Soft Constraints for Security

Giampaolo Bella [1]

Dipartimento di Matematica e Informatica, Università di Catania, Italy

Stefano Bistarelli [2,3]

Dipartimento di Scienze, Universitá degli Studi "G. D'Annunzio", Pescara, Italy
Istituto di Informatica e Telematica, C.N.R., Pisa, Italy

Simon N. Foley [4]

Department of Computer Science, University College Cork, Ireland

Abstract

Integrity policies and cryptographic protocols have much in common. They allow for a number of participating principals, and consist of sets of rules controlling the actions that principals should or should not perform. They are intended to uphold various security properties, the crucial ones being integrity, confidentiality and authentication.

This paper takes a unified view to the analysis of integrity policies and cryptographic protocols: they are artifacts that must be designed to be sufficiently robust to attack given an understood threat model. For example, integrity policy rules provide resilience to the threat of internal fraud, while cryptographic protocols provide resilience to the threat of replay and related attacks. The framework is modelled using (soft) constraints and analysis corresponds to the soft constraint satisfaction problem. Soft constraints facilitate a quantitative approach to analyzing integrity, confidentiality and authentication. Examples will be given: an integrity policy may achieve different levels of integrity under different circumstances; a protocol message may enjoy different levels of confidentiality for different principals; a principal can achieve different levels of authentication with different principals.

Key words: Constraints, Security Protocols, Integrity Policy.

[1] Email: giamp@dmi.unict.it
[2] Email: bista@sci.unich.it
[3] Email: stefano.bistarelli@iit.cnr.it
[4] Email: s.foley@cs.ucc.ie

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

1 Introduction

Integrity, confidentiality and authentication are essential security properties. In this paper we use the constraint solving framework to uniformly study all of them in the context of *integrity policies* and *cryptographic protocols*.

An integrity policy defines the situations when modification of information is authorized and is enforced by the security mechanisms of the system. This is done by modeling the system and infrastructure in terms of the *constraints* that they impose over security relevant components of the system. This results in a definition of integrity consistency that can be solved as a constraint solving problem A further advantage to using a constraint based framework is that it becomes possible to carry out a quantitative analysis of integrity using *soft constraints* [6, 7, 8, 2, 18, 19, 11, 14, 17]. A quantitative analysis provides a fine-grained measure of how secure a system is, rather than the simple coarse-grained false/true provided by the conventional 'crisp' constraints.

A cryptographic protocol is a prescribed set of message exchanges between principals of an insecure network. Confidentiality is the property of a message to remain undisclosed to malicious principals. Another crucial goal is authentication, confirming a principal's participation in a protocol session. Confidentiality has been essentially formalised as a mere "yes or no" property thus far, so one can just claim that a key is confidential or not. The motivation for our research was studying a finer formal notion for the goal. We have developed the notion of *l-confidentiality*, where l is the *security level* signifying the strength with which the goal is met. The security level belongs to the carrier set of a semiring, as we adopt semiring-based soft constraint programming. Each principal assigns his own security level to each message — different levels to different messages — expressing the principal's trust on the message. This lets us formalise that different levels of a goal are granted to different principals.

The paper is organized as follows. Section 2 provides an introduction to constraints and the constraint solving problem. Section 3 first proposes an abstract approach to modeling systems within a crisp constraint framework, and then describes how soft constraints are used to carry out quantitative integrity analysis. The paper continues with the presentation of the crucial SCSPs for analyzing cryptographic protocols, with the definitions concerning confidentiality, and with a significant example (Section 4). Finally, some conclusions (Section 5) are given.

2 Introduction to Constraint Solving

Constraint Solving is an emerging software technology for declarative description and effective solving of large problems. The constraint programming process consists of the generation of requirements (constraints) and solution of these requirements, by specialized constraint solvers.

When the requirements of a problem are expressed as a collection of boolean predicates over variables, we obtain what is called the *crisp* (or classical) Constraint Satisfaction Problem (CSP). In this case the problem is solved by finding any assignment of the variables that satisfies all the constraints.

Sometimes, when a deeper analysis of a problem is required, *soft* constraints are used instead. Soft constraints associate a qualitative or quantitative value either to the entire constraint or to each assignment of its variables. Such values are interpreted as levels of preference or importance or cost. The levels are usually ordered, reflecting the fact that some levels (constraints) are better than others. When using soft constraints it is necessary to specify, via suitable combination operators, how the level of preference of a global solution is obtained from the preferences in the constraints.

Several formalizations of the concept of *soft constraints* are currently available. In the following, we refer to the formalization based on c-semirings [6, 7, 9, 2], which can be shown to generalize and express both crisp and soft constraints [7, 3].

2.1 Semiring-based CSPs

A semiring-based constraint assigns to each instantiation of its variables an associated value from a partially ordered set. When dealing with crisp constraints, the values are the boolean *true* and *false* representing the admissible and/or non-admissible values; when dealing with soft constraints the values are interpreted as preferences or probabilities or costs.

The framework must also handle the combination of constraints. To do this one must take into account such additional values, and thus the formalism must provide suitable operations for combination ($\times$) and comparison ($+$) of tuples of values and constraints. This is why this formalization is based on the concept of c-semiring.

2.2 Semirings

A *semiring* is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that: A is a set and $\mathbf{0}, \mathbf{1} \in A$; $+$ is commutative, associative and $\mathbf{0}$ is its unit element; $\times$ is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element. A c-semiring ("c" stands for "constraint-based") is a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that $+$ is idempotent with $\mathbf{1}$ as its absorbing element and $\times$ is commutative [7, 2]. In the following we will always use the word semiring as standing for c-semiring.

Let us consider the relation $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$. It is possible to prove that: $\leq_S$ is a partial order; $+$ and $\times$ are monotone on $\leq_S$; $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum, and $\langle A, \leq_S\rangle$ is a complete lattice with lowest upper bound operator $+$. Moreover, if $\times$ is idempotent, then: $+$ distributes over $\times$, and $\langle A, \leq_S\rangle$ is a complete distributive lattice with greatest lower bound operator $\times$. The $\leq_S$ relation is what we will use to compare tuples and constraints: $a \leq_S b$ intuitively means that b is better than a.

2.3 Constraint Problems

Given a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ and an ordered set of variables V over a finite domain D, a *constraint* is a function which, given an assignment $\eta : V \to D$ of the variables, returns a value of the semiring.

By using this notation we define $\mathcal{C} = \eta \to A$ as the set of all possible constraints that can be built starting from S, D and V.

Consider a constraint $c \in \mathcal{C}$. We define his support as $supp(c) = \{v \in V \mid \exists \eta, d_1, d_2 . c\eta[v := d_1] \neq c\eta[v := d_2]\}$, where

$$\eta[v := d]v' = \begin{cases} d & \text{if } v = v', \\ \eta v' & \text{otherwise.} \end{cases}$$

Note that $c\eta[v := d_1]$ means $c\eta'$ where η' is η modified with the association $v := d_1$ (that is the operator [ ] has precedence over application).

A *constraint solving problem* is a pair $\langle C, con\rangle$ where $con \subseteq V$ and C is a set of constraints: *con* is the set of variables of interest for the constraint set C, which however may concern also variables not in *con*. Note that a classical CSP is an SCSP where the chosen c-semiring is: $S_{CSP} = \langle \{false, true\}, \vee, \wedge, false, true\rangle$.

Many other "soft" CSPs (Probabilistic, weighted, ...) can be modeled by using a suitable semiring structure ($S_{prob} = \langle [0,1], max, \times, 0, 1\rangle$, $S_{weight} = \langle \mathcal{R}, min, +, +\infty, 0\rangle$, ...).

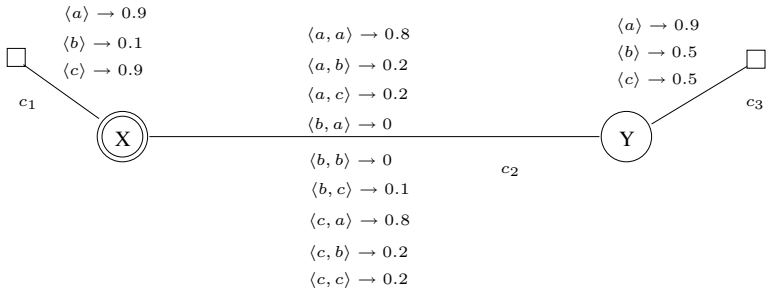

Fig. 1. A fuzzy CSP.

Example 1 Figure 1 shows the graph representation of a fuzzy CSP [5]. Variables X and Y, and constraints are represented respectively by nodes and by undirected (unary for c_1 and c_3 and binary for c_2) arcs, and semiring values are written to the right of the corresponding tuples. The variables of interest (that is the set *con*) are represented with a double circle. Here we assume that the domain D of the variables contains only elements a, b and c.

If semiring values represent probability/fuzziness values then, for instance, the tuple $\langle a, c\rangle \to 0.2$ in constraint c_2 can be interpreted to mean that the

[5] Fuzzy CSPs can be modeled in the SCSP framework by choosing the c-semiring $S_{FCSP} = \langle [0,1], max, min, 0, 1\rangle$.

probability/fuzziness of X and Y having values a and c, respectively, is 0.2. $\triangle$

2.4 Combining constraints

When there is a set of soft constraints $\mathcal{C}$, the combined weight of the constraints is computed using the operator $\otimes : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ defined as $(c_1 \otimes c_2)\eta = c_1\eta \times_S c_2\eta$.

Given a constraint $c \in \mathcal{C}$ and a variable $v \in V$, the *projection* of c over $V - \{v\}$, written $c \Downarrow_{(V-\{v\})}$ is the constraint c' s.t. $c'\eta = \sum_{d \in D} c\eta[v := d]$. Informally, projecting means eliminating some variables from the support. This is done by associating to each tuple over the remaining variables a semiring element which is the sum of the elements associated by the original constraint to all the extensions of this tuple over the eliminated variables. In short, combination is performed via the multiplicative operation of the semiring, and projection via the additive one.

2.5 Solutions

Solution of an SCSP $P = \langle C, con \rangle$ is the constraint $Sol(P) = (\bigotimes C) \Downarrow_{con}$. That is, we combine all constraints, and then project over the variables in *con*. In this way we get the constraint with support (not greater than) *con* which is "induced" by the entire SCSP. Note that when all the variables are of interest we do not need to perform any projection.

Solutions are constraints in themselves and can be ordered by extending the $\leq_S$ order. We say that a constraint c_1 is at least as constraining as constraint c_2 if $c_1 \sqsubseteq c_2$, where for any assignment η of variables then

$$c_1 \sqsubseteq c_2 \equiv c_1\eta \leq_S c_2\eta$$

Thus, if $c_1 \sqsubseteq c_2$ holds, then constraint c_1 may be thought of as a *refinement*, or 'suitable' (more restrictive) replacement of constraint c_2.

Example 2 Consider again the solution of the fuzzy CSP of Figure 1. It associates a semiring element to every domain value of variable X. Such an element is obtained by first combining all the constraints together and then projecting the obtained constraint over X.

For instance, for the tuple $\langle a, a \rangle$ (that is, $X = Y = a$), we have to compute the minimum between 0.9 (which is the value assigned to $X = a$ in constraint c_1), 0.8 (which is the value assigned to $\langle X = a, Y = a \rangle$ in c_2) and 0.9 (which is the value for $Y = a$ in c_3). Hence, the resulting value for this tuple is 0.8. We can do the same work for tuple $\langle a, b \rangle \rightarrow 0.2$, $\langle a, c \rangle \rightarrow 0.2$, $\langle b, a \rangle \rightarrow 0$, $\langle b, b \rangle \rightarrow 0$, $\langle b, c \rangle \rightarrow 0.1$, $\langle c, a \rangle \rightarrow 0.8$, $\langle c, b \rangle \rightarrow 0.2$ and $\langle c, c \rangle \rightarrow 0.2$. The obtained tuples are then projected over variable X, obtaining the solution $\langle a \rangle \rightarrow 0.8$, $\langle b \rangle \rightarrow 0.1$ and $\langle c \rangle \rightarrow 0.8$. $\triangle$

3 Analyzing Integrity Policies

An integrity policy defines the situations when modification of information is authorized and is enforced by the security mechanisms of the system. However, in a complex application system it is possible that an integrity policy may have been incorrectly specified and, as a result, a user may be authorized to modify information that can lead to an unexpected system compromise. In this section, inspired from the results of [4, 5], we describe how constraint solving can be used to model and analyze the effectiveness of application system integrity policies.

In the following we show by using examples from [12, 13], how functional requirements can be expressed as requirements in terms of constraints on variables that are invariant over the lifetime of the system.

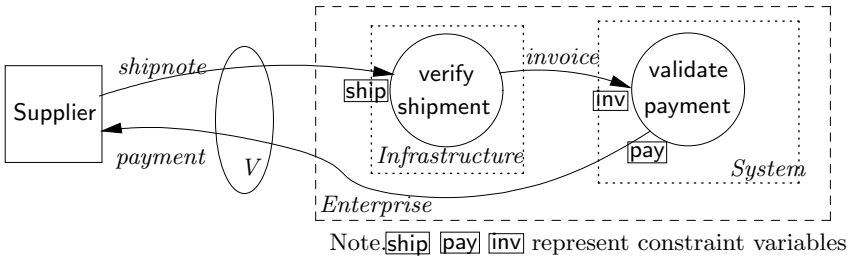

Fig. 2. A simple payment enterprise

Example 3 A simple enterprise receives shipments, and generates associated payments for a supplier. Requirements Analysis identifies the actions *shipnote* and *payment*, corresponding to the arrival of a shipment (note) and its associated payment, respectively. For the purposes of integrity, the analysis has identified a requirement that the system should not pay its supplier more than the stated value of goods shipped.

Let the constraint variables ship and pay represent the total value of goods shipped to date and the total value of payments made to date, respectively. Constraint Probity describes the requirement as an invariant over variables ship and pay.

$$\mathsf{Probity} \equiv \mathsf{pay} \leq \mathsf{ship}$$

Figure 2 outlines a possible implementation of this requirement. A clerk verifies shipment notes and enters invoice details (action *invoice*) to a computer system, which in turn, generates payment to the supplier. This implementation is described in terms of variables ship, pay and variable inv which represents the total value of invoices generated to date.

A clerk should not process more invoices than shipments and, therefore, the clerk's behavior is represented by the following constraint.

$$\mathsf{Clerk} \equiv \mathsf{inv} \leq \mathsf{ship}$$

The requirement on the invoice processing application system is

$$\mathsf{Appl} \equiv \mathsf{pay} \leq \mathsf{inv}$$

and the enterprise design is specified as the constraint

$$\mathsf{Imp1} \equiv \mathsf{Appl} \otimes \mathsf{Clerk}$$

obtained by combining together Appl and Clerk constraints. Intuitively, integrity is ensured in this system since Imp1 ensures the high-level requirement Probity. △

In the above example, the supplier's interface V to the system is modeled in terms of the variables ship and pay. Constraints between these variables are used to characterize our requirements for the system. We want to ensure that the implementation upholds probity through this interface, that is,

$$\mathsf{Imp1}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \sqsubseteq \mathsf{Probity}$$

We are unconcerned about the possible values of the 'internal' variable inv and thus the constraint relation $\mathsf{Imp1}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}}$ describes the constraints in Imp1 that exist between variables ship and pay. By definition, the above equation defines that all of the possible solutions of $\mathsf{Imp1}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}}$ are solutions of Probity, that is, for any assignment η of variables then

$$\mathsf{Imp1}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}}\ \eta \leq_S \mathsf{Probity}\ \eta$$

Definition 3.1 *We say that the requirement S locally refines requirement R through the interface described by the set of variables V iff $S_{\Downarrow V} \sqsubseteq R_{\Downarrow V}$.*

Example 4 Continuing Example 3, assume that the application system will behave reliably and uphold Appl. However, it is not reasonable to assume that the clerk will always act reliably as Clerk. In practice, the clerk could take on any behavior:

$$\overline{\mathsf{Clerk}} \equiv (\mathsf{inv} \leq \mathsf{ship} \vee \mathsf{inv} > \mathsf{ship}) = \mathit{true}$$
$$\mathsf{Imp2} \equiv \overline{\mathsf{Clerk}} \otimes \mathsf{Appl}$$

Imp2 is a more realistic representation of the actual enterprise. It more accurately reflects the reliability of its infrastructure than the previous design Imp1. However, since inv is no longer constrained it can take on any value, and therefore, pay is unconstrained and we have

$$\mathsf{Imp2}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \not\sqsubseteq \mathsf{Probity}$$

that is, the implementation of the system is not sufficiently robust to be able to deal with internal failures in a safe way and uphold the original probity requirement. △

In [15], integrity is given as one attribute of *dependability*. Dependability is characterized as a *"property of a computer system such that reliance can be justifiably placed on the service it delivers"* [15]. In [12, 13] this notion of dependability may be viewed as a class of refinement whereby the nature of the reliability of the enterprise is explicitly specified.

Definition 3.2 *(Dependability) If R gives requirements for an enterprise and S is its proposed implementation, including details about the nature of the reliability of its infrastructure, then S is as dependably safe as R at interface that is described by the set of variables E if and only if $S_{\Downarrow E} \sqsubseteq R_{\Downarrow E}$*

Separation of duties [20, 10] is a common implementation technique for achieving integrity. While fault-tolerant techniques replicate an operation, separation of duties can be thought of as a partitioning of the operation across different user domains.

Example 5 When a shipment arrives a clerk verifies the consignment at goods-inwards (entering *consign* into the system). When an invoice arrives, a different clerk enters details into the system, and if the invoice matches a consignments, a payment is generated. So long as the operations are separated then a single clerk entering a bogus consignment or invoice can be detected by the system. This is depicted in Figure 3.

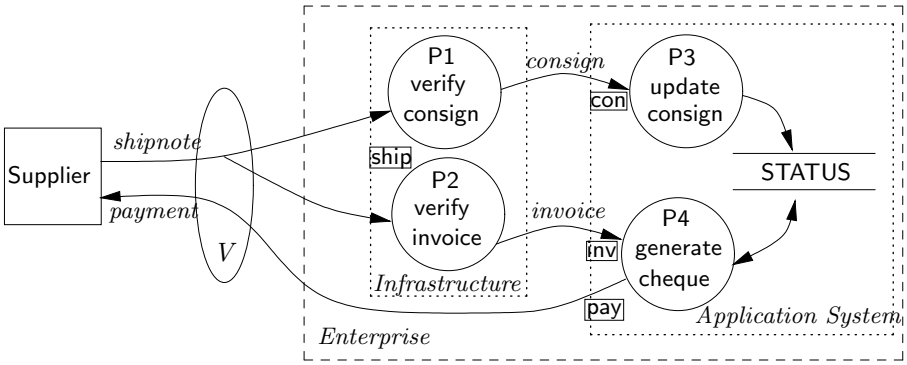

Fig. 3. Supporting separation of duties

Let variables inv and con represent the total value of invoices and consignments, respectively, received to date. Specifications Clerk1, Clerk2 and App3 define the constraints on the system variables, reflecting invariants that are expected to be upheld by the clerks and the application system.

$$\mathsf{Clerk1} \equiv \mathsf{con} \leq \mathsf{ship}$$
$$\mathsf{Clerk2} \equiv \mathsf{inv} \leq \mathsf{ship}$$
$$\mathsf{App3} \equiv \mathsf{pay} \leq min(\mathsf{con}, \mathsf{inv})$$

This system is as dependably safe as Probity even when a single clerk fails, that is, we have

$$(\mathsf{Clerk1} \otimes \mathsf{Clerk2} \otimes \mathsf{App3})_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \sqsubseteq \mathsf{Probity}$$
$$(\mathsf{Clerk2} \otimes \mathsf{App3})_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \sqsubseteq \mathsf{Probity}$$
$$(\mathsf{Clerk1} \otimes \mathsf{App3})_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \sqsubseteq \mathsf{Probity}$$

Note that the absence of a constraint means that it does not restrict variables to any values. However, the system is not resilient to the failure of both clerks nor to the failure of the application software. Removing the 'normal behavior' constraints imposed by both clerks or by the application yields the following.

$$\mathsf{App3}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \not\sqsubseteq \mathsf{Probity}$$
$$(\mathsf{Clerk1} \otimes \mathsf{Clerk2})_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \not\sqsubseteq \mathsf{Probity}$$

As currently defined, our specification favors the payment-enterprise, not the supplier: payments may be very late, or not made at all, but are never bogus. If a clerk fails then payment may not be made. In reality, the infrastructure contains many additional components; audit logs to record failures and supervisors, who make judgments and rectify these inconsistencies. △

3.1 Quantitative integrity analysis

The examples in the previous section use crisp constraints to describe system requirements and implementations. When a quantitative analysis of the system is required then it is necessary to represent these properties using soft constraints.

Example 6 Consider again the Probity requirement and suppose that we aim not only to have a correct implementation, but, if possible, to have the "best" possible implementation. To do this we consider a soft constraint between variables ship and pay that assigns to the configuration $\mathsf{ship} = a$ and $\mathsf{pay} = b$ the preference level represented by the integer $a - b$[6]. If we are looking for the best implementation for the buyer, then we will try to maximize this level. In this way, different instances of the same system can be compared, and different implementations can be checked and analyzed.

Soft constraints also provide a basis for evaluating and comparing less resilient implementations that cannot uphold the intended requirement. For example, if an acceptable implementation Imp cannot be found to satisfy

$$\mathsf{Imp}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \sqsubseteq \mathsf{Probity}$$

then one might be satisfied (in some sense) by selecting the best of the less resilient, but acceptable implementations. Given insufficiently resilient implementations Imp_1 and Imp_2 then their corresponding semiring levels provide a relative ordering that allow the selection of the 'best' of the less resilient implementations.

[6] This value represent how much pay differs from ship. Our goal is to have pay=ship, but sometimes this is impossible and our goal will be to minimize the $a - b$ difference.

Probabilistic based reasoning can also be done within the soft constraints framework. For example, consider an implementation Imp3 that ensures that the number of payments is never more than 3, regardless of the number of shipments received. This is represented as:

$$\mathsf{Imp3} \equiv \mathsf{pay} \leq 3.$$

Assume that there is a constraint on variable ship that specifies the probability of the possible number of shipments made at a certain time. If the nature of the probability distribution is such that it is generally more likely that the value of ship is greater than 3, then Imp3 is a not unreasonable implementation (despite $\mathsf{Imp3}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \sqsubseteq \mathsf{Probity}$ not holding). △

4 Analyzing Cryptographic Protocols

Security protocols stipulate how the remote principals of a computer network should interact in order to obtain specific security goals. The crucial goals of confidentiality and authentication may be achieved in various forms, each of different strength.

In this section, inspired from the results of [1], we describe how soft constraint solving can be used to develop a uniform formal notion for the two goals. They are no longer formalised as mere yes/no properties as in the existing literature, but gain an extra parameter, the *security level*. For example, different messages can enjoy different levels of confidentiality, or a principal can achieve different levels of authentication with different principals.

4.1 The Security Semiring

Our *security semiring* is used to specify each principal's trust on the security of each message, that is each principal's *security level* on each message. The security levels range from the most secure (highest) level *unknown* to the least secure (lowest) level *public*. Intuitively, if A's security level on m is *unknown*, then no principal (included A) knows m according to A, and, if A's security level on m is *public*, then all principals potentially know m according to A. The lower A's security level on m, the higher the number of principals that A believes authorised to know m. For simplicity, we state no relation between the granularity of the security levels and the number of principals authorised to know m.

More formally, given a natural number n, we define the set L of *security levels* as follows:

$$L = \{unknown,\ private,\ traded_1,\ traded_2,\ \ldots\ ,\ traded_n,\ public\}$$

We introduce an additive operator, $+_{sec}$, and a multiplicative operator, $\times_{sec}$. To allow for a compact definition of the two operators, and to simplify

the following treatment, let us define a convenient double naming:

- $unknown \equiv traded_{-1}$
- $private \equiv traded_0$
- $public \equiv traded_{n+1}$

Let us consider an index i and an index j both belonging to the closed interval $[-1,\ n+1]$ of integers. We define $+_{sec}$ and $\times_{sec}$ by the following axioms.
Ax. 1: $traded_i\ +_{sec}\ traded_j\ =\ traded_{max(i,j)}$
Ax. 2: $traded_i\ \times_{sec}\ traded_j\ =\ traded_{min(i,j)}$

The structure $\mathcal{S}_{sec} = \langle L, +_{sec}, \times_{sec}, public, unknown \rangle$ can be easily verified to be a c-semiring.

Using the security semiring, we define the *network constraint system* $CS_n = \langle \mathcal{S}_{sec}, \mathcal{D}, \mathcal{V} \rangle$ where:

- $\mathcal{S}_{sec}$ is the security semiring (Section 4.1);
- $\mathcal{V}$ is a bounded set of variables.
- $\mathcal{D}$ is a bounded set of values including the empty message $\{\!|\}$ and all atomic messages, as well as all messages recursively obtained by concatenation and encryption.

. The *network constraint system* represents the computer network on which the cryptographic protocols can be executed.

The elements of $\mathcal{V}$ stand for the network principals, and the elements of $\mathcal{D}$ represent all possible messages. Atomic messages typically are principal names, timestamps, nonces and cryptographic keys.

The development of the principals' security levels from manipulation of the messages seen during the protocol sessions can be formalised as a *security entailment*, that is an entailment relation between constraints [9].

We define rules to compute the security levels that each principal gives to the newly generated messages. The rules establish that the security level of a message gets somewhat lower each time the message is manipulated by encryption or decryption.

A possible set of rules is presented in Figure 4, where function *def* is associated to a generic constraint projected on a generic principal A. Different rules can be studied to capture other features.

4.2 Initial, Policy and Imputable SCSPs

The policy for a protocol $\mathcal{P}$ is a set of rules stating, among other things, the preconditions necessary for the protocol execution, such as which messages are public, and which messages are private for which principals.

It is intuitive to capture these policy rules by our security levels (Section 4.1). Precisely, these rules can be translated into unary constraints for the network constraint system. For each principal $A \in \mathcal{V}$, we define a unary constraint that states A's security levels as follows. It associates security level

Concatenation:

$$\frac{v_1, v_2 < unknown; \quad def(m_1) = v_1; \quad def(m_2) = v_2; \quad def(\{\!|m_1, m_2|\!\}) = v_3}{def(\{\!|m_1, m_2|\!\}) = (v_1 +_{sec} v_2) \times_{sec} v_3}$$

Splitting:

$$\frac{v_3 < unknown; \quad def(m_1) = v_1; \quad def(m_2) = v_2; \quad def(\{\!|m_1, m_2|\!\}) = v_3}{def(m_1) = v_1 \times_{sec} v_3; \quad def(m_2) = v_2 \times_{sec} v_3}$$

Encryption:

$$\frac{\begin{array}{l} traded_{l_1}, traded_{l_2} < unknown; \\ def(m_1) = traded_{l_1}; \ def(m_2) = traded_{l_2}; \ def(\{\!|m_1|\!\}_{m_2}) = traded_{l_3} \end{array}}{def(\{\!|m_1|\!\}_{m_2}) = (traded_{l_1+1} +_{sec} traded_{l_2}) \times_{sec} traded_{l_3}}$$

Decryption:

$$\frac{\begin{array}{l} traded_{l_2}, traded_{l_3} < unknown; \\ def(m_1) = traded_{l_1}; \ def(m_2^{-1}) = traded_{l_2}; \ def(\{\!|m_1|\!\}_{m_2}) = traded_{l_3} \end{array}}{def(m_1) = traded_{l_1} \times_{sec} traded_{l_2+1} \times_{sec} traded_{l_3}}$$

Fig. 4. Entailment rules for security levels

public to those messages that are known to all, typically principal names, timestamps and public keys; level *private* to A's initial secrets, such as keys (e.g., A's long-term key if $\mathcal{P}$ uses symmetric cryptography, or A's private key if $\mathcal{P}$ uses asymmetric cryptography, or A's pin if $\mathcal{P}$ uses smart cards); level *unknown* to all remaining domain values (including, e.g., the secrets that A will invent during the protocol execution, or other principals' initial secrets). This procedure defines what we name *initial SCSP for* $\mathcal{P}$, which specifies the principals' security levels when no session of $\mathcal{P}$ has yet started.

The policy for a protocol $\mathcal{P}$ also specifies how the messages that must be exchanged during a session between a pair of principals are formed. We read from the protocol policy each allowed step of the form $A \rightarrow B : m$ and its informal description, which explains whether A invents m or part of it. Then, we build the *policy SCSP for* $\mathcal{P}$ by adding new constraints to the initial SCSP according to the event that is considered. If that event is a principal A's inventing a message n, then a unary constraint is added on variable A assigning security level *private* to the domain value n (and *unknown* to all other values). If that event is a principal A's sending a message m to a principal B, then the semiring value, alias security level, associated to message m over A is considered. This level is computed by entailment (Figure 4) whenever m is obtained by manipulation of other messages (rather than m being e.g. a fresh nonce just invented with security level *private* by the previous case of the algorithm). A binary constraint that assigns the newly computed security level to the tuple $\langle \{\!|\}, m \rangle$ (and *unknown* to all other tuples) is now added to

the current SCSP on the pair of variables A and B. This reasoning is repeated for each of the bounded number of events allowed by the policy. When there are no more events to process, the current SCSP is returned as policy SCSP for $\mathcal{P}$, which is our formal model for the protocol.

A real-world network history induced by a protocol $\mathcal{P}$ must account for malicious activity by some principals. Each such history can be viewed as a sequence of events of the forms: a principal's inventing new messages, a principal's sending messages that are not intercepted, and a principal's sending messages that are intercepted. While the second event signifies that the intended recipient of a message indeed gets the message, the third signifies that some malicious principal prevents the delivery of the message that is sent.

We can model any network configuration at a certain point in any real-world network history as an SCSP. We take as inputs a protocol $\mathcal{P}$ and a network configuration nc originated from the protocol execution. The processing of the third type of event is added: when a message is sent by A to B and is intercepted by another principal C, the corresponding constraint must be stated on the pair A, C rather than A, B. We obtain what we name an *imputable SCSP* for $\mathcal{P}$.

4.3 Formalising Confidentiality

Using the security levels, we develop uniform definitions of confidentiality and of confidentiality attack, which appear to capture any policy requirement. Intuitively, if a principal's security level on a message is l, then the message is *l-confidential* for the principal because the security level in fact formalises the principal's trust on the security, that is confidentiality, of the message.

In practice, establishing whether our definition of *l-confidentiality* holds in an SCSP requires calculating the solution of the imputable SCSP and projecting it on certain principals of interest. The higher l, the stronger the goal.

Definition 4.1 [l-confidentiality] Given an imputable SCSP p and a principal A, we say that there is *l-confidentiality of m for A in* p iff $Sol(\mathsf{p}) \Downarrow_{\{A\}} (m) = l$.

By a *preliminary analysis*, we can study what goals the protocol achieves in ideal conditions where no principal acts maliciously. Most importantly, by an *empirical analysis*, we can study what goals the protocol achieves on a specific network configuration arising from the protocol execution under realistic threats. We concentrate on the corresponding imputable SCSP, calculate its solution and project it on a principal of interest: we obtain the principal's security levels on all messages. Having done the same operations on the policy SCSP, we can compare the outcomes. If some level from the imputable is lower than the corresponding level from the policy, then there is an attack in the imputable SCSP. In fact, some malicious operations contributing to the network configuration modeled by the imputable SCSP have taken place so to lower some of the security levels stated by the policy SCSP. It is important

to stress that any principal might have performed, either deliberately or not, those operations.

Definition 4.2 [Confidentiality attack]
Given a policy SCSP P, an imputable SCSP p for the same protocol, and a principal A, we say that *there is a confidentiality attack by A on m in p* iff there is l-confidentiality of m in P for A, l'-confidentiality of m in p for A, and $l' < l$.

Therefore, there is a confidentiality attack by A on m in p iff $Sol(\mathsf{P}) \Downarrow_{\{A\}} (m) < Sol(\mathsf{P}) \Downarrow_{\{A\}} (m)$. The more an attack lowers a security level, the worse that attack, so confidentiality attacks can be variously compared.

The authentication goal can be formalized in a similar fashion (details can be found elsewhere [1]).

4.4 An empirical analysis of the Needham-Schroeder protocol

Figure 5 presents the asymmetric Needham-Schroeder protocol, which is so popular that it requires little comments.

1. $A \rightarrow B : \{\!|Na, A|\!\}_{Kb}$
2. $B \rightarrow A : \{\!|Na, Nb|\!\}_{Ka}$
3. $A \rightarrow B : \{\!|Nb|\!\}_{Kb}$

Fig. 5. The asymmetric Needham-Schroeder protocol

The goal of the protocol is *authentication*: at completion of a session initiated by A with B, A should get evidence to have communicated with B and, likewise, B should get evidence to have communicated with A. Assuming that encryption is perfect and that the nonces are truly random, authentication is achieved here by confidentiality of the nonces. Indeed, upon reception of Na inside message 2, A would conclude that she is interacting with B, the only principal who could retrieve Na from message 1. In the same fashion, when B receives Nb inside message 3, he would conclude that A was at the other end of the network because Nb must have been obtained from message 2, and no-one but A could perform this operation.

Lowe discovers [16] that the protocol suffers the attack in Figure 6, whereby a malicious principal C masquerades as a principal A with a principal B, after A initiated a session with C. The attack, which sees C interleave two sessions, indicates failure of the authentication of A with B, which follows from failure of the confidentiality of Nb. The security levels of all other principals on the nonces Na and Nb are *unknown*. So, by Definition 4.1, those nonces are *unknown*-confidential for any principal different from A or B.

We start off by building the initial SCSP, whose fragment for principals A and B is in Figure 7 (the following only features suitable SCSP fragments

$$
\begin{aligned}
&1. \quad A \rightarrow C : \{\!|Na, A|\!\}_{Kc} \\
&1'. \quad C \rightarrow B : \{\!|Na, A|\!\}_{Kb} \\
&2'. \quad B \rightarrow A : \{\!|Na, Nb|\!\}_{Ka} \\
&2. \quad C \rightarrow A : \{\!|Na, Nb|\!\}_{Ka} \\
&3. \quad A \rightarrow C : \{\!|Nb|\!\}_{Kc} \\
&3'. \quad C \rightarrow B : \{\!|Nb|\!\}_{Kb}
\end{aligned}
$$

Fig. 6. Lowe's attack to the Needham-Schroeder Protocol

pertaining to the principals of interest).

Fig. 7. Fragment of the initial SCSP for Needham-Schroeder protocol

Then, we build the policy SCSP for the protocol. Figure 8 presents the fragment pertaining to a single session between principals A and B. The Figure indicates that, while A's security level on her nonce Na was initially *private*, it is now lowered to $traded_1$ by entailment because of the binary constraint formalising step 2 of the protocol. Similarly, B's security level on Nb is now $traded_2$ though it was originally *private*. The Figure omits the messages that are not relevant to the following discussion.

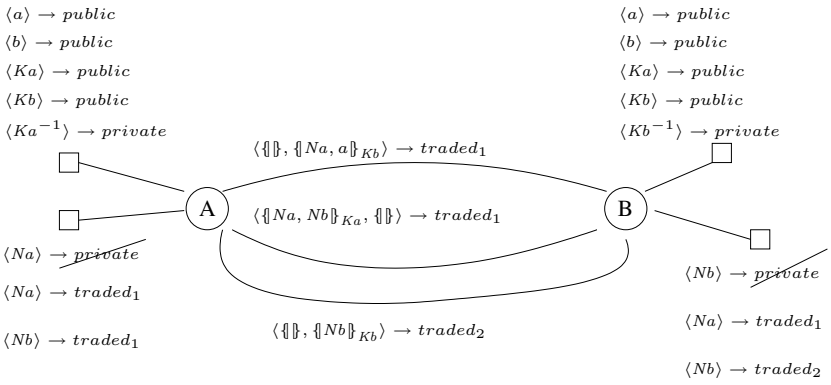

Fig. 8. Fragment of the policy SCSP for the Needham-Schroeder protocol

At this stage, we build the imputable SCSP given in Figure 9. It formalises the network configuration defined by Lowe's attack. The solution of this SCSP projected on variable C is a constraint that associates security level $traded_4$ to the nonce Nb. Following Definition 4.1, Nb is $traded_4$-confidential for C in this SCSP. Hence, by Definition 4.2, there is a deliberate confidentiality attack by

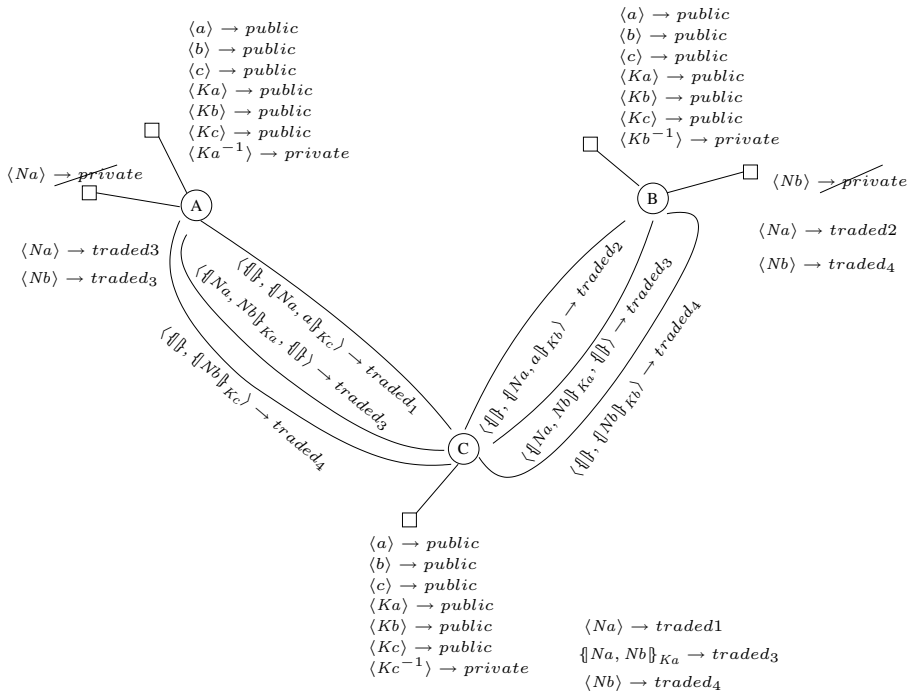

Fig. 9. Fragment of the Imputable SCSP corresponding to Lowe's attack

C on Nb in this problem, because Nb got level *unknown* in the policy SCSP. This leads to Lowe's attack.

We have discovered another attack in the same problem [1]. The problem solution projected on variable B associates security level $traded_2$ to the nonce Na, which instead got level *unknown* in the policy SCSP. This signifies that B has learnt a nonce that he was not allowed to learn by policy, that there is an indeliberate confidentiality attack by B on Na — notice that the two attacks are uniformly formalised. As a consequence of the former attack, Lowe reports that, if B is a bank, C can steal money from A's account. As a consequence of our attack, should also A be a bank, B would be able to steal money from C's account by deliberately exploiting his knowledge of Na (details appear elsewhere [1]).

5 Conclusions

The contribution of this paper is a unified technique for quantitatively analyzing the configuration of integrity policies and measuring the confidentiality and authentication goals of cryptographic protocols. In both cases, the use of soft, rather than crisp, constraints significantly deepens the analysis.

With integrity policies soft constraints allow the use of quantitative information in modeling the system and infrastructure configuration. For instance, by associating probability measures with component failures, we can describe that a system configuration/policy achieve integrity within some degree of probability.

With cryptographic protocols soft constraints introduce an extra parameter, called security level. It allows formal statements that different protocol principants get different forms of the confidentiality and authentication goals. Even comparing the forms of the same goal as achieved by different protocols appears to be at reach.

We were pleased to observe that our technique easily scales up to both contexts: with integrity policies, the implementations are compared with a given specification; with cryptographic protocols, the protocol runs under study are compared with an ideal run without spies. In both cases, the analysis reduces to checking the solutions of specific constraint satisfaction problems, a task that can be significantly supported by mechanical tools.

References

[1] Bella, G. and S. Bistarelli, *Soft constraint programming to analysing security protocol*, Theory and Practice of Logic Programming (TPLP) (Special Issue on Verification and Computational Logic) **4** (2004), pp. 1–28.

[2] Bistarelli, S., "Semirings for Soft Constraint Solving and Programming," LNCS **2962**, Springer, 2004.

[3] Bistarelli, S., H. Fargier, U. Montanari, F. Rossi, T. Schiex and G. Verfaillie, *Semiring-based csps and valued csps: Frameworks, properties, and comparison*, CONSTRAINTS: An international journal. Kluwer **4** (1999).

[4] Bistarelli, S. and S. Foley, *Analysis of integrity policies using soft constraints*, in: *Proc. of IEEE Workshop Policies for Distributed Systems and Networks*, 2003, pp. 77–80.

[5] Bistarelli, S. and S. Foley, *A constraint based framework for dependability goals: Integrity*, in: *Proc. of the 22nd International Conference on Computer Safety, Reliability and Security (SAFECOMP2003), 23-26 September 2003, Edinburgh, Scotland, United Kingdom*, LNCS **2788** (2003), pp. 77–80.

[6] Bistarelli, S., U. Montanari and F. Rossi, *Constraint Solving over Semirings*, in: *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)* (1995).

[7] Bistarelli, S., U. Montanari and F. Rossi, *Semiring-based Constraint Solving and Optimization*, Journal of the ACM (1997), pp. 201–236.

[8] Bistarelli, S., U. Montanari and F. Rossi, *Soft concurrent constraint programming*, in: *Proc. ESOP, April 6 - 14, 2002, Grenoble, France*, LNCS (2002).

[9] Bistarelli, S., U. Montanari and F. Rossi, *Soft concurrent constraint programming*, in: *Proc. ESOP, Grenoble, France*, LNCS **2305** (2002), pp. 53–67.

[10] Clark, D. D. and D. R. Wilson, *A comparison of commercial and military computer security models*, in: *Proceedings Symposium on Security and Privacy* (1987), pp. 184–194.

[11] Fargier, H. and J. Lang, *Uncertainty in Constraint Satisfaction Problems: a Probabilistic Approach*, in: *Proc. of European Conference on Symbolic and Qualitative Approaches to Reasoning and Uncertainty (ECSQARU)* (1993), pp. 97–104.

[12] Foley, S., *Evaluating system integrity*, in: *Proceedings of the ACM New Security Paradigms Workshop*, 1998.

[13] Foley, S., *A non-functional approach to system integrity.*, IEEE Journal on Selected Areas in Commications (2003), *forthcoming*.

[14] Freuder, E. C. and R. J. Wallace, *Partial constraint satisfaction*, AI Journal **58** (1992), pp. 21–70.

[15] Laprie, J., *Dependability: Basic concepts and terminology*.

[16] Lowe, G., *An Attack on the Needham-Schroeder Public-Key Authentication Protocol*, Information Processing Letters **56** (1995), pp. 131–133.

[17] Ruttkay, Z., *Fuzzy Constraint Satisfaction*, in: *Proc. of 3rd IEEE International Conference on Fuzzy Systems*, 1994, pp. 1263–1268.

[18] Schiex, T., *Possibilistic Constraint Satisfaction Problems, or "How to Handle Soft Constraints?"*, in: *Proc. of 8th Conference on Uncertainty in AI*, 1992, pp. 269–275.

[19] Schiex, T., H. Fargier and G. Verfaille, *Valued Constraint Satisfaction Problems: Hard and Easy Problems*, in: *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)* (1995), pp. 631–637.

[20] United States General Accounting Office, Accounting and Information Management Division, "Financial Audit Manual," (1996), gAO/AFMD-12.19.5A.