain text.

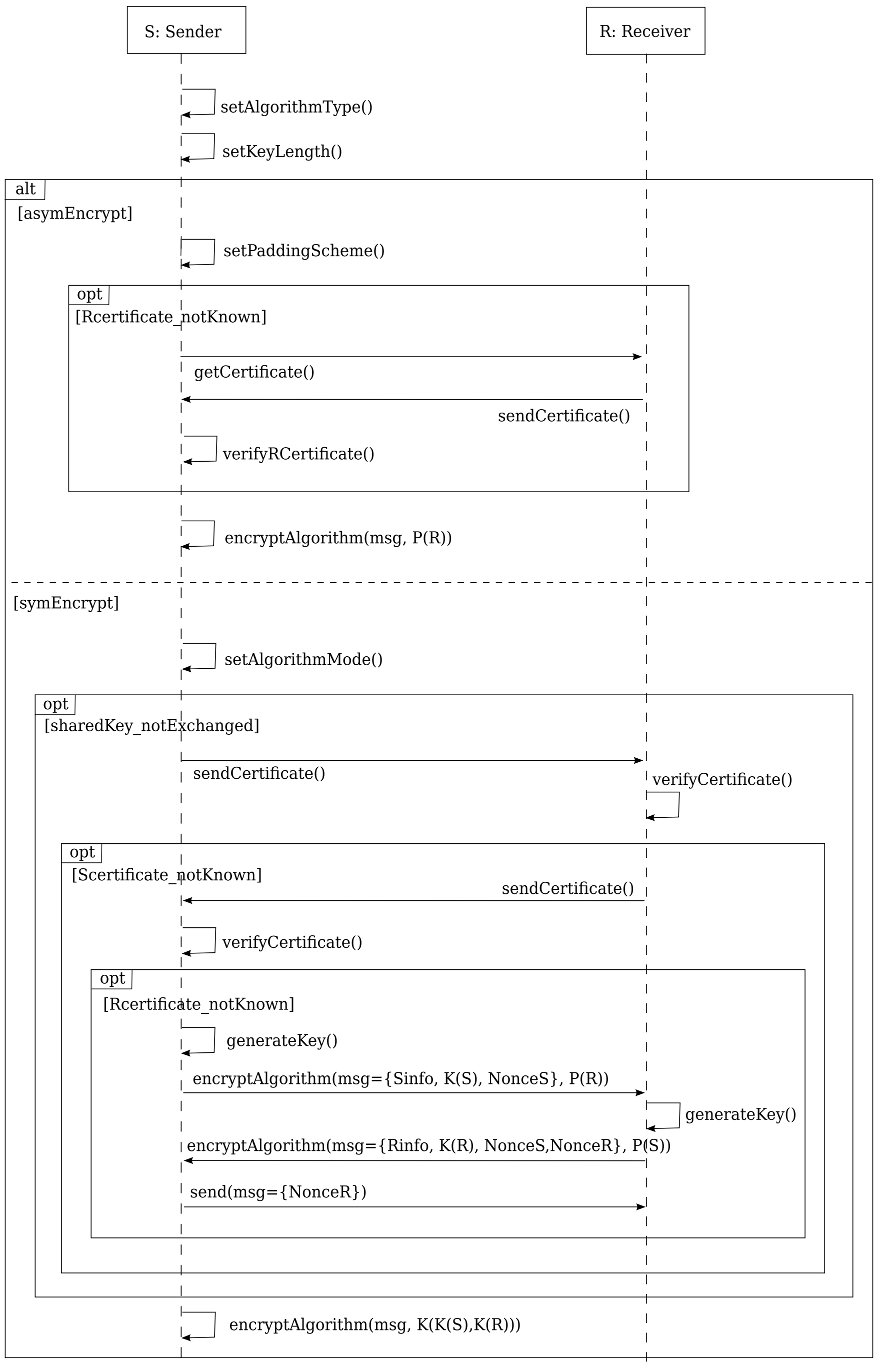
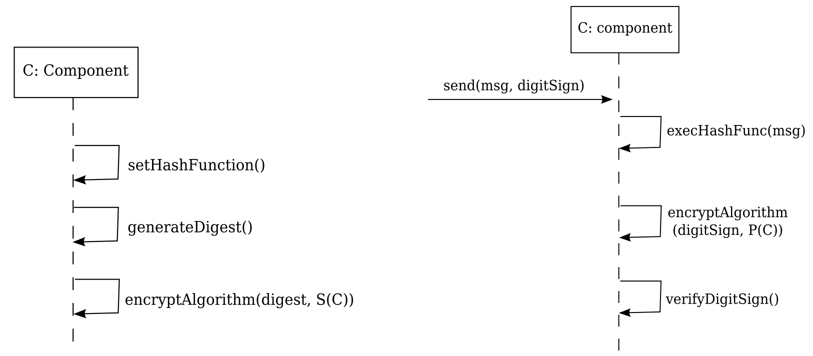


Fig. 3. UML Sequence Diagram of the *Encryption* mechanism.



(a) Digital Signature *Generation* (b) Digital Signature *Verification*

Fig. 4. UML Sequence Diagram of the *Digital Signature* mechanism.

4.2 Security-Enabled Application Model

In this section we briefly discuss how the Composed Mechanisms of Table 1 are annotated and embedded in the application model to obtain a *Security-Enabled Application Model*.

The Data Confidentiality mechanism can be annotated on a software connector, and it means that data exchanged between the connected components are critical and need to be kept secret.

In Figure 5 we illustrate how the Composed Mechanism is enabled in the application model. On the left side the security annotation is added in the static architectural model (i.e. UML Component Diagram) on the software connector, and it means that client and supplier components exchange critical data. On the right side all Basic Mechanisms used to build the composed one (see Table 1) are embedded in the dynamic architectural model (i.e. UML Sequence Diagram), hence data are encrypted by the client component before their exchange and later decrypted by the supplier component.

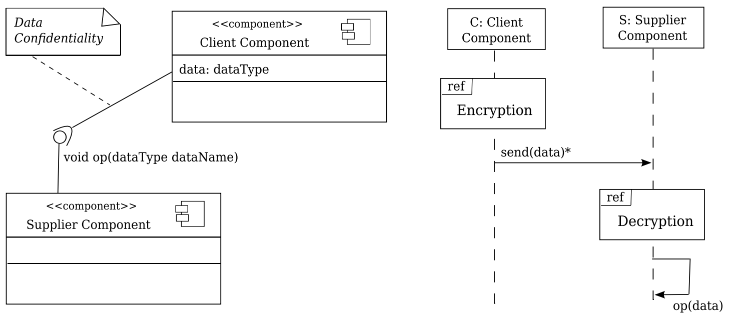


Fig. 5. Enabling *Data Confidentiality* mechanism.

The Peer Entity Authentication and Data Integrity mechanisms are both obtained by the generation of the digital signature followed by its verification, as reported in Table 1. In fact they have the same modeling structure in terms of the sequence of operations. The difference is in the content of the message used to generate the digital signature. The Peer Entity Authentication can be annotated for software components and the content of the message is represented by its credentials, whereas Data Integrity can be annotated for attributes and the message is represented by application specific data.

Finally, the Data Origin Authentication mechanism can be annotated on attributes and means that data are critical and need to be authenticated. It depends on the digital signature generation, as reported in Table 1.

5 Experimental validation

In this section we apply our approach to a real case study: a large distributed system in the domain of cultural asset management, built in the context of the CUSPIS project [7]. This experimental validation highlights the potential of our approach ([3]).

We denote by SC_i a *system configuration* that represents the required security settings to be included in the application model. It is obvious that the same application model may have multiple configurations, each leading certain security characteristics to the system.

For sake of experimentation we have numbered the *system configurations* in a hierarchical way SC_0, SC_1,..., SC_n, so that for configuration SC_i the required security settings properly include all the ones adopted for SC_j with $i > j$. SC_0 represents the system without any security setting. Such hierarchical organization of system configurations has been adopted in our case study in order to stress the progressive performance degradation introduced by the increasing of security.

However, our framework can be used to study the tradeoff between security and performance across different system configurations, not strictly ordered on raising security settings. For example, two generic configurations SC_i and SC_j may share certain settings but, at the same time, they may differ for other settings.

5.1 The CUSPIS system

The CUSPIS system aims to improve the protection of cultural assets (CA), such as sculptures and paintings, through the use of computer-based strategies (e.g. cryptography and satellite tracking). Our experimentation focuses on two services: *cultural asset authentication* and *cultural asset transportation* ([4]).

[3] For sake of space we only report the most relevant results among all the evaluations carried out.

[4] For sake of space we report only the *authentication* scenario, whereas we refer to [6] for the *transportation* one.

Cultural asset authentication aims to ensure that visitors to an exhibition, or potential buyers at an auction, can obtain cultural asset information and verify its authenticity (see [17] for details). Authentication is achieved by assigning a Geo Data (GD) tag containing information referring to each asset. A tag must be produced by a qualified organisation (e.g. the sculptures producer) to improve the asset protection.

Our experimentation in the cultural asset authentication service focuses on the *GD generation* scenario. It is performed in the following way: the qualified organisation generates the GD information (*genGDinfo*) and sends it (*send*) to a database that stores it (*storeGDinfo*).

The analysis of the *GD generation* scenario leads us to define two different system configurations (i.e. SC_1 and SC_2), as motivated in the following.

Figure 6(a) shows the *Security-Enabled Application Model* for the configuration SC_1: the qualified organisation provides *Data Origin Authentication* of the *gd* data, that is uploaded to a database. The uploading does not require any security solution since we assume that it is performed through a secure channel. The operation that returns that value (*genGDinfo()*) is defined as a critical operation and tagged with a star in the UML Sequence Diagram of Figure 6(a). A “ref” fragment that points to the Digital Signature Generation mechanism is added, as stated in Table 1.

The system configuration SC_1 is not security-wide when the qualified organisation device and the database communicate through an insecure network; the configuration SC_2 solves this problem by adding *Data Confidentiality* to the software connector requiring the *storeGDinfo()* operation. The exchange of data is performed through the *send()* operation that is defined as a critical operation, and it is tagged with a star in the UML Sequence Diagram of Figure 6(b). The references to the Encryption/Decryption mechanisms are added, as stated in Table 1.

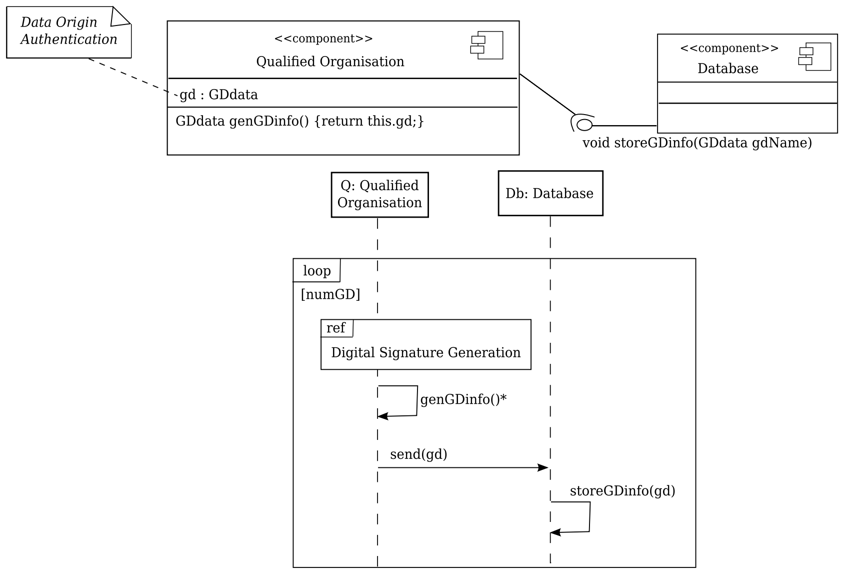
5.2 CUSPIS implementation details

Experiments for the CUSPIS system have been performed by running the same application code on two machines. The first machine has an Intel(R) Core2 T7250 CPU running at 2GHz with 2GB RAM, and runs the Windows Vista operating system. The second machine has an Intel Pentium4 3.4Ghz with 2GB RAM, and runs the Windows XP operating system.

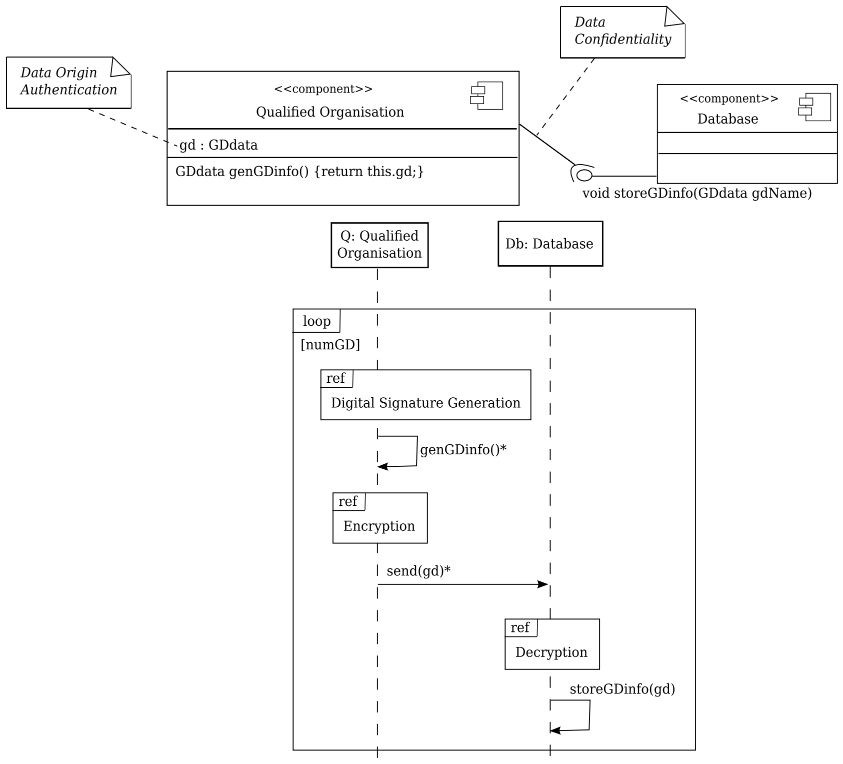
In configuration SC_1 the digital signature was performed by using *SHAwithRSA* with different key sizes of 1024, 2048 and 4096 bits. In configuration SC_2 the encryption/decryption was performed by using an AES algorithm with a 256-bit key size in CBC mode.

5.3 Applying our approach to CUSPIS

In this Section we describe the experimental results that we have obtained from applying our approach to the CUSPIS system. From a performance analysis


Data Origin Authentication
<<component>>
Qualified Organisation
gd : GDdata
GDdata genGDinfo() {return this.gd;}
<<component>>
Database
void storeGDinfo(GDdata gdName)
Q: Qualified Organisation
Db: Database
loop
[numGD]
ref
Digital Signature Generation
genGDinfo()*
send(gd)
storeGDinfo(gd)

(a) *GD generation* in SC_1



(b) *GD generation* in SC_2

Fig. 6. Security-Enabled Application Models for *GD generation.*

viewpoint, our experiments follow standard practices: construct the model, validate the model by comparing model results with real numerical values obtained from monitoring the implemented system while varying model parameters [10].

The validation of the *GD generation* scenario undergoes generic performance and security goals of a qualified organisation, which can be summarised as follows: (i) the number of tags generated per second must be as high as possible (a performance issue); (ii) tags must be hard to compromise (a security issue).

Based on the description in Section 5.1, two different GSPN performance models are built ([5]), one for each configuration considered: SC_1 and SC_2. Tables 2 and 3 report the results that we have obtained, respectively, for configurations SC_1 and SC_2.

	KeySize (byte)	Model Solution Results (tags/sec)	Implementation Monitoring Data (tags/sec)	Model Prediction Error (%)
Platform 1	1024	17.45	17.3	*0.86*
	2048	9.32	9.11	*2.25*
	4096	1.98	1.92	*3.03*
Platform 2	1024	17.13	16.61	*3.03*
	2048	8.45	8.11	*4.02*
	4096	1.85	1.78	*3.78*

Table 2. *GD generation* - analysis of configuration SC_1.

The columns of Table 2 can be divided into three sets. The first column reports the size of the key used in the encryption algorithm. The second set of columns reports the experimental results: the number of tags per second obtained from the model solution, the same values as monitored on the real implementation. Finally the last column reports the prediction error, expressed in percentage, of the model results in comparison to the monitored ones.

	KeySize (byte)	Model Solution Results (tags/sec)	Implementation Monitoring Data (tags/sec)	Model Prediction Error (%)
Platform 1	1024	3.43	3.29	*4.08*
	2048	2.93	2.85	*2.73*
	4096	1.35	1.33	*1.48*
Platform 2	1024	4.16	4.09	*1.68*
	2048	3.33	3.2	*3.90*
	4096	1.38	1.34	*2.90*

Table 3. *GD generation* - analysis of configuration SC_2.

The rows of Table 2 can be divided into two sets, one for each platform considered (see Section 5.2). Within each set, numbers are reported for three values of the key size. For example, the first row of the Table 2 indicates the

[5] GSPN performance models are shown here [6].

GD generation for the configuration SC_1 on Platform 1 with a 1024-bit key size: the model predicts that the system is able to generate 17.45 tags/second, the monitoring of the implementation reveals that the system actually generates 17.30 tags/second, and this leads to a gap between the model and the application of about 0.86%. Similarly promising results have been obtained for other key sizes and on both platforms, as shown in the last column of the Table, where the error never exceeds 4.02%.

Table 3 is similar to Table 2 for the organization of columns and rows; it collects the results for the configuration SC_2. For example, the first row of Table 3 indicates the *GD generation* for the configuration SC_2 on Platform 1 with a 1024-bit key size: the model predicts that the system is able to generate 3.43 tags/second, the monitoring of the implementation reveals that the system actually generates 3.29 tags/second, and this leads to a gap between the model and the application of about 4.08%. In the Table this latter value is the worst one, in fact better predictive results have been obtained for other key sizes ons both platforms, as shown in the last column of the Table 3.

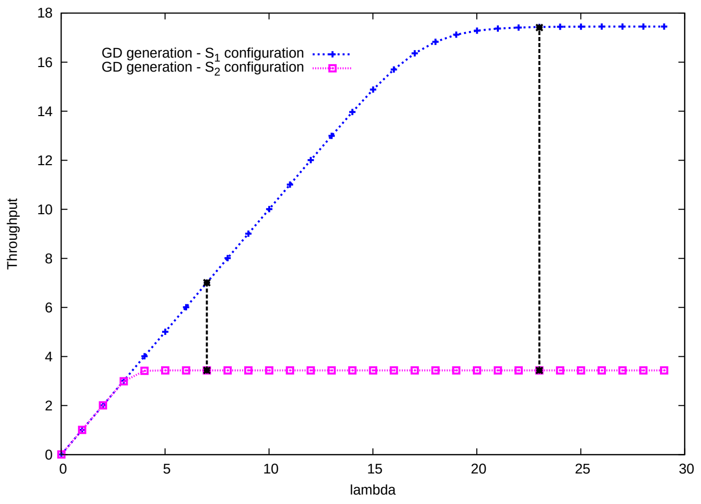


Fig. 7. *GD generation* throughput.

In both tables the number of tags per second for the model solution was obtained by measuring the throughput value of the ending timed transition in both GSPN models. Besides, the corresponding metrics has been monitored on the actual implementation of the system. All these measures have been obtained with the system under workload stress, which occurs when the arrival rate is high enough to make the system always busy.

The analysis of workload for both SC_1 and SC_2 configurations is shown in Figure 7. The curves are obtained by solving the GSPN models under the configuration of Platform 1 with a key size of 1024 byte (i.e. the first row of Tables 2 and 3). In particular, on the x-axis the rate Λ of arrivals to the system is reported; on the y-axis the throughput of the ending timed transition is shown. Note that in SC_1 the maximum throughput of 17.45 (see Table 2) is achieved for $\Lambda = 23$ requests/second, whereas in SC_2 the maximum throughput of 3.43 (see Table 3) is achieved for $\Lambda = 7$ requests/second.

From the comparison of Tables 2 and 3 some interesting issues emerge with regard to the performance degradation induced in SC_2 by raising security settings (i.e. by introducing the additional Data Confidentiality mechanism). In Table 4 we have reported the percentage of performance degradation obtained (for model results and for monitored data) when moving from SC_1 to SC_2 configuration in both platforms while varying the key size. For example, the value 80.34% in the upper leftmost cell of the Table is obtained as $100 - (3.43 * 100)/17.45$ (see Tables 2 and 3).

KeySize (byte)	Model Results Platform 1 (%)	Monitored Data Platform 1 (%)	Model Results Platform 2 (%)	Monitored Data Platform 2 (%)
1024	80.34	80.98	75.71	75.38
2048	68.56	68.71	60.59	60.54
4096	31.82	30.73	25.40	24.72

Table 4. *GD generation*: from SC_1 to SC_2.

We note that our model consistently provides almost the same amount of performance degradation as the one observed in practice. This further supports the validity of our approach. An interesting consideration is that for smaller values of the key size the performance degradation is more dramatic. This is due to the fact that this key size affects the execution time of the Data Origin Authentication mechanism that is part of both configurations. Hence, while growing this size, the latter mechanism dominates in terms of execution time with respect to the Data Confidentiality, executed only in SC_2, whose execution time does not vary with this key size.

6 Conclusions

In this paper we have introduced a framework to support the analysis of software architecture performance degradation due to the introduction of security mechanisms. Such a framework is that it is able to numerically quantify the system performance degradation while varying the adopted security solutions. This type of analysis can in fact support many decisions of software architects that span from simply evaluating if such performance degradation can be reasonably accepted from users, to choosing among different security solutions the one that provides the best tradeoff between security and performance properties.

A peculiar characteristic of our approach is the introduction of models for Basic Security Mechanisms. With this modular approach it is possible to study the performance degradation introduced by any meaningful combination of these (and possible newly built) security mechanisms. By pushing this concept ahead, a more complex type of analysis can be performed on models built by multiple Composed Mechanisms to represent the specification of an existing protocol, such as SSL. In this case an interesting study would be to observe how our models estimate the performance indices, and compare these results to what claimed in the corresponding protocol specifications.

As shown in the experiments, our architectural models very promisingly and quite accurately predict the performance of critical systems equipped with different security settings and implementation options. The results that we have obtained on our case study are somehow quite surprising in terms of percentage of degradation that can be introduced even from common security settings. Furthermore, we have been able to quantify the difference of degradation across platforms that, in some cases, achieves non-negligible thresholds.

The security mechanisms we consider (i.e. encryption and digital signature) can be seen as test beds for more complex security concerns. Modern applications may have to face with larger security vulnerabilities, and strategies for mitigating most of such vulnerabilities are cross-cutting and difficult to encapsulate (e.g., prevention of cross-site scripting errors). In this direction, our framework has been conceived to enable the modeling and analysis of security patterns that do not break the defined architectural abstraction level. In other words, the complexity of a pattern that implements a certain security strategy is not a problem on our framework as long as it can be (even piecewise) plugged into the application model. Moreover, such patterns can also spread from static to dynamic features of the system architectural model (e.g. see Figure 5).

In the near future we plan to automate the tradeoff analysis of the security configurations by automatically exploring the trade space. Such automation is feasible because performance models embedding security properties are generated once and the exploration of the trade space can be automatically performed by instrumenting the model with different numerical values for the input parameters. Besides, we devise to apply our approach to other real world examples in order to assess the scalability of the framework.

We consider this work as a starting point for studying even more sophisticated tradeoffs between security and performance. We plan to introduce into our evaluations the costs of security solutions as an additional attribute that very often affects software architects' decisions.

In the long term, it is of great interest to study the tradeoff between security and other non-functional attributes, such as availability. For example, addressing the problem of quantifying and locating data replicas for availability purposes without heavily affecting the security of the system would be crucial in certain domains.

Acknowledgments

This work has been partly supported by the italian project PACO (Performability-Aware Computing: Logics, Models, and Languages) funded by MIUR and by the UK EPSRC research grant EP/D076633/1 UbiVal (Fundamental Approaches to the Validation of Ubiqituous Systems).

References

1. UML 2.0 Superstructure Specification, OMG document formal/05-07-04, Object Management Group (2005), *http://www.omg.org/cgi-bin/doc?formal/05-07-04*.
2. UML Profile for MARTE beta 2, OMG document ptc/08-06-09 (2008), *http://www.omgmarte.org/Documents/Specifications/08-06-09.pdf*.
3. S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: A survey. *IEEE TSE 30(5)*, pages 295–310.
4. M. Blaze, J. Ioannidis, and A. D. Keromytis. Trust management for ipsec. *ACM Transactions on Information and System Security*, 5(2):95–118, 2002.
5. V. Cortellessa and C. Trubiani. Towards a library of composable models to estimate the performance of security solutions. In *WOSP*, pages 145–156, 2008.
6. V. Cortellessa, C. Trubiani, L. Mostarda, and N. Dulay. An Architectural Framework for Analyzing Tradeoffs between Software Security and Performance - Extended results. Technical Report 001-2010, Dipartimento di Informatica - Università dell'Aquila, http://www.di.univaq.it/cortelle/docs/001-2010-report.pdf, 2010.
7. European Commision 6th Framework Program. Cultural Heritage Space Identification System (CUSPIS). www.cuspis-project.info.
8. R. B. France, I. Ray, G. Georg, and S. Ghosh. Aspect-oriented approach to early design modelling. *IEE Proceedings - Software*, 151(4):173–186, 2004.
9. V. Gupta, S. Gupta, S. C. Shantz, and D. Stebila. Performance analysis of elliptic curve cryptography for SSL. pages 87–94, 2002.
10. A. Harbiterr and D. A. Menasce. A methodology for analyzing the performance of authentication protocols. *ACM TISSEC*, 2002.
11. C. Hirel, R. Sahner, X. Zang, and K. Trivedi. Reliability and performability modeling using sharpe 2000. In *LNCS 1786*, pages 345–349, 2000.
12. M. B. Juric, I. Rozman, B. Brumen, M. Colnaric, and M. Hericko. Comparison of performance of web services, ws-security, rmi, and rmi-ssl. *Journal of Systems and Software*, 79(5):689–700, 2006.
13. J. Jurjens. *Secure Systems Development with UML.* 2004.
14. K. Kant, R. K. Iyer, and P. Mohapatra. Architectural impact of Secure Socket Layer on internet servers. pages 7–14, 2000.
15. M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* Fourth edition, November 1994.
16. D. A. Menascé. Security performance. *IEEE Internet Computing*, 7(3):84–87, 2003.
17. L. Mostarda, C. Dong, and N. Dulay. Place and Time Authentication of Cultural Assets. In *2nd Joint ITRUST and PST Conferences on Privacy, Trust and Security (IFIPTM 2008)*, 2008.
18. W. Stallings. *Cryptography and network security: Principles and Practice.* Prentice Hall, fourth edition, 2006.
19. A. T. Tai, J. F. Meyer, and A. Avizienis. *Software Performability: From Concepts to Applications.* Kluwer Academic Publishers, Boston, 1996.

20. R. Tawhid and D. C. Petriu. Towards automatic derivation of a product performance model from a UML software product line model. In *WOSP*, pages 91–102, 2008.
21. K. Trivedi. Sharpe interface, user's manual, version 1.01. Technical report, http://www.ee.duke.edu/ chirel/MANUAL/gui.doc, 1999.
22. C. M. Woodside, D. C. Petriu, D. B. Petriu, J. Xu, T. A. Israr, G. Georg, R. B. France, J. M. Bieman, S. H. Houmb, and J. Jürjens. Performance analysis of security aspects by weaving scenarios extracted from UML models. *Journal of Systems and Software*, 82(1):56–74, 2009.