# Enforcing Mobile Application Security Through Probabilistic Contracts

Fabio Martinelli[2], Ilaria Matteucci[2],
Andrea Saracino[1 2], and Daniele Sgandurra[2]

[1] Dipartimento di Ingegneria dell'Informazione,
Università di Pisa, Pisa, Italy
name.surname@iet.unipi.it
[2] Istituto di Informatica e Telematica,
Consiglio Nazionale delle Ricerche, Pisa, Italy
name.surname@iit.cnr.it

**Abstract.** Security for mobile devices is a problem of capital importance, especially due to new threats coming from malicious applications. Though several security solutions have already been proposed, security requirements have been always considered as binary: allow or deny. We argue that a more realistic vision of security can be given using probabilistic and quantitative requirements.
In this paper, we introduce a probabilistic description of the behavior of an application that a user is going to execute. We also allow the definition of finer grained user security requirements, by introducing probabilistic clause modifiers. Later, we present a probabilistic version of the Security-by-Contract framework to guarantee probabilistic security requirements.

**Keywords:** Probabilistic Contract, Probabilistic Policy Compliance, Contract-based Security approaches, Run-Time Enforcement.

## 1 Overview

New generation mobile devices (*e.g.*, smartphones and tablets) are becoming day-by-day more powerful and popular. The growth in computing power, ubiquitousness and capabilities of these devices has been paralleled by the growth of available applications, specifically developed for smartphones and tablets. However, these applications may be not completely secure. In fact, malicious developers strive to design and deliver applications that may damage both users and devices. In particular some applications may hide a Trojan horse that, even if it looks unharmful, in background it performs malicious actions that the users did not expect to happen.

The current security model, which rules (i) if an application can be safely installed on the device, (ii) what kind of actions the application may execute once installed, still suffers from several weaknesses, in particular in its capacity of expressing proper *contracts*. Semantics of current security models is too naïve since it is either based upon trust relationships or upon statements of purpose. In the first case, users accept to run an application if they trust the provider. In the second one, providers state the security

relevant actions performed by an application and it is up to the users to decide whether run the application if they consider these operations safe. In the former case the trust level of the trusted entity also determines the code privileges, essentially relegating an application into the "all or nothing" policy, while in the latter case the semantics is too-coarse grained (*e.g.*, Android permissions) or hardly usable. For example, in the Android system, security relevant actions are declared through permissions, which are difficult to understand for average users.

In this paper, we introduce probability aspects into the workflow of a contract-based approach developed for mobile devices: the Security-by Contract [1] (S×C) framework. This approach integrates several security techniques to build a chain of trust, which, in the end, ensures that the downloaded application will execute only security actions that are allowed by the user's policy. To this end, we introduce a probabilistic description of the behavior of an application and a more expressive version of the user's security requirements. Indeed, the current models only permit the definition of a set of allowed actions, *e.g.*, the Android permission system (first box of Figure 1). More expressive policies which take in account a possible action history are modelled through automata that represent allowed executions.

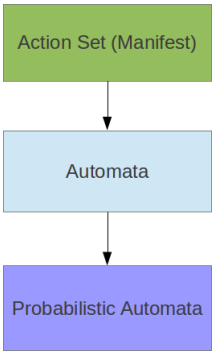


**Fig. 1.** Graphical representation of the improvement in policies expressiveness.

We propose a probabilistic automata-based model that enables the developers to define more expressive contracts through probabilistic clauses, *e.g.*, how often a security-relevant action may happen. The same expressiveness is given to users to specify security policies. Since we include probabilistic clauses in the specification of contracts and policies, the security mechanisms involved into the workflows of S×C has to be redefined. Hence, we present a new workflow for the S×C framework in which each module is updated to support probabilistic functions. The advantage of using probabilities is the possibility of describing more realistic usage scenarios for an application. In fact, many applications depend on user inputs or context information and it is difficult to define realistic policies based upon boolean conditions only. In these models, all the possible execution paths are considered legal. Hence, a low-probability operation is considered valid even if performed several times. For this reason, we introduce probabilities in the

definition of security clauses to define more fine-grained contracts and policies. These descriptions better fit real application use cases and can be defined without alteration of the Security-By-Contract-with-Trust workflow.

*The paper is structured as follows.* Section 2 introduces the main concepts of contract-based approaches and briefly recalls the Security-by-Contract framework. In Section 3, we propose a probabilistic version of Security-by-Contract. Section 4 briefly concludes proposing a future working line.

## 2 Contract-based Approaches

Contract-based approaches have been developed for mobile devices, such as the Security-by-Contract [1] (S×C) and the Security-by-Contract-with-Trust [2,3] (S×C×T) frameworks. They integrate several security techniques to build a chain of trust by sequentially applying them to safely execute applications. The three cornerstones of these security frameworks are application *code* $A$, application *contract* $C$, and client *policy* $P$, where a *contract* is a formal, complete, and correct specification of an application security relevant behavior, *e.g.*, security critical virtual machine API call, or critical system calls [4]. A *policy* is a formal complete specification of the acceptable security-relevant behavior allowed to applications executed on the platform [4]. We assume that both contract and policy are syntactically described by exploiting the same language.

The basic idea of a contract-based approach is the usage of the contract for guaranteeing that security aspects are satisfied. More in detail, using the contract, it is possible to check at deploy time, *i.e.*, before the application execution, if the application satisfies the user policy or not. Let $\preceq$ denote the compliance between two of the previous elements. A contract-based approach guarantees that

$$A \preceq C \preceq P \Rightarrow A \preceq P \tag{1}$$

In the following, we describe the Security-by-Contract (S×C) framework as approach that integrates the described techniques to guarantee security at application execution time.

### 2.1 Security-by-Contract

The Security-by-Contract paradigm provides a full characterization of the contract-based interaction. It combines different functionalities in an integrated way (see Figure 2). In particular, it includes a module for automatically checking the formal correspondence between code and contract (*Application-Contract matching*). If the result is negative, then the monitor is run to enforce the policy (*Policy Enforcement*), otherwise a matching between the contract and the policy (*Contract-Policy Matching*) is performed to establish if the contract is compliant with the policy. In this case, the code is executed without overhead (*Safe Execution*), otherwise the policy is enforced again (*Policy Enforcement*).

The advantages of contract-based frameworks are that they are able to identify unsafe applications before and without running them. In particular, using the contract-policy matching functionality, it checks at deploy-time if the declared behavior of the

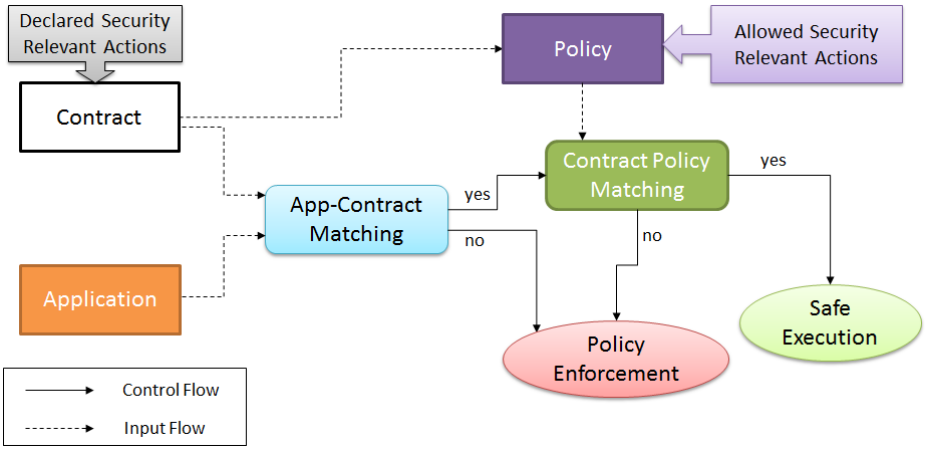


**Fig. 2.** The Security-by-Contract process.

application is compliant with the required policy. This check, along with the assurance that the application code is compliant with the application contract, which is obtained through the application-contract matching module, guarantees that the application satisfies the user requirements. Anytime the contract-policy matching finds that a contract is not compliant with the policy, the application is run in a controlled way through the enforcement module. It is worth noticing that the cost, in terms of energy, of running a contract policy matching is much lower than performing the enforcement. Hence, unsafe applications are not run at all by the user and possible unsafe application are run in a controlled way. This leads to an attack risk reduction.

## 3 Probabilistic Security-by-Contract

In this section, we describe a probabilistic version of the S×C architecture. It is worth noticing that, in both cases the original workflow is not changed. Only the components are modified in such a way that, on one hand, they are able to cope with probability metrics and, on the other hand, Equation 1 still holds for an appropriate choice of the notion of compliance.

Let us assume that both probabilistic contract and probabilistic policy are expressed through the same formalism.

Probabilistic contract and policy will be modelled as *(substochastic) generative probabilistic automata* [5,6].

**Definition 1.** *A* fully probabilistic *or* generative *automata is a tuple* $(S, Act, P)$ *consisting of a finite set* $S$ *of states, a set of actions* $Act$*, and a* transition probability function

$$P : S \times Act \times S \to [0, 1]$$

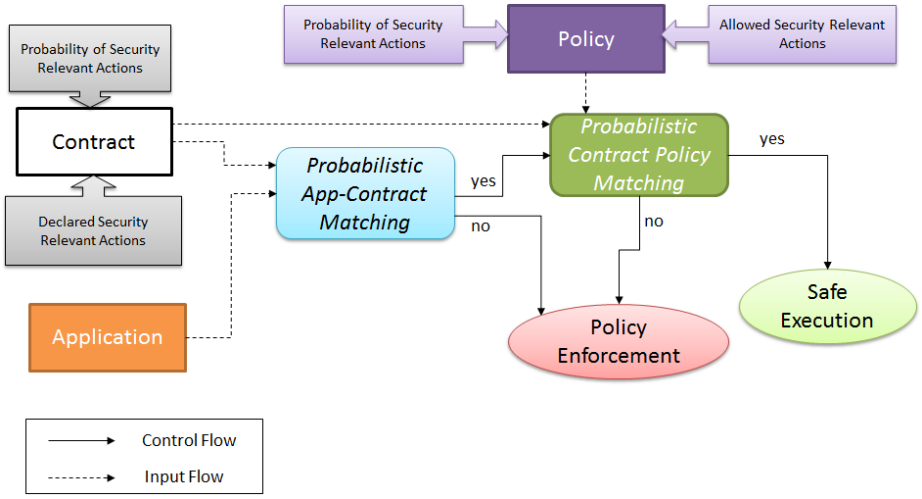


**Fig. 3.** Workflow for Probabilistic Security-by-Contract

*A generative automata is said to be* stochastic *if*

$$\sum_{a \in Act} \sum_{t \in S} P(s,a,t) = 1$$

*for all* $s \in S$ *for all* $a \in Act$*. On the other hand, a generative automata is said to be* semistochastic *or* substochastic *if*

$$\sum_{a \in Act} \sum_{t \in S} P(s,a,t) < 1$$

*for all* $s \in S$ *for all* $a \in Act$*. For* $C \subseteq S$*, we put* $P(s,a,C) = \sum_{t \in C} P(s,a,t)$*. A state* $s \in S$ *is said to be* terminal *iff* $\sum_{a,t} P(s,a,t) = 0$*.*

Hereafter, we consider generative automata such that for each action there is only one possible transition for each action $a \in Act$.

### 3.1 Probabilistic Security-by-Contract Workflow

Being the Security-By-Contract framework modular, introducing probability metrics implies the substitution of some components with their probabilistic counterpart. The Probabilistic Security-by-Contract workflow is depicted in Figure 3.

**Probabilistic application contract matching** is verified using some static validation techniques able to deal with probabilistic description of behavior. For instance, as *proof carrying code* [7] is used in S×C, here we can use the *Probabilistic Proof Carrying Code*, *e.g.*, [8,9]. In particular, this method guarantees that, for all possible $k$-length execution traces whose probability is calculated as $P_k$ =

$\prod_{i=1}^{k} P(s_i, a_i, t_i)$, the application is considered compliant if $P_k > \theta_k$, where $\theta_k$ is a given threshold value $0 < \theta_k < 1$ dependent from the length of the execution trace.

**Probabilistic contract policy matching** is performed by checking the compliance between a contract and a policy. According to the level of required accuracy, several relations can be considered in order to verify the compliance between probabilistic contract and policy. In S×C, the contract-policy matching function checks if the contract and the policy are similar. This means that for each action described in the contract, we check if there exists the same action described in the policy and the description of the transition are similar again. Hence, we assume that the policy specifies a rule for each security relevant action, which we call *SecAction*.
Referring to the notion of $\varepsilon$-simulation given in [10], hereafter, we define a slightly different $\varepsilon$-simulation.

**Definition 2.** *A relation $R \subseteq S \times S$ is a relation of* positive $\varepsilon$-simulation*, where $\varepsilon \in [0, 1]$ if whenever $(s, s') \in R$, then $\forall a \in SecAction$, $\forall W \in S$*

$$\sum_{t \in W} P(s, a, t) \leq \sum_{t' \in R(W)} P(s', a, t') \leq \sum_{t \in W} (P(s, a, t) + \varepsilon)$$

*where $R(W)$ is the set of all states that are in relation with states in $W$ trough $R$. We say that $s$ is $\varepsilon$-simulated by $s'$, written $s \prec_\varepsilon s'$, if $(s, s') \in R$ for some relation of $\varepsilon$-simulation $R$ on $S$.*

The idea is that, while the $\varepsilon$-simulation allows a deviation of a values $\varepsilon \in [-1, 1]$, here, we are only interested in positive values of $\varepsilon$. Hence, the probabilistic distribution of the contract have to be less that the probability distribution of the policy of, at most, a value $\varepsilon$.
It is worth noticing that, according to our assumptions, having a positive $\varepsilon$-simulation $R$ means that whether $(s, s') \in R$ then, for each action $a \in SecAction$,

$$P(s, a, t) \leq P(s', a, t') \leq P(s, a, t) + \varepsilon$$

and $(t, t') \in R$.

**Enforcement of Probabilistic Policies** is performed when either the application is not compliant with the contract or the contract is not compliant with the policy.
At each step, the enforcement computes the probability that the application performs a specific security relevant action $a$, starting from the current state $s$, $P^p(s, a, t)$, where $t$ is the destination state of the transition and $p$ is the expected one stated by the policy. This computation exploits history-based concerning the current execution of the application. The computation of the probability of the execution trace is similar to the one described in the application-contract matching module $P_k^p = \prod_{i=1}^{k} P^p(s_i, a_i, t_i)$. The application is considered compliant if $P_k^p > \theta_k$, where $\theta_k$ is the same considered in the application-contract module. The enforcement denies the non compliant operation sequence, ensuring that the policy is correctly enforced.

It is worth noticing that Equation 1 holds. Indeed, the fact that $C \preceq_\varepsilon P$ means that $P_k \leq P_k^p$ because

$$P_k = \prod_{i=1}^{k} P(s_i, a_i, t_i) \leq \prod_{i=1}^{k} P^p(s_i, a_i, t_i) = P_k^p$$

Hence, $\theta_k < P_k \leq P_k^p$ Let $\preceq_\Theta$ be the compliance relation used in both application-contract matching and enforcement mechanisms, where $\Theta$ denotes the set of threshold values $\theta_k$ for any $k$-length execution trace, and let us consider to use the positive $\varepsilon$-simulation for the contract-policy matching then the following holds

$$A \preceq_\Theta C \preceq_\varepsilon P \Rightarrow A \preceq_\Theta P$$

## 4 Conclusion and Future Work

In this paper, we have discussed the current limitations of the semantics of the security models for mobile applications. To this end, we have presented a probabilistic version of the Security-by-Contract, which is able to guarantee probabilistic requirements. We have discussed the advantages in terms of expressiveness achieved including probability in the S×C framework. Future extensions to this work will be the definition of probabilistic formalisms and languages, which should be used to programmatically define probabilistic contracts and policies, then to verify their compliance. This languages should be equivalent in expressiveness to the probabilistic automata that we have used to express policies and contracts. Furthermore, we are going to include the presented framework in real mobile devices, investigating if it is possible to distribute it as common mobile application, which can give users a way to better control their mobile devices.

## References


1. Dragoni, N., Martinelli, F., Massacci, F., Mori, P., Schaefer, C., Walter, T., Vetillard, E.: 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)
2. Costa, G., Dragoni, N., Lazouski, A., Martinelli, F., Massacci, F., Matteucci, I.: Extending Security-by-Contract with quantitative trust on mobile devices. In: Proceeding of the Fourth International Conference on Complex, Intelligent and Software Intensive Systems, IEEE Computer Society (2010) 872–877
3. Costa, G., Dragoni, N., Issarny, V., Lazouski, A., Martinelli, F., Massacci, F., Matteucci, I., Saadi, R.: Security-by-Contract-with-Trust for mobile devices. JOWUA **1**(4) (2010) 75–91
4. Greci, P., Martinelli, F., Matteucci, I.: A framework for contract-policy matching based on symbolic simulations for securing mobile device application. In: ISoLA. (2008) 221–236
5. Hermanns, H., Parma, A., Segala, R., Wachter, B., Zhang, L.: Probabilistic logical characterization. Inf. Comput. **209**(2) (2011) 154–172
6. Baier, C., Engelen, B., Majster-Cederbaum, M.: Deciding bisimilarity and similarity for probabilistic processes. Journal of Computer and System Sciences **60**(1) (2000) 187–231

7. Necula, G.C.: Proof-carrying code. In: Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Langauges (POPL '97). (1997) 106–119
8. Sharkey, M.I.: Probabilistic Proof-carrying Code. PhD thesis, Carleton University (2012)
9. Tsukada, Y.: Interactive and probabilistic proof of mobile code safety. Automated Software Engineering **12**(2) (2005) 237–257
10. Desharnais, J., Laviolette, F., Tracol, M.: Approximate analysis of probabilistic processes: Logic, simulation and games. In: Proceedings of the 2008 Fifth International Conference on Quantitative Evaluation of Systems. QEST '08, Washington, DC, USA, IEEE Computer Society (2008) 264–273