Security-by-Contract-with-Trust for Mobile Devices *†

Gabriele Costa[1,2], Aliaksandr Lazouski[1,2]
Fabio Martinelli[2], Ilaria Matteucci[2]
[1]*Dipartimento di Informatica, Università di Pisa*
[2]*IIT-CNR, Italy*
name.surname@iit.cnr.it

Valérie Issarny, Rachid Saadi
ARLES Project-Team
INRIA, CRI Paris-Rocquencourt
France
name.surname@inria.fr

Nicola Dragoni
Department of Informatics
and Mathematical Modelling
Technical University of Denmark
ndra@imm.dtu.dk

Fabio Massacci
Dipartimento di Ingegneria
e Scienza dell'Informazione
Università di Trento
massacci@disi.unitn.it

Abstract

Security-by-Contract (S×C) is a paradigm providing security assurances for mobile applications. In this work, we present the an extension of S×C, called Security-by-Contract-with-Trust (S×C×T). Indeed, we enrich the S×C architecture by integrating a trust model and adding new modules and configurations for managing contracts. Indeed, at deploy-time, our system decides the run-time configuration depending on the credentials of the contract provider. The run-time environment can both enforce a security policy and monitor the declared contract. According to the actual behaviour of the running program our architecture updates the trust level associated with the contract provider.

We also present a possible application of our framework in the scenario of a mobile application marketplace, *e.g.*, Apple AppStore, Cydia, Android Market, that, nowadays, are considered as one of the most attractive e-commerce activity for both mobile application developers and industries of mobile devices. Since the number of applications increases, Mobile Applications Marketplace (MAMp) sets up recommendation systems that rank and highlight mobile applications by category, social activity, etc.

The S×C×T framework we propose is applied in this scenario for providing security on customers' mobile devices as well as help Mobile Applications Marketplaces to enhance their recommendation systems with security feedback.

The main advantage of this method is an automatic management of the level of trust of software and contract releasers and a unified way for dealing with both security and trust.

Keywords: Security-by-Contract, Contract Monitoring, Trust Management, Mobile Application Criticality, Managing Feedback.

1 Introduction

In the last decades, the number of devices, *e.g.*, mobile phones, smart phones and slates, used in our daily life is rapidly growing up. Furthermore, the computational capabilities of such devices tend to increase over and over. This allows to download and run a rich variety of applications on mobile devices.

Mobile Java applications (MIDlets) offer a clear example of fixed trust relationship. Indeed, a MIDlet is a software released by some vendor that clients download and install on their device. The serious constraints on the resources of mobile devices (*e.g.*, CPU, memory, battery) make several security mechanisms practically infeasible. The current technique for providing security assurances to mobile device users is based on software certification released by a accredited *certification authority* (CA).

*This paper is an extended version of the paper [7] presented at IMIS 2010.

†Work partially supported by EU-funded project FP7-257930 ANIKETOS, by EU-funded project FP7-231167 CONNECT and by EU-funded project FP7-256980 NESSoS.

The certificate-based approach has several, well known drawbacks. Mainly, it implements a white list strategy. While certified MIDlets have all the privileges they need, uncertified applications have very little access to the system independently from their actual behaviour, leading to a significant reduction of their usability. On the other hand, executing a malicious, signed application can have obvious, dramatic consequences. There are many ways in which this attack can take place. A simple attacking scenario is based on the user's unawareness about security. Basically, a device owner wanting to install a MIDlet could decide to ignore whether it is not signed. This scenario is becoming popular, for instance, with local providers offering small, contextual applications (*e.g.*, catalogues, interactive guides). Often, MIDlet spots dispatch unsigned or self-certified applications to users moving inside some area of interest (*e.g.*, a museum).

Another danger arises from the hierarchical structure of certificates. In fact, when purchasing a certificate, the owner is often authorized to produce and distribute sub-certificates. The features of a sub-certificate depend on the structure of the original one (*e.g.*, a certificate can generate sub-certificates with an expiration date lower or equal to its own). For instance, an attacker acquiring a certificate can use it for signing a malicious MIDlet. Then, after detecting the attack, it should be possible, analysing the certificate, to trace back the certificate history and discover what went wrong in the sub-certificates chain. However, this is a reactive approach that can lead to identifying misbehaving entities (CAs, developers, vendors), while, in general, a proactive solution would be preferable.

For these reasons we advocate a *contract-based* approach for mobile applications trust management, referred to as Security-by-Contract-with-Trust (S×C×T). In our model contracts are in charge of providing guarantees on the correct behaviour of programs. Contracts are automatically produced by any provider and attached to the code. The counterpart of contracts are security policies. Policies define which behaviours are considered to be safe. To implement this strategy, we present a contract monitoring framework responsible for verifying whether a running application respects its contract. When an attack, namely an attempt to violate a contract, is detected, our system reacts immediately by enforcing a security policy and preventing the attack from being actually performed. Moreover, a contract breaking causes an automatic modification of the trust relationship between the device and the authority providing the contract. In particular, we propose a mechanism for managing trust feedback according to the concept of *mobile application criticality*. The degree of criticality reflects how much an application may be considered critical with respect to security and trust aspects according to its type and category (*i.e.*, according to which kind of data it may access or which resources it uses). Hence, our system can immediately react to threats and prevent further attacks coming for the same source.

In order to show how the proposed framework works, we present an application of the S×C×T to a Mobile Applications Marketplaces (MAMp) scenario, like, for instance, Apple AppStore, Cydia, Android Market, in which mobile applications (MA) are released by some vendors and customers can download and install them on their devices. Our mechanism manages trust by rewarding and penalising MA's provider according to the MA's trust recommendation given by the MAMp and the MA criticality. Furthermore, the proposed framework offers a high degree of flexibility providing applications clients with a reliability feedback and assuring security guarantees also under pervasive, contextual mobility conditions.

This paper is structured as follows: Section 2 presents our proposed extension of the security-by-contract paradigm with trust measurement. In Section 3 we show the application of the Security-by-Contract-with-Trust to the case study of a mobile application marketplace and in Section4 we present some simulation results. Section 5 relates our contribution to previous ones already in literature and Section 6 provides the conclusion of the paper and our future work.

2 Security-by-Contract-with-Trust

The main novelty of our approach consists in integrating the Security-by-Contract (S$\times$C) paradigm with a monitoring infrastructure for trust management by exploiting Role-based Trust Management Language (RTML) to deal with both trust and reputation management. In particular we aim to automatize the updating of the trust relationship between users and application/contract providers.

We start by recalling the S$\times$C paradigm in its original formulation.

2.1 Security-by-Contract Paradigm

The *Security-by-Contract* (S$\times$C) [5] paradigm provides a full characterisation of the contract-based interaction. Indeed, the basic notion of the Security-by-Contract paradigm is the *contract*, which is an over-approximation of all possible application execution behaviours, provided in addition to the application itself. Loosely speaking, a contract contains a description of the relevant features of the application and the relevant interactions with its host platform. The contract is released by the developer of the application.

The other cornerstone of the S$\times$C approach is the concept of *policy* that is usually specified on the platform on which the application is supposed to run. It may be specified by the owner or by the producer of the platform and consists of the set of admitted application execution behaviours.

The core idea behind the Security-by-Contract approach is depicted in Figure 1. When a client receives an application, the system automatically checks the formal correspondence between code and contract (Check Evidence). This step is intended to provide a formal proof that the contract effectively denotes the behaviour of the running program. This step can be implemented, for instance, using the *model-carrying code* [19] method. If the result is negative then the monitor runs to enforce the policy (Enforce Policy), otherwise a matching between the contract and the policy is performed to establish if the contract is compliant with the policy. If it is the case than the application is executed without overhead (Execute Application), otherwise the policy is enforced again (Enforce Policy). Finally, if the previous checks were positively passed, the MIDlet can be executed with no active runtime monitor.

The contract-policy matching function ensures that any security relevant behaviour allowed by the contract is also allowed by the policy. This matching could be done with respect to different behavioural relation, *e.g.*, language inclusion [4] or simulation relation [10]. This matching function allows the user to check whether the behaviour of the MIDlet is compliant with the policy on his device or not, without the need of running the MIDlet.

The enforcing approach has been shown to be feasible on mobile devices. In particular two techniques have been detailed in the literature and exploited for experiments and tools: JVM customization [1] and bytecode in-lining [3]. Briefly, the first replaces the standard JVM with a modified one dispatching signals to the monitoring agent whenever a program makes a call to (a subset of) the system APIs. The second instruments the sequence of bytecode instructions with invocations to the security policy monitor making the program send security signals at run-time. Both approaches use an external component, namely a *Policy Decision Point* (PDP), holding the set of rules that compose the security policy. Moreover the PDP reads the current device state (battery consumption, link strength, available credit) through dedicated internal components. When the PDP receives a request for an action violating the security policy, it answers denying the necessary permission. Then, the system reacts by throwing an exception. Note that the notion of trust was not integrated in the Security-by-Contract approach. Basically, the user can trust or not the application's provider according to a software certification delivered by a certification authority.

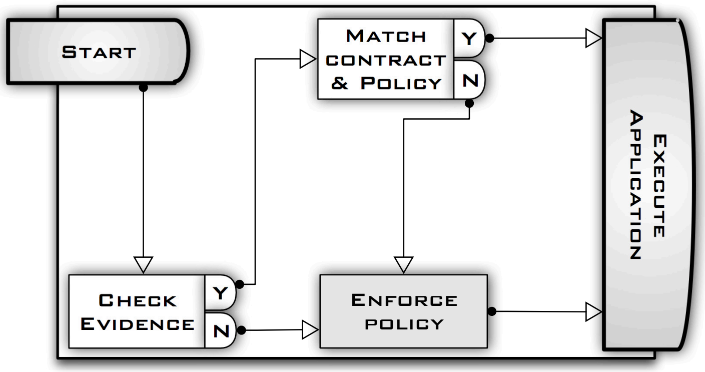


Figure 1: The Security-by-Contract process [5].

2.2 Extending S$\times$C with Trust

As mentioned above, a crucial point of the S$\times$C architecture is the verification of the relation that exists between the MIDlet and its contract. Nowadays the mobile code is run if its source is trusted. This means that we can only reject or accept the signature of the application provider.

Here we propose an extension of the existing architecture by adding a component for the contract monitoring. Checking whether the execution of an application adheres to declared contract we can modify the level of trust of the application provider.

Driving trust management with contract monitoring offers several advantages to both the mechanisms. Basically, in order to have a precise trust measure, a behavioural feedback is needed. Indeed, the trust weight associated to a certain provider should change according to whether its applications behave correctly or not. In some cases, the trust value modification is triggered by a program trying to execute some forbidden action, namely violating a security policy. However, policy violations are not always caused by a real security attack. As a matter of fact, in case of customised security policies, we can not expect that every provider is aware about the requirements of each customer. Hence, a policy violation could simply be the result of a mismatch between the provider's specification and the platform restrictions.

On the other side, a precise trust value has some clear advantage for the platform. A first benefit derives from the possibility of avoiding security enforcement. Two important issues about policy enforcement are execution overhead and semantics interference. In other words, when an enforcement mechanisms follow the execution of a program, it slightly decreases the standard performances. Furthermore, enforcing a policy may cause a modification of the original behaviour of programs. If a program comes from a trusted source, the user could decide to run it free from any control.

A second important issue is policy specification. Users applying customised security properties over their devices must specify all the acceptable behaviours of running applications. Needless to say that this can be a cumbersome task especially for users having no technical skills. In fact, many users do not specify any security restriction on the applications they trust, *e.g.*, utilities provided by the device's manufacturer. Hence, allowing trusted application to run according to their contracts may also reduce the complexity of specifying security policies.

Our strategy takes place in two phases: at deploy-time by setting the monitoring state and at run-time by applying the contract monitoring procedure for adjusting the provider trust level.

2.2.1 Deployment Architecture

This S×C paradigm (Figure 1) does not require the software provider to be a trusted entity and simply relies on the correctness of local, internal components (*i.e.*, Check Evidence and Contract-Policy Matching). Here, we deploy a framework for quantitative trust management. In this way it is possible to dynamically update the levels of trust. Updating is done according to the adherence between the real execution of the application and its contract. Indeed, we extend the existing architecture by adding a

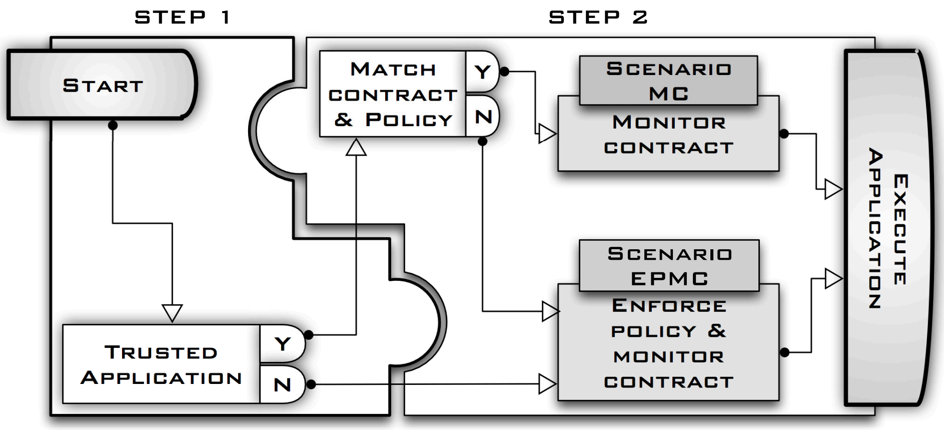


Figure 2: The Extended Security-by-Contract Application workflow.

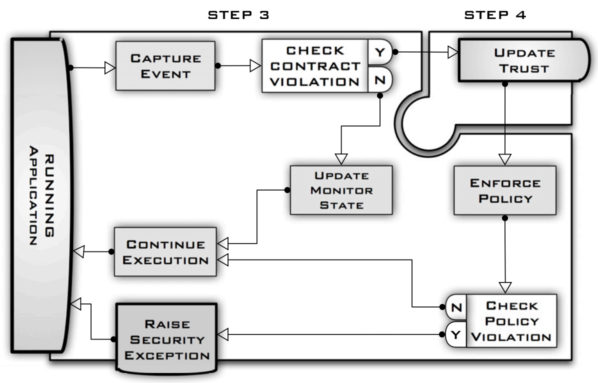
contract monitoring that checks the compliance between the application and its contract. We replace the contract verification (Check Evidence) with a simple check on the level of trust of the provider (Trusted Provider). If the provider is untrusted then the policy is enforced, otherwise the contract-policy matching is performed. Hence the S×C×T workflow results as in Figure 2 and Figure 3. It consist of the following steps:

- **Step 1-*Trust Assessment*:** Each downloaded mobile application comes with a given recommendation rate, which allows the trust module of the user device to decide if the application can be considered as trusted or not (see section 3.1).

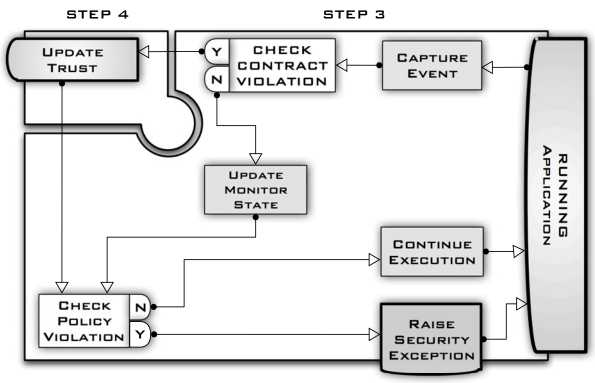
- **Step 2-*Contract Driven Deployment*:** According to this trust measure, the security module defines if just monitoring the contract or both enforce the policy and monitoring the contract going into one on the scenarios described in Step 3 (see section 3.2). Indeed,we have two cases:

  Scenario MC The contract satisfies the policy. In this case our monitoring/enforcement infrastructure is required to monitor only the application contract. Indeed, under these conditions, contract adherence also implies policy compliance. If no violation is detected then the application worked as expected. Otherwise, we discovered that a trusted party provided us with a fake contract. Our framework reacts to this event by reducing the level of trust of the indicted provider and switching to the policy enforcement modality.

  Scenario EPMC The contract does not satisfy the policy. Since the contract declares some potentially undesired behaviour, policy enforcement is turned on. Similarly to a pure enforcement framework, our system guarantees that executions are policy-compliant. However, monitoring contract during these executions can provide a useful feedback for better tuning the trust vector. Hence, our framework also allows for a mixed monitoring and enforcement configuration. This configuration is activated on a statistical base.



(a) Scenario MC



(b) Scenario EPMC

Figure 3: The contract monitoring configurations

Let us notice that, in both the previous scenarios, contract monitoring plays a central role. Indeed, a contract violation denotes that a trusted provider released a fake contract.

- **Step 3-*Contract Monitoring vs Policy Enforcement Scenarios***: Depending on the chosen scenario the security module is in charge to monitor either the policy or the contract and save the execution traces (logs) (see section 3.2).
- **Step 4-*Trust Feedback Inference*:** Finally, the trust module parses the S×C×T produced logs and infers trust feedback (see section 3.3).

2.2.2 Description of the Scenarios

We outline how trust measures assigned to security assertions can be adjusted as a result of a contract monitoring strategy. Indeed, trust measures associated with the provider concern on the contract goodness mainly. Updated trust measures will influence on future interactions with an application and contract providers. In other words, our system penalizes the provider more when the contract does not specify application's behaviour correctly, rather when the application itself contradicts user's security policy.

In [3] a monitoring infrastructure consists in a PDP that holds the actual security state and is responsible for accepting or refusing new actions and *Policy Enforcement Points* (PEPs) that are both in charge of intercepting actions to be dispatched to the *policy decision point* (PDP) and preventing the execution of not allowed operations.

Starting from this model, we extend it by making the PDP also responsible for the contract monitoring operations and for the trust vector updating. According to [1, 3], we assume that both contracts and policies are specified through the same formalism. Hence, the policy enforcement configuration of the PDP keeps unchanged. The PDP must load application contracts as well as security policies dynamically. Moreover, it must be able to run under the two different execution scenarios in Figure2.

EPMC Scenario. Both the policy enforcement and the contract monitoring are active. During the execution contract violations are checked for updating trust levels and policy enforcement is activated in order to guarantee that the application does not violate the security policy on the device. Indeed, the contract monitoring receives event signals from the executing code and keeps trace of the execution trace. When a signal arrives, its consistency with respect to the monitored contract is checked. If the contract is respected then its internal monitoring state is updated and the operation is allowed, and a good behaviour is logged (*i.e.*, contract respected). Otherwise, if a violation attempt happens, a security error occurs and a violation feedback is logged for the trust module. The policy enforcer is only in charge to following the execution of the application and whenever it attempts to violate the security policy of the device the enforcement mechanism halts the execution in such a way the security policy is satisfied.

MC Scenario. The contract monitoring is performed. It works according to the following strategy: the contract monitoring receives event signals from the executing code. The execution trace is kept in memory. When a signal arrives, its consistency with respect to the monitored contract is checked. As in the previous case, if the contract is respected then its internal monitoring state is updated and the operation is allowed, and a good behaviour is logged (*i.e.*, contract respected). Otherwise, if a violation attempt happens, a security error occurs, and a bad feedback is trigged (*i.e.*, contract violation), and the system switches from contract monitoring to policy enforcement configuration in order to guarantee that the security policy is satisfied. Since an instance of the policy is always present, this operation does not imply a serious computational overhead.

Summing up, both execution scenarios check contract violations through the contract monitoring strategy described above and update providers' trust level according to the contract monitoring feedback.

3 A Case Study: The Mobile Application Market-Place

Let us consider a Mobile Applications Marketplaces (MAMp), like, for instance, Apple AppStore, Cydia, Android Market, in which mobile applications (MAs) are released by a provider (MAP) and customers can download and install them on their devices. Let us also suppose that a trust reputation is associated to each MAP. Reputations are managed by MAMp and updated according to customers' feedback.

Generally speaking, we consider a customer, owner of a mobile device, who needs to add a certain functionality to his own device. The customer asks for an application for doing that. Such MA is downloaded from the MAMp.

The customer decides to download or not the mobile application according to the given recommendation and the trust threshold he has set on his device. Each mobile application comes with its contract released by the provider of the application itself.

Let us suppose that a security policy is embedded on the customer's device. It can be set by the customer himself or by the manufacturer of the device. If the application is downloaded, whenever it is executed, some security mechanisms are needed for guaranteeing that the device's security policy is satisfied.

In this scenario we have applied the Security-by-Contract-with-Trust in order to guarantee that a downloaded application is executed without violating the security policy required by the customer.

In the following sessions we detail the MA lifecycle according to the S$\times$C$\times$T workflow.

3.1 Trust Assessment - Step 1

MAMp computes trust recommendations for mobile applications only by aggregating subjective customers' opinions. However, this aggregation process does not take into account security and privacy feedback. Thus, we introduce a new parameter that reflects the global reputation of a Mobile Application Provider (MAP) in term of security behaviour in such a way that the provider's reputation grows according to the number of times the application satisfies its contract, and vice versa.

In the literature, managing reputation is widely investigated. Indeed, the reputation can be managed (i) locally by the customer himself, (ii) by a centralised entity (*e.g.*, ebay), or (iii) in a distributed way, where the reputation of each entity is managed by other ones with a distributed hash table such as in CAN [16] or Chord [20]. We consider MAMp as a trusted entity, since, the MAMp is often managed by the company that released the mobile device (*e.g.*, Nokia) and/or the mobile operating system (*e.g.*, Google). Hence, we assume that each MAMp works as a centralised reputation manager that maintains MAPs' reputation. Note that, the MAMp considers only accredited feedback, *i.e.*, all the feedback that can not be reproduced.

Therefore, the trust recommendation value of a MA is enhanced by weighting it by the reputation of its provider, as follows:

$$Rec(MAMp, MA) = Rep(MAP) * AGG(MA, Customer_{opinion}) \tag{1}$$

where, the amount of the recommendation is denoted by *Rec*(MAMp, MA); the $AGG(MA, Customer_{opinion})$ represents the aggregation process of users' opinions that is performed by the MAMp (this value is normalised into the interval [0,1]); the *Rep*(MAP) falls into the interval [0,1] and denotes the reputation of the provider (MAP) that releases MA.

In order to decide if a given mobile application from MAMp is trusted or not, each customer has to fix its trust threshold Th^0 (bounded between 0 and 1). Hence, the ones that their recommendation is over Th^0 are considered as *trusted*, where those with a recommendation below than Th^0 are *untrusted*.

The trust threshold is set according to the criticality of the mobile application. For instance, a mobile application that manipulates users' localisation can be considered as a risky application and hence will lead to define a high Th^0 close to '1'. Whereas, for a harmless application, Th^0 will be close to '0'.

For bootstrapping, we assume that each MAP is initialised with a low reputation (i.e., 0.1) in order to reduce Whitewashing phenomena[11], where malicious consumers with low reputation leaves the system and then backs with a new identity. Furthermore, neither this kind of attack nor the Sybil attack[11] are efficient, since in MAMp is very tedious to create multiple accounts especially if it requires an expense from the part of the attacker.

3.2 Monitoring/Enforcement Process - Step 2 and Step 3

Once the Trust module established the threshold according to the application criticality and once it gets the trust recommendation of the MA, we have the following two possibilities:

(i) The customer does not trust that the MA behaviour adheres to the MA contract. In this case the contract policy matching is not performed since it does not provide any guarantee about the compliance between the MA behaviour and the policy. Hence we turn to the *Enforce policy & Monitor Contract* scenario (EPMC scenario of Figure 2) described later. This allows us to guarantee that the security policy is respected by applying the enforcement mechanism and trust feedback are provided according to the contract monitoring answer.

(ii) The customer trusts the MA, *i.e.*, the customer trusts that the MA behaviour adheres to the MA contract. In this case we check if the contract satisfies the policy by the *Contract-Policy Matching* function. Two subcases arise:

- The contract satisfies the policy. This allows the customer to establish that the MA satisfies also the security policy. For that reason, the MA is executed and we turn into the *Monitor Contract* scenario (MC scenario of Figure 2).
- The contract does not satisfy the policy. Hence, we are not able to infer anything about the compliance between the MA behaviour and the security policy. The enforcement mechanism is applied

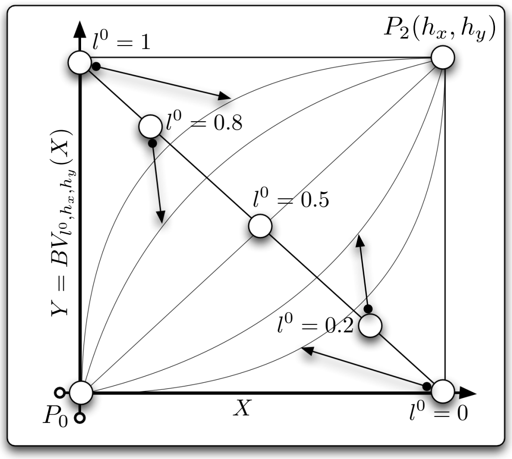
for guaranteeing that the security policy is respected. Thus, also in this case, we turn into the *Enforce Policy & Monitor Contract* scenario (EPMC scenario of Figure 2). The contract monitoring can provide a useful feedback for better tuning MAPs' reputation. The contract monitoring is performed only for providing feedback to the trust module. It is performed on statistical bases according to a probability called P_{mon}. P_{mon} is computed according to the level of the given recommendation and the remaining battery life. Indeed, P_{mon} is inversely proportional to the measure of trust of the application and proportional to the remaining battery life. For instance, the more trusted the application is or the lower the battery life is, the less frequent the contract monitoring is performed.

We implement the behaviour function *BV* [17] in order to compute P_{mon}. The *behaviour function BV* is monotonically increasing function, and is able to oscillate from parabolic to hyperbolic shape according to a behaviour level called l^0.

Note that it is possible to use other mathematical functions (logarithmic, exponential or stepwave) to assess the monitoring probability but the originality of using Bezier curves comes from the easiness of plotting different monotonic and increasing curves (i.e., logarithmic-like and exponential-like) with a unique function taking as inputs only two parameters, namely the curve thresholds and the degree of the curvature.

Definition 3.1. *[17] [BV_{l^0,h_x,h_y}: The behaviour Function] The BV function is the Cartesian form of a quadratic Bezier curve. Thus, the BV function is able to draw smooth curves (see Figure4) which are achieved through three points P_0, P_1 and P_2, starting at P_0 going towards P_1 and terminating at P_2, where:*

- $P_0(0,0)$ *is the origin point.*
- $P_2(h_x,h_y)$ *is called the threshold point, where the h_x and h_y represent respectively the abscissa and the ordinate thresholds.*
- $P_1(b_x,b_y)$ *defines the behaviour point. The coordinate of this point (b_x,b_y) are guided by a given level 'l^0' ($l^0 \in [0,1]$) through the diagonal of the rectangle that is defined by P_0 and P_2,.*



$$
\begin{array}{rcl}
BV : [0,h_x] & \longrightarrow & [0,h_y] \\
X & \longrightarrow & Y
\end{array}
$$

$$
BV_{l^0,h_x,h_y}(X) =
$$

$$
\begin{cases}
\frac{(h_y-2b_y)}{4b_x^2}X^2 + \frac{b_y}{b_x}X & si\ (h_x - 2b_x = 0) \\
(h_y - 2b_y)(\propto(X))^2 + 2b_y \propto(X), & si\ (h_x - 2b_x \neq 0)
\end{cases}
$$

$$
Where \begin{cases}
\propto(X) = \frac{-b_x+\sqrt{{b_x}^2-2b_x*X+h_x*X}}{h_x-2b_x} \\
0 \leq b_x \leq h_x \wedge h_x > 0 \\
b_x = (1-l^0)\cdot h_x \wedge b_y = h_y \cdot l^0
\end{cases}
$$

Figure 4: The BV function.

Thus, by aligning the behaviour level l^0 with the trust level of the application $Rec(MAMp,MA)$ (*i.e.*, $l^0 = 1 - Rec(MAMp,MA)$) and by representing, through the abscissa, the battery life $Battery_{remain}$, we obtain the following equation:

$$\mathscr{P}_{mon}(C) = BV_{(1-Rec(MAMp,MA)),1,1}(Battery_{remain}) \tag{2}$$

Table 1 shows some examples of monitoring probabilities that are obtained by varying the trust recommendation value of the mobile application and the percentage of the remaining battery. Hence, as expected, the probability decreases when the battery life falls (proportionately) or the given recommendation become higher (inversely proportional).

$Rec(MAMp,MA))$ \ $Battery_{remain}$	20%	40%	60%	80%	100%
0.2	0.47	0.70	0.84	0.93	1.00
0.5	0.20	0.40	0.60	0.80	1.00
0.8	0.06	0.16	0.30	0.53	1.00
1.0	0.01	0.05	0.16	0.31	1.00

Table 1: Monitoring probability examples ($P_{mon}(C)$).

3.3 Trust Feedbacks -Step 4

According to the previous steps, a trust log that reflects the behaviour of the deployed application throughout the monitoring process is generated. Table 2 represents all possible logs by crossing all possible behaviours with a trusted and an untrusted case. Thus, four types of feedback are highlighted, namely, Highly Reward, Reward, Penalise, and Highly Penalise.

Trust \ Security	Respect the contract		Violate the contract	
	MC scenario	EPMC scenario	MC scenario	EPMC scenario
Trusted	Highly reward	Reward	Highly penalise	
Untrusted	——	Reward	——	Penalise

Table 2: Feedback types.

As illustrated in Table 2, for a trusted application, it is obvious to reward the corresponding MAP if that application respects its contract but we have to turn on the EPMC scenario, and to highly reward if the application matches the contract and we are in the MC scenario. On the contrary, we highly penalise MAP if its recommended application violates its contract.

However, for an untrusted application, the security by contract is only applied under the EPMC scenario, whose aim is to check the contract and enforce the policy. In this case, we adopt a reserved behaviour, by just rewarding if the contract is respected and penalising otherwise.

In order to reduce the impact of malicious customers on our framework (i.e., Slandering and denial of services attacks [11]), in which they try to send multiple bad feedback for good application or several good feedback for bad ones, each customer can only provide one reward and one penalty feedback for each mobile application. Furthermore, as mentioned above, each given recommendation is processed, by our model, proportionately to the reputation of its sender.

Then, MAMp performs the Algorithm 1, with the given trust feedback, to update MAP's reputation which directly results in updating all the recommendation of the applications of that provider.

MAMp updates the MAP's reputation (line 18) by increasing or decreasing current evaluation proportionately to α and also to the quantitative variable $Feedback\_amount$. The $Feedback\_amount$ is respectively equal to '0.5' for reward or penalise and '1' for highly reward or highly penalise.

Algorithm 1 Update trust

```
Require:
    $MA$: The monitored Mobile Application
    $Feedback \in \{”Reward”, ”Highly\ reward”, ”Penalise”, ”Highly\ penalise”\}$
    $\alpha \in [a,b]$

 1: $MAP = Pr(MA)$ {$Pr(MA)$ returns the MAP of MA}
 2: $OldPr = Rep(MAP)$ {$Rep(MAP)$ returns or updates the current reputation of MAP}
 3: if $Feedback = ”Reward”$ or $Feedback = ”Highly\ reward”$ then
 4:    if $Feedback = ”Reward”$ then
 5:       Feedback_amount=0.5
 6:    else if $Feedback = ”Highly\ reward”$ then
 7:       Feedback_amount=1
 8:    end if
 9:    $NewPr = OldPr + Feedback\_amount * (1 - OldPr)$
10: else if $Feedback = ”Penalise”$ or $Feedback = ”Highly\ penalise”$ then
11:    if $Feedback = ”Penalise”$ then
12:       Feedback_amount=0.5
13:    else if $Feedback = ”Highlypenalise”$ then
14:       Feedback_amount=1
15:    end if
16:    $NewPr = OldPr - Feedback\_amount * OldPr$
17: end if
18: $Rep(MAP) = (1 - \alpha) * OldPr + \alpha * NewPr$
```

The parameter α ($\alpha \in [a,b]$) defines how significant is the impact of new actions will be on the previous acquired trust history. We choose to bound α into the interval [a, b], where a is close to 0 (*e.g.*, $a = 0.2$) and b is close to '1' (*e.g.*, $a = 0.8$). Thus, $\alpha = a$ means that past acquired history is highly relevant, whereas $\alpha = b$ gives the highest impact to new actions.

In order to automatically infer the value of α, two statements are considered:

- α is proportional to the recommendation value. In fact, if the given recommendation of the application is low and a new good behaviour is monitored, MAMp has to slightly update trust values (*i.e.*, α is close to a). Whereas, if the given recommendation of the application is high and a bad behaviour is trigged, MAMp has to give a high impact to new computed feedback (*i.e.*, α is close to b).

- α is proportional to Th^0 (*i.e.*, application criticality) in case of a penalty and inversely proportional to Th^0 in case of a reward. Indeed, risky applications have to be less rewarded than harmless applications and inversely, risky applications have to be more strongly penalised than harmless applications.

Thus, as for contract monitoring process, α can be inferred using the BV function as follows:

$$\alpha = \begin{cases} BV_{Th^0,1,(b-a)}(REC) + a & if \text{ Feedback} \in \{Penalise, Highly\ penalise\} \\ BV_{(1-Th^0),1,(b-a)}(REC) + a & if \text{ Feedback} \in \{Reward, Highly\ reward\} \end{cases} \tag{3}$$

$where\ REC = Rec(MAMp, MA)$

Table 3 shows some examples of $alpha$ that are obtained by varying the criticality of the mobile application and the amount of the recommendation.

	Penalty					Reward			
$Rec(MA)$ / Th^0	0.2	0.4	0.6	0.8		0.2	0.4	0.6	0.8
0.8	0.48	0.62	0.70	0.76		0.24	0.30	0.38	0.52
0.2	0.24	0.30	0.38	0.52		0.48	0.62	0.70	0.76

Table 3: α computing examples.

4 Simulation and results

The aim of the following simulation is to figure out the complementarity between trust and security in enhancing the security by contract paradigm. We implemented a simulator in C++. The Simulator is able to randomly generate a MAMp by defining the number of MAPs, the number of MAs, the criticality level and the number of malicious MAs' providers.

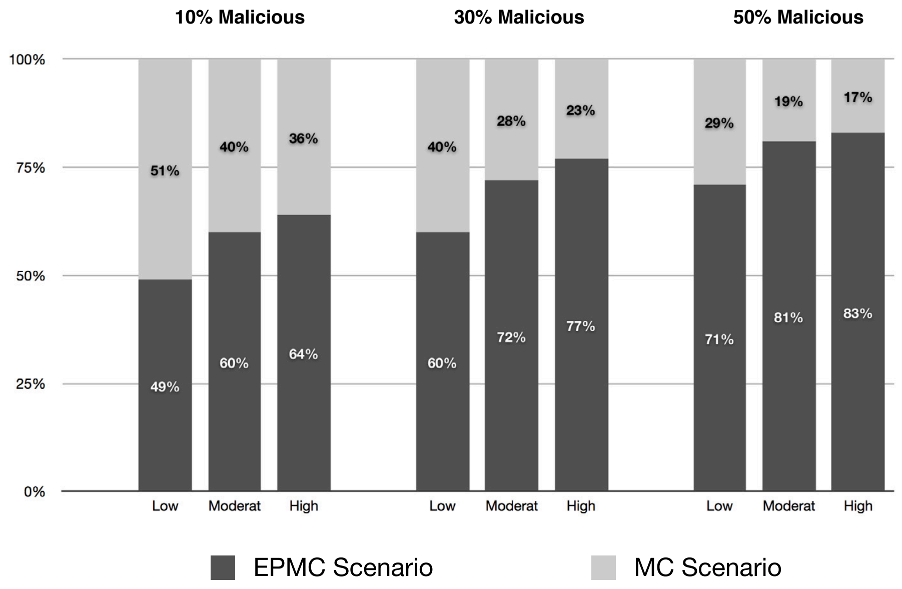


Figure 5: Percentage of activated scenarios.

We based our simulations on a MAMp with 100 MAPs, where each one provides around 10 applications. We perform our simulation by varying two parameters:
(i) The percentage of malicious MAPs which can be 10%, 30% and 50%. We consider that most of their applications are malicious and provide a high probability of violating local policies and/or their contract.
(ii) The criticality level of MAMp allows to define the category of applications that is proposed by the MAMp. We set up three categories, namely, applications with low criticality (*i.e.*, $0 \leq Th^0 < 0,33$), applications with moderate criticality (*i.e.*, $0,33 \leq Th^0 < 0,66$), and applications with high criticality (*i.e.*, $0,66 \leq Th^0 \leq 1$).

Then, in order to stabilise our result we performed each simulation through one hundred random MAMp. For this simulation, we count the percentage of activation for each execution scenario (*i.e.*, EPMC scenario and MC scenario) after a download of 200 applications. We represent the results in Figure 5, to compare usage percentage of each scenario during this monitored period. This figure shows that our framework works as expected and well balances between preserving battery lifetime (*i.e.*, performing the MC scenario) and enforcing both the security policy and monitoring the contract (*i.e.*, EPMC scenario). In fact, S×C×T becomes more suspicious, by activating the EMPC scenario to the detriment of the second one, when the number of malicious applications or the criticality of the applications increase, and inversely, relaxes the security policy into less risky environment (*i.e.*, low criticality application or less number of malicious applications) in order to maximise the devices' battery life.

5 Related work

To the best of our knowledge, there is not much work about the integration of trust management and policy enforcement for mobile code in literature. However, works about the integration of trust module into policy enforcement exist. In particular some work has been done for integrating trust management and fine grained access control in Grid Architecture. An attempt can be found in [12] where it is proposed an access control system the enhances the Globus toolkit with a number of features. This copes with the fact that access control policies and access rights management becomes one of the main bottleneck using Globus for sharing resource in a Grid architecture. Along this line of research [2] presents an integrated architecture, extending the previous one, with an inference engine managing reputation and trust credentials. This framework is extended again in [13] where it is introduced a mechanism for trust negotiating credential to overcome scalability problem. In this way the framework provided preserves privacy credentials and security policy of both users and providers. Even if the application scenario and the implementation are different, the basic idea consists in considering the trust as a metrics for deciding the reliability of an application provider.

Also [14] presents a reputation mechanism to facilitate the trustworthiness evaluation of entities in ubiquitous computing environments. It is based on probability theory and supports reputation evolution and propagation. The proposed reputation mechanism is also implemented as part of a QoS-aware Web service discovery middleware and evaluated regarding its overhead on service discovery latency. On the contrary, our approach is not probabilistic. We provide a method according to which we update the level of trust of an application provider.

Four main approaches to mobile code security can be broadly identified in the literature.

The sandbox model [9] limits the instructions available for use, with the weakness that it provides security only at the cost of unduly restricting the functionality of mobile code (e.g., the code is not permitted to access any files). The sandbox model has been subsequently extended in Java 2 [8], where permissions available for programs from a code source are specified through a security policy.

Cryptographic code-signing is used for certifying the origin (i.e. the producer) of mobile code and its integrity, typically by means of private/public keys to sign/verify the executable content. Several limitations of this approach can be identified [6]. A key weakness is that in the signing process it is not checked at all if the application is doing things not wanted by the user e.g., sending data with information the user does not want to be sent (F-Secure Weblog (www.f-secure.com/weblog) 11/05/2007, "Just because it's Signed doesn't mean it isn't spying on you").

The *Proof-Carrying Code* (PCC) approach [15] enables safe execution of code from untrusted sources by requiring a producer to furnish a proof regarding the safety of mobile code. Then the code consumer uses a proof validator to check, that the proof is valid and hence the foreign code is safe to execute. The PCC approach is problematic for several reasons [18], such as that automatic proof generation for complex properties is still a daunting problem, making the PCC approach not suitable for real mobile applications.

The *Model-Carrying Code* (MCC) approach is strongly inspired by PCC, sharing with it the idea that untrusted code is accompanied by additional information that aids in verifying its safety [19]. With MCC, this additional information takes the form of a model that captures the *security-relevant behaviour* of code, rather than a proof.

6 Conclusion and Future Work

In this paper we presented a contract-based trust management framework for mobile applications. At deploy-time, the monitoring structure is decided depending on both the application contract and the credentials (*i.e.*, trust measure) of the contract releaser. The main novelty of our model consists in the contract monitoring scenario. At run-time, a trusted program violating its contract leads to a correction of the trust relationship with the provider and activates the policy enforcement configuration of our system.

As an example of a possible trust management strategy, we have applied the SxCxT framework to a mobile application marketplace scenario by presenting mechanism for managing trust feedback according to the concept of mobile application criticality. In particular, we have presented a mechanism for rewarding and penalising mobile application developers according to its level of the recommendation that is provided by the Mobile Application Marketplace and the criticality of the application.

The information about the amount of the recommendation can be retrieved from the MAMp and the contract of the application that the developer has to send in addition to the application itself, respectively. This means that we have a unified architecture for managing both security and trust.

Many future directions are viable. Mainly our trust management strategy is still a work in progress. Currently, trust weights can only decrease monotonically as a consequence of contract violations with the only exception of a direct intervention of the user. Also the contracting infrastructure can be further improved. As a matter of fact, in this work we only referred to single-contract applications. However, we can extend our model in order to accept more contract instances for a single program. This scenario seems to be realistic and open new directions of investigation.

We also envision to extend the trust module to allow banishing malicious applications and/or developers from MAMp by taking into account the time fading aspect.

Finally, similarly to [3], we plan to implement a working prototype for testing the practical feasibility of our approach including performance and resources consumption analysis.

Acknowledgement

We thank the anonymous referees of IMIS10 for valuable comments that helped us to improve this paper.

References

[1] Alessandro Castrucci, Fabio Martinelli, Paolo Mori, and Francesco Roperti. Enhancing java me security support with resource usage monitoring. In *ICICS*, pages 256–266, 2008.

[2] Maurizio Colombo, Fabio Martinelli, Paolo Mori, Marinella Petrocchi, and Anna Vaccarelli. Fine grained access control with trust and reputation management for globus. In *OTM Conferences (2)*, pages 1505–1515, 2007.

[3] Gabriele Costa, Fabio Martinelli, Paolo Mori, Christian Schaefer, and Thomas Walter. Runtime monitoring for next generation java me platform. *Computers & Security*, July 2009.

[4] Lieven Desmet, Wouter Joosen, Fabio Massacci, Pieter Philippaerts, Frank Piessens, Ida Siahaan, and Dries Vanoverberghe. Security-by-contract on the .net platform. volume 13, pages 25–32, Oxford, UK, UK, 2008. Elsevier Advanced Technology Publications.

[5] N. Dragoni, F. Martinelli, F. Massacci, P. Mori, C. Schaefer, T. Walter, and E. Vetillard. Security-by-contract (SxC) for software and services of mobile systems. In *At your service - Service-Oriented Computing from an EU Perspective.* MIT Press, 2008.

[6] N. Dragoni, F. Massacci, T. Walter, and C. Schaefer. What the heck is this application doing? - a security-by-contract architecture for pervasive services. To appear in *Computers & Security*, Elsevier.

[7] A. Lazouski F. Martinelli F. Massacci G. Costa, N. Dragoni and I. Matteucci. Extending Security-by-Contract with quantitative trust on mobile devices. In *Proceeding of CISIS 2010, The Fourth International Conference on Complex, Intelligent and Software Intensive Systems*, pages 872–877, 2010.

[8] L. Gong and G. Ellison. *Inside Java(TM) 2 Platform Security: Architecture, API Design, and Implementation*. Pearson Education, 2003.

[9] Li Gong. Java Security: Present and Near Future. *IEEE Micro*, 17(3):14–19, 1997.

[10] Paolo Greci, Fabio Martinelli, and Ilaria Matteucci. A framework for contract-policy matching based on symbolic simulations for securing mobile device application. In *ISoLA*, pages 221–236, 2008.

[11] K. Hoffman, D. Zage, and C. Nita-Rotaru. A survey of attack and defense techniques for reputation systems. *ACM Computing Surveys (CSUR)*, 42(1):1–31, 2009.

[12] H. Koshutanski, F. Martinelli, P. Mori, L. Borz, and A. Vaccarelli. A fine grained and x.509 based access control system for globus. In *OTM*, pages 1336–1350. Springer, 2006.

[13] Hristo Koshutanski, Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. Enhancing grid security by fine-grained behavioral control and negotiation-based authorization. *Int. J. Inf. Sec.*, 8(4):291–314, 2009.

[14] Jinshan Liu and Valérie Issarny. An incentive compatible reputation mechanism for ubiquitous computing environments. *Int. J. Inf. Secur.*, 6(5):297–311, 2007.

[15] George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Langauges (POPL '97)*, pages 106–119, January 1997.

[16] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[17] Rachid Saadi, Jean Marc Pierson, and Lionel Brunie. T2D: A Peer to Peer trust management system based on Disposition to Trust. In *25th ACM Symposium On Applied Computing (SAC)*, pages 1472–1478. ACM Press, 2010.

[18] R. Sekar, C. R. Ramakrishnan, I. V. Ramakrishnan, and S. A. Smolka. Model-Carrying Code (MCC): a New Paradigm for Mobile-Code Security. In *NSPW '01: Proceedings of the 2001 Workshop on New security paradigms*, pages 23–30, New York, NY, USA, 2001. ACM Press.

[19] R. Sekar, V.N. Venkatakrishnan, Samik Basu, Sandeep Bhatkar, and Daniel C. DuVarney. Model-carrying code: a practical approach for safe execution of untrusted applications. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 15–28, 2003.

[20] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

Gabriele Costa is a Ph.D. student in Computer Science at University of Pisa and a researcher of the security group of the National Research Council (CNR). His research interests include foundational and practical aspects of programming language security. He is currently involved in the EU projects CONNECT, Aniketos and NESSoS. Detailed information about his publications and activities can be found at `http://www.di.unipi.it/~costa`.

Nicola Dragoni obtained a M.S. Degree in Computer Science in 2002 and a Ph.D. in Computer Science in 2006, both at University of Bologna, Italy. From 2002 to 2006 he also worked as Research Assistant at the Department of Computer Science at the same University. He visited the Knowledge Media Institute at the Open University (UK) and the MIT Center for Collective Intelligence (USA), respectively in 2004 and 2006. In 2007 and 2008 he joined University of Trento as post-doctoral Research Fellow working on the S3MS project. Between 2005 and 2008 he also worked as freelance IT consultant. Since 2009 he is an assistant professor in security and distributed systems at the Department of Informatics and Mathematical Modelling at Denmark Technical University (DTU).

Valérie Issarny is Directrice de Recherche at INRIA, France. Since 2002, she is the head of the INRIA ARLES research project-team at the Paris-Rocquencourt research center. Her research interests relate to distributed systems, software engineering, middleware, pervasive computing, service-oriented computing and trust management. She has been involved in a number of European and French projects and initiatives in the above areas. She has further been member of organization and program committees of leading events in those areas. She is in particular steering committee member of the Middleware and ESEC/FSE conferences. Further information about Valerie's research interests and her publications can be obtained from the ARLES Web site at `https://www-roc.inria.fr/arles/index.php/members/94-valerie-issarny.html`.

Aliaksandr Lazouski received M.Sc. in Electronics from Belorussian State University in 2006. He is currently a PhD student in computer science department at the University of Pisa in collaboration with IIT-CNR. His research interests include access and usage control models, trust management, Grid computations.

Fabio Martinelli is a senior researcher of IIT-CNR where He leads the information security group. He is co-author of more than 100 papers on international journals and conference/workshop proceedings. His main research interests involve security and privacy in distributed and mobile systems and foundations of security and trust. He usually teaches at graduate level courses in computer security. He serves as PC-chair/organizer in several international conferences/workshops. He has been guest co-editor of 7 books/journals. He is the co-initiator of the International Workshop series on Formal Aspects in Security and Trust (FAST), one of the first events to recognize the necessity to consider trust and security issues altogether in new virtual and dynamic organizations/coalitions. He has a specific expertise on running summer research schools on topics related to cyber security. In particular, He is serving as scientific co-director of the international research school on Foundations of Security Analysis and Design (FOSAD) since 2004 edition. He has been recently awarded by NATO as co-director for a Advanced Training Course . He promoted and chaired the WG on security and trust management (STM) of the European Research Consortium in Informatics and Mathematics (ERCIM) and He is currently member of the Executive Committee of the IFIP Trust Management WG. He usually manages R&D projects on information and communication security. In particular He is currently the project coordinator of NESSoS Network of Excellence on Engineering Secure Future Internet Services funded by the EU.

Fabio Massacci is a professor of computer security at the University of Trento, Italy. His research interests are in security requirements engineering and security verification for mobile systems. Massacci received a PhD in computer science and engineering from Sapienza University of Rome, Italy. He has published more that 100 papers in security conferences, journals and book chapters and has been the European scientific and administrative coordinators for many multimillion EU projects in Security and Trust. He is the co-founder and steering board member of the ESSOS (Engineering Secure Software and Systems) conference series. He got a PhD from the University of Rome ”La Sapienza”. He is a member of the ACM, IEEE, and ISACA. Contact him at `fabio.massaci@unitn.it`.

Ilaria Matteucci is a researcher of the information security group at IIT-CNR and co-author of several scientific publications. Her main research interest involves formal models for security by focusing on the problem of the synthesis of secure systems and controllers. In particular, Matteucci’s current research interests include the study of process algebra based techniques for run-time enforcement of security properties in such a way that considered systems, ranging from distributed systems to web services, will be secure. She also participates in European projects in the area of security on mobile devices and of data-centric context-aware information sharing. Additionally, she is/has been involved in the following EU projects: CONNECT, CONSEQUENCE, S3MS and the network of excellence NESSoS.

Rachid Saadi is is currently a post-doc at INRIA Paris-Rocquencourt within the INRIA ARLES research project-team. He got his Ph.D in Computer Engineering at INSA de Lyon in 2009. His main topics of interest include: security and trust management, pervasive systems, ubiquitous computing and Software engineering. More information about his research interests and his publications can be obtained from his INRIA web site at `http://www-roc.inria.fr/~rsaadi`.