Trust-based Enforcement of Security Policies

Roberto Vigo[1], Alessandro Celestini[2], Francesco Tiezzi[3], Rocco De Nicola[3], Flemming Nielson[1], and Hanne Riis Nielson[1]

[1] DTU Compute, Technical University of Denmark, Denmark
[2] Istituto per le Applicazioni del Calcolo, IAC-CNR, Rome, Italy
[3] IMT, Institute for Advanced Studies Lucca, Italy
rvig@dtu.dk, a.celestini@iac.cnr.it, francesco.tiezzi@imtlucca.it, rocco.denicola@imtlucca.it, fnie@dtu.dk, hrni@dtu.dk

Abstract. Two conflicting high-level goals govern the enforcement of security policies, abridged in the phrase “high security at a low cost”. While these drivers seem irreconcilable, formal modelling languages and automated verification techniques can facilitate the task of finding the right balance. We propose a modelling language and a framework in which security checks can be relaxed or strengthened to save resources or increase protection, on the basis of trust relationships among communicating parties. Such relationships are automatically derived through a reputation system, hence adapt dynamically to the observed behaviour of the parties and are not fixed a priori. In order to evaluate the impact of the approach, we encode our modelling language in StoKlaim, which enables verification via the dedicated statistical model checker SAM. The overall approach is applied to a fragment of a Wireless Sensor Network, where there is a clear tension between devices with limited resources and the cost for securing the communication.

Keywords: Security policies, probabilistic aspects, reputation systems, stochastic verification.

1 Introduction

Security policies are usually formalised as fixed rules concerning actions that have to be executed whenever given conditions are met. Nonetheless, the scenarios in which security policies operate are often open and highly dynamic, and decisions may depend on external factors, not entirely known at design-time: this is for example the case when human agents play a role in the system, or when systems components have limited resources that need be used carefully.

The challenge of modelling security policies in an uncertain world has been first addressed by AspectKP [1], where policies are implemented as probabilistic aspects: when the triggering conditions are met, an aspect (i.e., a policy) is enforced with a given probability. This approach is suitable for taking into account a great many real concerns, including the possibility that some undertaken security measures are eluded.

In a more complex scenario, the probabilistic enforcement of security policies can account for relaxing or strengthening a security check, aiming at saving resources or increasing protection. In order to react to a dynamic environment, the probability to undertake, relax, or strengthen a given security measure cannot be fixed a priori, as in [1], but depends on our knowledge of the past and on our expectations for the future. In a distributed system, knowledge and expectation can be interpreted as the opinion we have formed about the parties we are interacting with, hence they can be connected to the *trust* we put on our peers.

Probabilistic reputation systems [2, 3] offer a framework for quantifying trust in probabilistic terms. In a reputation system, communicating parties rate each other when an interaction is completed, and then use such rates to compute a reputation score, which is levered to decide about future interactions. The main advantage of reputation systems is that they permit computing the probability that a policy is enforced as the system evolves and according to its observed behaviour. This leads to the notion of *trust-based enforcement of security policies*.

The trust we put on the involved parties at a given time can be used to determine the actual enforcement of security polices. A policy can be relaxed with the aim of saving resources: in case enforcing the prescribed checks is expensive (in terms of time, energy, etc.), they can be skipped or relaxed when interacting with parties who exhibited fair behaviour in the past. On the contrary, we can strengthen a security check when interacting with a party we do not trust, or dually we can exchange critical information only with highly-trusted parties. In this view, the enforcement or relaxation of security policies becomes (probabilistically) optional with respect to a default mode, which is to be followed when we do not have enough information to take meaningful decisions.

In order to tackle the challenge of modelling and reasoning about systems where security policies are enforced on a trust basis, we propose TESP (Trust-based Enforcement of Security Policies), a calculus inspired by [1], that enriches the StoKlaim [4] formalism with probabilistic aspects for security policies and with primitives for managing reputation scores. The transformational semantics of TESP associates to each TESP term a StoKlaim one, thus enabling the verification of properties specified in MoSL (Mobile Stochastic Logic) [5] by means of the model checker SAM [6]. The main technical challenge addressed in this work is the integration of all these elements (in particular, probabilistic aspects, security policies, and reputations) in a uniform framework. The key role in this integration is played by the policy evaluation described in Sect. 2.3.

The choice of a stochastic process calculus seems natural once one considers that trust relationships vary over time. Moreover, also resource usage is conditioned temporally: batteries deplete and may recharge, and for instance we might wonder whether the probability of disclosing some information to a third party within the average life-span of a component meets a given threshold.

The flexibility of the framework is demonstrated on the example of a Wireless Sensor Network, where the need for securing the communication in an open environment coexists with "lazy" components that have limited resources and thus want to perform as few operations as possible.

Related work. Aspects Oriented Programming (AOP) [7] focuses on separation of concerns in developing software systems, and has been extensively studied in connection with the enforcement of security policies [8–10]. TESP extends and combines two lines of research, putting reputation systems to use in the development of security policies through aspects. On the one hand, the integration of aspects into a coordination language in the KLAIM family [11] has been explored in [10]. On the other hand, STOKLAIM has been used in [12] to model and verify properties of reputation systems, aiming at providing a formal understanding of such objects. The statistical model checker [13] SAM is presented in [14].

The relationship of TESP with respect to its inspiring languages, namely AspectKP and STOKLAIM, can be summarized as follows. On the one hand, our language is essentially a superset of AspectKP, and can certainly model all the systems expressible with AspectKP. In fact, one of our aims is to loosen the rigidity of the latter, where policies are applied with fixed probabilities. On the other hand, we have that our language is fully encodable in STOKLAIM, as argued throughout Sect. 2, and we can thus exploit the statistical model checker SAM, that works on STOKLAIM models, to check properties expressed in MOSL.

Statistical inference offers a formal framework for analysing historical observation and synthesising probability distributions. Whilst probabilistic reputation systems surely relies on the applications of statistical methods, they have the specific merit of defining a framework where trust scores are adjusted dynamically, as the system evolves. We consider here reputation systems based on probabilistic trust [15], whose basic postulate is that the behaviour of each player in the system can be modelled as a probability distribution over a given set of interaction *outcomes*. Once this postulate is accepted, the task of computing reputation scores reduces to inferring the *true* distribution parameters for a given player. A study of reputation systems in the realm on WSNs is presented in [16].

In this work we stick to basic mechanisms for managing policies, focusing instead on their trust-based enforcement. For more elaborate approaches to policy selection and combination the reader may refer to XACML [17].

Outline. We present TESP syntax in Sect. 2, showing how to encode the new constructs in STOKLAIM. The developments of the language is demonstrated on a running example, showing how reputation management and policy enforcement seamlessly integrate in the basic calculus. Some quantitative properties of interest are verified by means of statistical model checking and simulation in Sect. 3. Finally, Sect. 4 concludes and sketches a line for future work.

2 TESP: Syntax and Informal Semantics

TESP (Trust-based Enforcement of Security Policies) is a distributed process calculus which can be used to model security policies and their probabilistic enforcement, relying on a reputation system to infer probabilities. The calculus, displayed in Table 1, enriches (a subset of) STOKLAIM with primitives for handling reputations and inherits the aspect-oriented mind-set of AspectKP for the design of security policies.

A system is rendered as a network whose nodes represent the communicating parties (referred to also as *players* in the following). As customary in KLAIM [18], each node is equipped with a tuplespace, modelling a local memory. Communication between nodes is asynchronous and point-to-point: sending is modelled as storing a tuple in the receiver's locality space, whereas receiving is modelled as reading a tuple from the local tuplespace. However, other forms of interaction typical of tuplespace-based communication are allowed: e.g., receiving can be rendered as reading a tuple from the sender's locality. With respect to KLAIM-like calculi we refrain from allowing process mobility. We deem that this simplification facilitates focusing on the novelty of our work, and that the technical developments needed to encompass the full-fledged version of STOKLAIM would provide little additional insight into our approach.

In the following we present the syntax and the intended semantics of the calculus. The formal semantics of STOKLAIM is introduced in [4]; we provide the semantics of the new constructs by means of a translation into STOKLAIM processes.

2.1 Basic calculus

The top-level entities of the calculus are networks. A network is a collection of nodes $l ::^{\Pi} (P, \mathcal{T})$ combined with the parallel operator $||$. Each node represents a player, which is uniquely identified by its locality l, and is characterised by a list of policies Π, a running process P, and a tuplespace $\mathcal{T}$.

Given a process P, the special locality self denotes the locality where P is executing. A process is either the parallel composition of processes (i.e., threads), the guarded sum of action-prefixed processes, or the invocation of a process identifier. We assume that each process identifier A has a unique definition of the form $A \triangleq P$, visible from any locality of a network. The terminated process **nil** is obtained as the empty sum. For the sake of writing more readable processes, we introduce the conditional statement $\mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$, which is is not pure STOKLAIM syntax but can be easily encoded in the language and is available in the model checker SAM together with Boolean expressions e, that can check locality equality or compare integers.

An action has the form $a@u : \lambda$, denoting that action a has target locality u and a stochastic duration governed by a negative exponential distribution with rate λ. We write $l :: a@u : \lambda$ to stress that a is attempted from locality l.

An action can be an output to or an input from a tuplespace. Tuples t are communicable objects consisting of sequences of data elements u. Such elements can be constant locality values l[4] and applied occurrences of variables v. Templates T consist of sequences of data element u and defining occurrences of variables $v?$. Prefixes $\mathsf{in}(\ldots, v?, \ldots)@u : \lambda.P$ and $\mathsf{read}(\ldots, v?, \ldots)@u : \lambda.P$ bind

[4] For the sake of simplicity we consider here only localities as communicable values. Observe however that Boolean, integer, and string values can be encoded with ad hoc localities.

Table 1. Syntax: Networks, Processes and Actions.

$N ::= N_1 \vert\vert N_2 \quad \vert \quad l ::^{\Pi} (P, \mathcal{T})$	(Network)
$P ::= P_1 \vert P_2 \quad \vert \quad \sum_i a_i @ u : \lambda_i . P_i \quad \vert \quad A \quad \vert \quad \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$	(Process)
$a ::= \mathsf{out}(t) \quad \vert \quad \mathsf{in}(T) \quad \vert \quad \mathsf{inp}(T) \quad \vert \quad \mathsf{read}(T) \quad \vert \quad \mathsf{readp}(T) \quad \vert \quad \mathsf{rate}(\mathit{res})$	(Action)
$\mathit{res} ::= \mathsf{true} \quad \vert \quad \mathsf{false}$	(Interaction result)
$\mathcal{T} ::= \{\{\}\} \quad \vert \quad \mathcal{T} \uplus \langle t \rangle$	(Tuplespace)
$u ::= l \quad \vert \quad v \qquad t ::= u \quad \vert \quad t_1, t_2 \qquad T ::= u \quad \vert \quad v? \quad \vert \quad T_1, T_2$	(Data)

variable v in P. Operator $\uplus$ increments tuplespaces, modelled as multi-sets of tuples.

As for communication, input works by pattern-matching on tuples: the process $\mathsf{in}(T)@u:\lambda.P$ looks for a tuple t matching the template T in the tuplespace located at u, and whenever such a t is found it is removed from u and the process evolves to $P\sigma$, where σ is a substitution containing the bindings of variables defined in T and instantiated as in t. If t is not found, then $\mathsf{in}(T)@u:\lambda.P$ is blocked until a matching tuple is available. The input action $\mathsf{read}(T)@u:\lambda$ behaves like $\mathsf{in}(T)@u:\lambda$, except that t is not removed from u. A process $\mathsf{out}(t)@u:\lambda.P$ puts tuple t in the tuplespace at locality u and proceeds as P.

For convenience, we inherit from Linda and some versions of Klaim the distinction between blocking and non-blocking input actions, the latter being denoted $\mathsf{inp}(T)@u:\lambda$ and $\mathsf{readp}(T)@u:\lambda$, for destructive and non-destructive input, respectively. These are *predicate* operations that returns true if a matching tuple is found, false otherwise. As $\mathsf{inp}, \mathsf{readp}$ always allow progressing with the computation, some defining occurrence of variable might have not received a value after such an operation is consumed. There are different approaches to handle a non-bound variable in a continuation process. In Linda, where non-blocking input operations have been first introduced, the intention was to check the boolean result of the operation, so as to split the continuation process in two branches, and use such variables only in the branch related to a successful input [19, pg. 2-25]. However, no enforcement mechanism is provided which guarantees that a variable is accessed only if it carries some information. A more elegant approach has been recently proposed in [20], where the notion of option data type is exploited to differentiate an input variable possibly carrying no value from one actually bound to some value. Whilst this would be an interesting development, it is independent from the use of reputation systems, and thus in the following we resort to Linda style, to avoid overly complicating the syntax. Finally, a third possibility consists in restricting the syntax of $\mathsf{inp}, \mathsf{readp}$ not to range on binding occurrences of variables, hence writing $\mathsf{inp}(t)@u:\lambda$.

As usual, in the following we consider closed processes (no free variables).

Running example 1/3. Consider a hierarchical Wireless Sensor Network in which a number of *sensors* monitor environmental parameters (e.g., temperature, pressure, etc.) and communicate them to *base stations*, in charge of validating the data and forwarding them to a *central server*. Since sensors and stations are generally powered by batteries and have limited computational capabilities, communication between them is not encrypted but relies on signatures appended to messages, so as to enable stations to verify the integrity of received sensor readings. Due to cheap hardware, unreliable wireless communication, and active attackers it may indeed be the case that the message received by a station gets corrupted.

A possibly faulty sensor can be modelled by the following process:

$$\mathsf{sensor}_i \triangleq \mathsf{sensor}_i^{sound} \mid \mathsf{sensor}_i^{faulty}$$

$$\begin{aligned}\mathsf{sensor}_i^{sound} \triangleq\ & \mathsf{in}(\text{“token”})@\mathsf{self}:\lambda_{sign}.\,\mathsf{out}(\text{“reading”}, m, \mathsf{self}, \mathsf{self})@\mathsf{station}_j:\lambda_1.\\ & \mathsf{out}(\text{“token”})@\mathsf{self}:\lambda_2.\,\mathsf{sensor}_i^{sound}\end{aligned}$$

$$\begin{aligned}\mathsf{sensor}_i^{faulty} \triangleq\ & \mathsf{in}(\text{“token”})@\mathsf{self}:\lambda_{corrupt}.\,\mathsf{out}(\text{“reading”}, m, n, \mathsf{self})@\mathsf{station}_j:\lambda_1.\\ & \mathsf{out}(\text{“token”})@\mathsf{self}:\lambda_2.\,\mathsf{sensor}_i^{faulty}\end{aligned}$$

where the first parallel component ($\mathsf{sensor}_i^{sound}$) models the case in which the sensor issues a reading with the correct message signature, and the second component ($\mathsf{sensor}_i^{faulty}$) describes the case in which the message is corrupted. The choice is modelled by two concurrent withdrawals (in) of a local ($@\mathsf{self}$) shared tuple (“token”), whose rates (λ_{sign} and $\lambda_{corrupt}$) determine the frequency with which a message is delivered intact or corrupted, respectively. Sending a sensor reading to the parent $\mathsf{station}_j$ corresponds to storing (out) a tuple in its tuplespace ($@\mathsf{station}_j$), containing the string “reading” as first field, the content of the reading m (assumed to be a fresh constant), the signature, and the source locality. For the sake of simplicity, a signed message is ideally described by a pair (m, l), where m is the message payload and l the signature (second and third fields, respectively), represented as the locality that generated the message[5]. If the delivered message is not corrupted, at $\mathsf{station}_j$ the third and fourth fields of the tuple will be identical (declared and actual source coincide), otherwise they will not (n is a fresh constant). Once the sensor reading is sent, the component releases the token ($\mathsf{out}(\text{“token”})@\mathsf{self}$) and restarts.

A base station is then in charge of receiving readings and forwarding them to the central server. A sensor reading is forwarded only after the signature check is successfully passed. A station can modelled by the following process:

$$\begin{aligned}\mathsf{station}_j \triangleq\ & \mathsf{in}(\text{“reading”}, v_m?, v_s?, v_{source}?)@\mathsf{self}:\lambda_3.\\ & \mathsf{out}(\text{“check\_req”}, v_m, v_s, v_{source})@\mathsf{self}:\lambda_4.\\ & \mathsf{in}(\text{“check\_res”}, v_{res}?)@\mathsf{self}:\lambda_5.\\ & \mathsf{if}\ (v_{res})\ \mathsf{then}\ \mathsf{out}(\text{“reading”}, v_m)@\mathsf{server}:\lambda_7.\,\mathsf{station}_j\ \mathsf{else}\ \mathsf{station}_j\end{aligned}$$

The retrieval of a reading from the local tuplespace binds variables v_m, v_s, v_{source} to the reading measure, the corresponding signature, and the sensor's locality,

[5] Modelling cryptographic primitives and keys falls outside the scope of this work. The presence of the sender locality in the signed message accounts for the usage of a private key in a more precise encoding.

respectively. The signature check is then performed by a check service local to each base station, rendered as the following parallel process:

$$\begin{aligned}\mathsf{check}_j \triangleq\ & \mathsf{in}(\text{"check\_req"}, v_m?, v_s?, v_{source}?)@\mathsf{self}\!:\!\lambda_8.\\ & \mathsf{if}\ (v_s = v_{source})\ \mathsf{then}\ \mathsf{out}(\text{"check\_res"}, \mathsf{true})@\mathsf{self}\!:\!\lambda_9.\,\mathsf{check}_j\\ & \qquad\qquad\qquad\ \mathsf{else}\ \mathsf{out}(\text{"check\_res"}, \mathsf{false})@\mathsf{self}\!:\!\lambda_9.\,\mathsf{check}_j\end{aligned}$$

This process consumes a check request, compares[6] the signature with the source locality and sends a Boolean value representing the result of the comparison to the requester. In case the check fails, i.e., the message is corrupted, the base station discards it, otherwise it is forwarded to the server.

2.2 Reputations

The novelty of TESP consists in coupling networks with reputation systems; specifically, we focus on probabilistic reputation systems [2], where players' behaviour is modelled as a probability distribution over a set of interaction outcomes.

We assume that each locality l (i.e., player) has an associated reputation value $rep(l) \in [0, 1]$, representing the trust that the system puts on l, namely 0 denoting a completely distrusted player and 1 a completely trusted one. A new action is introduced to deal with reputations: $\mathsf{rate}(res)@u\!:\!\lambda$ updates $rep(u)$ according to the outcome res of an interaction involving the process located at u. We shall see in Sect. 2.3 how reputation scores are levered to enforce policies probabilistically.

In the StoKlaim implementation, the reputation score of a player is modelled as a set of tuples stored in some tuplespace. Accordingly, $\mathsf{rate}(res)@u$ would take the form of one or more output actions, whose values and target tuplespaces are defined by the reputation model.

For the sake of simplicity, in the grammar of Table 1 the outcome of an interaction between processes is a Boolean value, ideally denoting whether or not the interaction took place as expected. We refrain from discussing more complex choices (e.g., using Boolean expressions as argument of rate), as they would provide no additional insight into our approach. The specification of a system, finally, requires defining the initial reputation of each locality. This is an application-dependent task: in our running example we make a conservative choice and start with $rep(l) = 0$ for all localities.

Running example 2/3. As a base station is connected to a great many sensors, it is the bottle-neck on the way to the server in charge of transforming sensed data into information. In order to save time and energy, the behaviour of stations can be revised so that they check signatures probabilistically, on a trust basis. In fact, together with communication, checking a signature is the most demanding

[6] In a real-world application this consists in hashing the message plain-text, decrypting the signature with the source's public key, and comparing the two bit-strings.

operation performed by stations. The first step consists in coupling processes with a reputation system, where sensors are rated by stations according to their behaviour. In the base station process the last line is replaced by:

$$\begin{array}{ll} \mathsf{if}\ (v_{res}) & \mathsf{then}\ \mathsf{rate}(\mathsf{true})@v_{source}:\lambda_6.\,\mathsf{out}(\text{“reading”}, v_m)@\mathsf{server}:\lambda_7.\,\mathsf{station}_j \\ & \mathsf{else}\ \mathsf{rate}(\mathsf{false})@v_{source}:\lambda_6.\,\mathsf{station}_j \end{array}$$

The base station exploits the result of the signature check to rate sensors: in case the check succeeds, the sensor receives a positive score ($\mathsf{rate}(\mathsf{true})@v_{source}$), otherwise if the message is corrupted the sensor receives a negative score.

As for the reputation system, in the implementation of the example we chose the ML model [3], where the reputation score of a sensor is given by the number of positive interactions (i.e., successful signature checks) over the total number of interactions. rate is thus implemented as an output performed by the base station to its own locality, recording for each sensor the number of positive and negative interactions. In case more base stations were considered, each one would have its own opinion about the reputation of connected sensors, even if a sensor were communicating with more than one station, hence obtaining a local view of reputations. A global perspective would be implemented with a unique tuplespace (e.g., at the server) where all the reputation scores were stored and retrieved from.

2.3 Policies

We assume that each locality l is annotated with a list of policies $\pi_1.\pi_2.\pi_3\ldots$, denoted $\Pi(l)$. The syntax of policies is displayed in Table 2, and extends Table 1. Basic policies do and skip stand for policies allowing and denying everything, respectively, while more interesting policies are implemented with aspects.

A policy $[?_{rep(u)} rec\ \mathsf{if}\ cut : cond]$ is applied when an action $l :: a@u$ is executed which matches the aspect trigger *cut*. The selection mechanism scans policy lists from left to right and selects the first policy that applies, disregarding other relevant policies that may occur later in the list. First, the local policies $\Pi(l)$ are scanned, and only upon local approval by the action source l the policies at the target u are scanned. This approach is suitable to combine optimisation with security concerns: while the source l may deny the execution of an action to save resources or to ensure security, the target u implements access control on its local tuplespace and thus has the last word. Hence, either no policy is enforced (action denied at source), or one policy (action allowed at source and no policy at target), or two policies (both at source and target).

The matching between an action $l :: a@u$ and an aspect trigger *cut* works via pattern-matching and produces a substitution from variables that occur free in *cut* to terms in corresponding positions in a. We shall feel free to use the wild-card symbol _ (obtained by matching a variable never used in the policy.)

Assume that the execution of a process located at l reaches a point $a@u:\lambda.Q$, and that $[?_{rep(u')} rec\ \mathsf{if}\ cut : cond]$ is the policy to be enforced. If the predicate

Table 2. Syntax: Aspects and Policies.

$\pi ::= \mathsf{do} \mid \mathsf{skip} \mid [?_{rep(u)} rec\ \mathsf{if}\ cut : cond] \mid \pi_1.\pi_2$	(Policy)
$cut ::= l :: a@u$	(Aspect trigger)
$cond ::= \mathsf{true} \mid \mathsf{false} \mid d_1 = d_2 \mid \mathsf{readp}(T)@u : \lambda$ $\mid \neg cond \mid cond_1 \wedge cond_2 \mid cond_1 \vee cond_2$	(Applicability condition)
$rec ::= \mathsf{skip} \mid \mathsf{do} \mid P$	(Recommendation)

specified by the condition *cond* evaluates to true then the probabilistic recommendation $?_{rep(u')} rec$ is evaluated. The condition *cond* can either test the presence of a tuple in a given tuplespace (and thus generates a substitutions that applies to the whole policy), check locality equality, or be the Boolean combination of simpler conditions.

A recommendation can prescribe to skip the triggering action $l :: a@u$, to execute $a@u$, or to replace $a@u$ *and its continuation* with a *closed* process P. Recommendations behave probabilistically, levering trust relationships. $?_{rep(u)} rec$ enforces *rec* with probability $rep(u)$, while with probability $1 - rep(u)$ the recommendation is not enforced and $a@u$ is executed according to the default plan. Hence, the enforcement of policies relies on reputation scores rather than fixed probabilities (that can however be introduced seamlessly).

Running example 3/3. We can now show how a base station exploits reputation scores to enforce checking signatures probabilistically: the higher reputation a sensor has, the less wary the station will be when interacting with it.

A base station is instrumented with a policy triggered by the output requesting to check a signature. The policy allows or denies such request according to the reputation of the message source, as prescribed by the probabilistic recommendation:

$$\mathsf{Pol}_{check} \triangleq \begin{bmatrix} ?_{rep(v_{source})}\ \mathsf{out}(\text{“reading”}, v_m)@\mathsf{server} : \lambda_7.\,\mathsf{station}_j \\ \mathsf{if}\ \mathsf{self} :: \mathsf{out}(\text{“check\_req”}, v_m, \_, v_{source})@\mathsf{self} : \mathsf{true} \end{bmatrix}$$

Whenever an output matching $\mathsf{out}(\text{“check\_req”}, v_m, \_, v_{source})$ is attempted by the base station, the probabilistic recommendation is evaluated (the condition being true). The higher the reputation of the source v_{source}, the higher the probability to enforce the recommendation, which simply replaces the output and its continuation with a new process that directly forwards the message to the server skipping the signature check and the emission of the rating, and then restarts the original process.

It is worth noticing that another way to obtain similar (or better) results in reducing the computational overhead of sensors is by redesigning and modifying the system using checks less expensive than those based on digital signatures (e.g., based on Message Authentication Codes). However, refactoring the system is much more costly than deploying new aspects, which are independent from the implementation once the binding mechanism is in place.

Semantics. Our policy selection mechanism leads to replace an action $l_i :: a@u : \lambda$ with a conditional structure testing whether a local policy applies to the action and, in case the action is granted, whether a remote policy applies to the action. In particular, the more interesting latter check translates as follows:

```
if($u = l_1$) then
   select the first $\pi$ in $\Pi(l_1)$ s.t. $\pi$ applies to $a$;
   evaluate($\pi$);
else if ($u = l_2$) then ...
```

where $l_1, \ldots, l_n$ are all the localities in the network, skip and do apply to every action, and an aspect applies if its *cut* matches a (yielding another conditional structure). Assuming that $a@u : \lambda$ triggered the policy and that Q is its continuation, the evaluation of π is defined as follows:

$$\mathsf{evaluate}(\pi) = \begin{cases} a@u : \lambda.Q & \mathsf{if}\ \pi = \mathsf{do} \\ Q & \mathsf{if}\ \pi = \mathsf{skip} \\ \ldots & \mathsf{if}\ \pi = [?_{rep(u)} rec\ \mathsf{if}\ cut : cond] \end{cases}$$

(the last case is detailed below). Observe that we can only achieve this behaviour in closed systems, where nodes know each other's policies. As the lists of policies are fixed, the translation can be automatically generated. The machinery necessary to remove this restriction would imply fairly elaborate technical developments that falls outside the scope of this work (basically, code mobility and reflection capabilities). Similarly, more complex look-up criteria can be considered, well-beyond the sequential priority we resort to (which is however motivated by implementations common in firewalls). Nonetheless, such elaborate mechanisms are independent from our usage of reputation, hence we point the curious reader to [1] for further details on policy combination in an aspect-oriented mind-set.

It is worthwhile noticing that skip allows ignoring an action and going on with the continuation process, whereas in [1] the failure of a policy leads to entering a busy waiting state where the action might be attempted at a later time. Obviously, due care has to be paid in the continuation process, as variables bound by an action that might be skipped cannot be used freely. This issue advocates for a programming style similar to the one prescribed with predicate input actions inp and readp. Other actions do not return a Boolean value denoting success or failure, but this behaviour can be encoded initialising variables with a default value not used elsewhere (e.g., $\perp$) and check whether or not they have been modified after the evaluation of the policy and before accessing them. This would result in replacing $a@u : \lambda.Q$ with $Q[\perp/v_1, \ldots, \perp/v_n]$ in the definition of evaluate for the case $\pi = \mathsf{skip}$, the v_i's being the variables bound by action a.

It remains to show how aspects are translated. Consider the policy $\pi \triangleq [?_{rep(u)} rec\ \mathsf{if}\ cut : cond]$, and assume that the matching between the action a and the trigger *cut* yields substitution θ_1, under which *cond* yields substitution θ_2. Then, $\mathsf{evaluate}(\pi)$ is defined as

```
if($\theta_1(cond)$) then
```

$\texttt{if(}[\![(\theta_2 \circ \theta_1)(?_{rep(u)}rec)]\!]\texttt{) then enforce(}rec,\ a@u : \lambda.Q\texttt{) else } Q$
$\texttt{else } a@u : \lambda.Q$

where $\theta_2 \circ \theta_1$ denotes the composition of substitutions, and its application to the probabilistic recommendation can only determine the location u whose reputation is being considered, as skip, do, P are closed.

The translation relies on the auxiliary functions $[\![\cdot]\!]$ and $\mathsf{enforce}$, for evaluating probabilistic recommendations and the outcome of their enforcement. The probabilistic enforcement of a recommendation translates to a Boolean guard:

$$[\![?_{rep(u)}\mathsf{skip}]\!] = ((rand(0,1) > rep(u)) \vee (rep(u) = 0)) = [\![?_{rep(u)}P]\!]$$
$$[\![?_{rep(u)}\mathsf{do}]\!] = ((rand(0,1) < rep(u)) \vee (rep(u) = 1))$$

where $rand(x, y)$ picks a number randomly[7] in the interval $[x, y]$. For instance, with $?_{rep(u)}\mathsf{skip}$ we obtain false, i.e., the action governed by the policy is skipped, with probability $rep(u)$: the higher the trust, the higher the chances to skip executing the action governed by the policy. Conversely, with $?_{rep(u)}\mathsf{do}$ we obtain true, i.e., the action governed by the policy is executed, with probability $rep(u)$: the higher the trust, the higher the chances to execute the action.

The result of enforcing a recommendation is defined as follows:

$$\mathsf{enforce}(\mathsf{skip}, a@u : \lambda.Q) = \mathsf{enforce}(\mathsf{do}, a@u : \lambda.Q) = a@u : \lambda.Q$$
$$\mathsf{enforce}(P, a@u : \lambda.Q) = P$$

Finally, observe that $?_{rep(u)}\mathsf{skip}$ and $?_{rep(u)}\mathsf{do}$ may be both used to relax or strengthen probabilistically a security measure, depending on the application. For example, by enforcing to encrypt a message by default sent as plain-text, or by skipping outputting it at all when possible, we are heightening the overall security of the system either way.

3 Analysing TESP

The stochastic nature of TESP specifications inherently calls for a quantitative verification approach. Quantitative techniques allow determining the probability that a given event will occur, and thus checking whether or not such value meets a threshold of interest.

In Sect. 2.3 we showed how a system in TESP can be translated into StoKlaim. Once a StoKlaim specification is obtained for the system under study, we can express the properties of interest in the temporal stochastic logic MoSL, and then verify whether or not a property holds by means of SAM, a statistical model checker that determines the probability associated to a path formula after a set of independent observations. The model checking algorithm is parametrised

[7] Function $rand(x, y)$ is implemented in StoKlaim as a selection among a number of tuples representing the interval values. In fact, the semantics of StoKlaim chooses with uniform probability among all matching tuples.

Table 3. Results of the analysis of the formula $true\ U^{\leq t} \phi_{FwdCorrupted}$.

Behaviour θ	Time t	Threshold th	Probabilities	Time t	Probabilities
0.2	20	1%	0.6534	40	0.8894
0.2	20	5%	0.4693	40	0.6087
0.2	20	10%	0.2378	40	0.2565
0.5	20	1%	0.8974	40	0.9869
0.5	20	5%	0.7256	40	0.8584
0.5	20	10%	0.3747	40	0.4010
0.8	20	1%	0.9279	40	0.9928
0.8	20	5%	0.5535	40	0.6151
0.8	20	10%	0.1379	40	0.1382

on a given tolerance threshold ϵ and error probability p, and guarantees that the difference between the computed values and the exact ones exceeds ϵ with a probability that is less than p.

In order to carry out a quantitative analysis on a StoKlaim network, we first have to specify the value of the rates characterising the actions. As an illustrative example, we show how the rate λ_1 of action $\mathsf{out}(\text{“reading”}, m, \mathsf{self}, \mathsf{self})@\mathsf{station}_j$ can be determined. The rate specifies the duration of a communicating action: assuming that the sensors are relying on an wireless connection providing 250 Kbit/s transfer rate, and that sending a reading requires transferring 20KB, we obtain $\lambda_1 = \frac{1}{(20\times 8)/250} = 1.5$ actions per unit of time.

3.1 Experimental results

The implementation of the example is available at

`http://www.imm.dtu.dk/~rvig/reputation-policies-WSN.zip`

The first property we investigate tests whether the probability that *the number of corrupted messages forwarded by a base station is greater than a fixed fraction of the total number of forwarded messages.* This property is expressed in MoSL by the formula

$$\phi_{FwdCorrupted} = \langle \text{“forwarded\_corrupted”} \rangle @\mathsf{server} \rightarrow true$$

This formula relies on the *consumption* operator $\langle T \rangle @l \rightarrow \phi$ [4], which is satisfied whenever a tuple matching template T is located at l and the remaining part of the system satisfies ϕ. Hence, the formula $\phi_{FwdCorrupted}$ is satisfied if and only if a tuple $\langle \text{“forwarded\_corrupted”} \rangle$ is stored in the server tuplespace. Notice that the TESP model of the example has been enriched with some outputs: the base station takes note of the numbers of corrupted and forwarded messages, and uses this information to produce a service tuple in case the number of corrupted forwarded is greater than the fixed percentage.

Exploiting the previous formula, we can specify the more interesting property whether the number of corrupted messages forwarded by a base station is greater than a fixed percentage of the total number of forwarded messages *within time* t, defined as $true\ U^{\leq t} \phi_{FwdCorrupted}$, where the *until* formula $\phi_1 U^{\leq t} \phi_2$ is satisfied

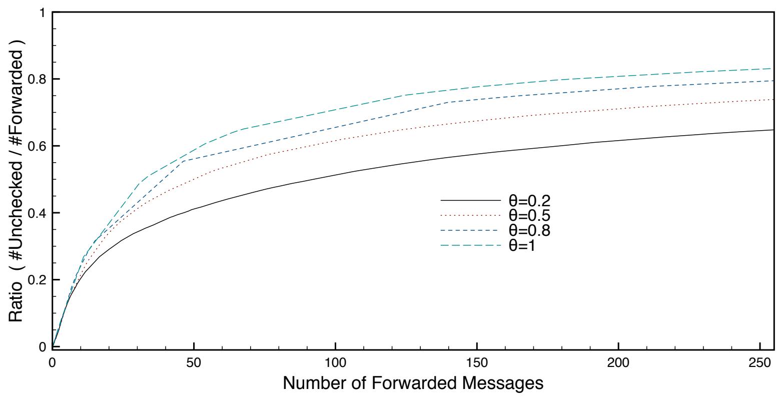

Fig. 1. Ratio between unchecked and forwarded messages.

by all the runs that reach within t time units a state satisfying ϕ_2 while only traversing states that satisfy ϕ_1. The model checking analysis estimates the total probability of the set of runs satisfying such formula. Table 3 reports the results of the analysis with respect to the following parameters: ϵ and p are both set to 0.05; in the network there are four sensors and one base station; three sensors always send correct messages while one behaves according to a parameter θ; three possible sensor's behaviours θ, with $\theta \in \{0.2, 0.5, 0.8\}$ being the probability of sending a correct message; two time limits t, with $t \in \{20, 40\}$; three threshold percentages th, with $th \in \{1\%, 5\%, 10\%\}$.

Inspecting the table, we observe that the probability of satisfying the formula is strictly related to the sensor's behaviour and to the threshold value. In particular, it is not always true that the better the behaviour of the sensor, the lower the probability of forwarding corrupted messages, as one might expect. For $th = 1\%$ the best result - that is, the lowest probability of sending more than 1% corrupted messages - is achieved when the sensor's behaviour is the worst ($\theta = 0.2$). This is due do to the fact that messages from trusted parties are seldom checked, yielding a higher number of forwarded messages.

Figures 1 and 2 show the ratio between the number of unchecked and forwarded messages, and the number of corrupted and forwarded messages, respectively. These results have been obtained thanks to the simulation engine provided with SAM. As expected (see Fig. 1) we observe that the better the sensor behaviour, the higher the ratio of unchecked messages, and thus the higher the quantity of saved resources. On the contrary, Fig. 2 seems not to confirm the results of the model checking analysis: the ratio of corrupted messages over the forwarded is lower for $\theta = 0.8$ than for $\theta = 0.2$. Nonetheless, mind to observe that the simulation computes an average value, while the model checking algorithm estimates the probability of exceeding a threshold. Hence, we can conclude that on average the ratio between corrupted and forwarded messages is lower for $\theta = 0.8$, but the probability that the number of corrupted messages exceeds a percentage of the forwarded messages is lower for $\theta = 0.2$. This is due to the decreasing number of checks that are performed for $\theta = 0.8$, and to the fact that the model checking analysis has a temporal horizon: for a fixed time limit t, the total number of messages processed by the system is higher for $\theta = 0.8$ than for $\theta = 0.2$ (less checks, more time for processing new messages).

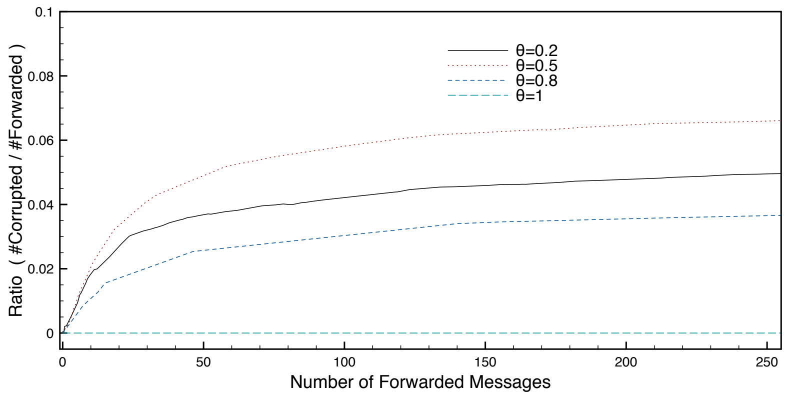

Fig. 2. Ratio between corrupted and forwarded messages.

Hence, the analysis facilitates determining the best trade-off between saving resources (skipping checks) and guaranteeing security (enforcing them, thus forwarding only correct messages). Comparing the number of saved checks with the number of wrong messages that are forwarded we are indeed studying the effect of the policy in terms of resource optimisation and security relaxation, that should match an utility function defined by the system administrator.

4 Conclusions

This work combines two lines of research, showing how aspects and reputation systems can be used to obtain trust-based enforcement of security policies, where trust relationships are determined as the system evolves and according to the behaviour of the parties. Moreover, it shows how the notion of trust can be exploited to optimise resources or increase security, skipping controls if peers are trustworthy or enforcing additional checks when they are not, thus adapting to the environment.

In order to reason about trust-based enforcement of security policies in a natural way we have defined TESP, a stochastic process calculus that extends StoKlaim with primitives for reputation management and aspects. By means of a translation to the original StoKlaim we can verify quantitative properties of interest via statistical model checking and simulation. The expressiveness of the framework has been demonstrated on the simple yet meaningful example of a WSN, where reputations are used to balance the need for security with the shortage of computational resources and power supply.

A promising direction for future work is to investigate the impact of trust-based enforcement of policies on a more optimisation-oriented framework, where explicit optimisation goals are given. This is a challenging task, as even in our simple example multiple conflicting criteria need be considered. At the implementation level, a more succinct translation of TESP into StoKlaim would seem desirable. Moreover, concerning the translation, we also intend to provide a formal proof of its correctness.

Acknowledgement. This work is supported by the IDEA4CPS project, granted by the Danish Research Foundations for Basic Research (DNRF86-10). The work is also partially supported by the EU projects ASCENS (257414) and QUANTICOL (600708), and by the MIUR PRIN project CINA (2010LHT4KM).

References

1. Hankin, C., Nielson, F., Nielson, H.R.: Probabilistic Aspects: Checking Security in an Imperfect World. In: 5th International Conference on Trustworthly Global Computing (TGC'10). (2010) 348–363
2. Jøsang, A., Ismailb, R., Boyd, C.: A survey of trust and reputation systems for online service provision. Decision Support Systems **43**(2) (2007) 618–644
3. Despotovic, Z., Aberer, K.: P2P reputation management: Probabilistic estimation vs. social networks. Computer Networks **50**(4) (2006) 485–500
4. De Nicola, R., Loreti, M.: A modal logic for mobil agents. ACM Trans. Comput. Log. **5**(1) (2004) 79–128
5. De Nicola, R., Katoen, J.P., Latella, D., Loreti, M., Massink, M.: Model checking mobile stochastic logic. Theoretical Computer Science **382**(1) (August 2007) 42–70
6. Loreti, M.: Stochastic Analyser for Mobility (2010) http://rap.dsi.unifi.it/SAM/.
7. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J.m., Irwin, J., Lopes, C.: Aspect-Oriented Programming. In: ECOOP. Volume 1241 of LNCS., Springer (1997) 220–242
8. Georg, G., Ray, I., France, R.B.: Using Aspects to Design a Secure System. In: 8th International Conference on Engineering of Complex Computer Systems (ICECCS'02), IEEE (2002)
9. Win, B.D., Joosen, W., Piessens, F.: Developing Secure Applications through Aspect-Oriented Programming. In: Aspect-Oriented Software Development. (2004) 633–650
10. Hankin, C., Nielson, F., Nielson, H.R., Yang, F.: Advice for Coordination. In: COORDINATION 2008. Volume 5052 of LNCS., Springer (2008) 153–168
11. Bettini, L., Bono, V., De Nicola, R., Ferrari, G.L., Gorla, D., Loreti, M., Moggi, E., Pugliese, R., Tuosto, E., Venneri, B.: The Klaim Project: Theory and Practice. In: Global Computing. Volume 2874 of LNCS., Springer (2003) 88–150
12. Celestini, A., De Nicola, R., Tiezzi, F.: Specifying and Analysing Reputation Systems with a Coordination Language. In: 28th Annual ACM Symposium on Applied Computing (SAC'13), ACM (2013) 1363–1368
13. Legay, A., Delahaye, B.: Statistical Model Checking : An Overview. ArXiv (2010)
14. Calzolai, F., Loreti, M.: Simulation and Analysis of Distributed Systems in Klaim. In: COORDINATION 2010. Volume 6116 of LNCS., Springer (2010) 122–136
15. Gambetta, D. In: Can We Trust Trust? Basil Blackwell (1988)
16. Alzaid, H. et al.: Reputation-based trust systems for wireless sensor networks: A comprehensive review. In: Trust Manag. VII. AICT 401. Springer (2013) 66–82
17. OASIS: eXtensible Access Control Markup Language (XACML) Version 3.0 (2013) http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.
18. De Nicola, R., Ferrari, G.L., Pugliese, R.: KLAIM: A Kernel Language for Agents Interaction and Mobility. IEEE Trans. Software Eng. **24**(5) (1998) 315–330
19. Linda: User's Guide and Reference Manual. Scientific Computing Associates (1995)
20. Nielson, H.R., Nielson, F., Vigo, R.: A Calculus for Quality. In: 9th International Symposium on Formal Aspects of Component Software (FACS'12). Volume 7684 of LNCS., Springer (2012) 188–204