Soft Constraints for Security Protocol Analysis: Confidentiality

Giampaolo Bella[1,2] and Stefano Bistarelli[3]

[1] University of Cambridge, Computer Laboratory,
Pembroke Street, Cambridge CB2 3QG, England.
giampaolo.bella@cl.cam.ac.uk

[2] Università di Catania, Dipartimento di Matematica e Informatica,
Viale A. Doria 6, I-95125 Catania, Italy.
giamp@dmi.unict.it

[3] Università di Pisa, Dipartimento di Informatica,
Corso Italia 40, I-56125 Pisa, Italy.
bista@di.unipi.it

Abstract. We model any network configuration arising from the execution of a security protocol as a soft constraint satisfaction problem (SCSP). We formalise the protocol goal of *confidentiality* as a property of the solution for an SCSP, hence confidentiality always holds with a certain *security level*. The *policy* SCSP models the network configuration where all admissible protocol sessions have terminated successfully, and an *imputable* SCSP models a given network configuration. Comparing the solutions of these two problems elicits whether the given configuration hides a *confidentiality attack*. We can also compare attacks and decide which is the most significant. The approach is demonstrated on the asymmetric Needham-Schroeder protocol.

1 Introduction

Modern computer networks are *insecure* in the sense that the ongoing traffic can be intercepted and eavesdropped by a malicious attacker, called *spy* below. Agents trying to communicate over an insecure network execute suitable *security protocols* to take advantage of the protocol *goals*. A major goal is *confidentiality*, which holds of a message that remains undisclosed to the spy. Failure to achieve the claimed goals of a protocol [AN96,Low96,LR97] has motivated a number of approaches to reasoning formally on security protocols (e.g. [Low95,BR97,Pau98,Bel99]).

Our original contribution to formal protocol analysis is an approach to modelling any network configuration arising from the execution of a protocol as a soft constraint satisfaction problem (SCSP), and to detecting confidentiality attacks mounted by the spy *in the given configuration*. Also, we can establish which is the more significant out of a pair of attacks.

Recall that an SCSP may be viewed as a classical constraint satisfaction problem (CSP) [Mac92,Wal96] where each assignment of values to variables in the constraints is associated to an element taken from a partially ordered set. These elements can then be interpreted as levels of preference, or of certainty, etc. When modelling security protocols, the partially ordered set contains *security levels*. For example, the Kerberos protocol [BP98] relies on two separate sets of session keys — one contains the "authorisation keys", and the other contains the "service keys". Each authorisation key is used to encrypt several service keys. In consequence, should the spy get hold of an authorisation key, by mere decryption she would also discover all the associated service keys. In fact, different keys (and, in general, different messages) must be associated with different security levels, and there are confidentiality attacks of different significance. We formally develop this reasoning using the soft constraint framework, whereas, to our knowledge, confidentiality is a mere yes/no property in the existing literature.

We demonstrate our approach on the *asymmetric* Needham-Schroeder protocol [NS78]. This is one of the the most discussed security protocols for it hides a subtle but realistic attack that was discovered nearly two decades after the protocol publication [Low95]. We assume a basic familiarity with the concepts of encryption and decryption [QUI77,RSA76]. Encryption will be indicated by fat braces, so that $\{\!|m|\!\}_K$ will stand for the ciphertext obtained by encrypting message m under key K. Encryption is *perfect* when m can be recovered from $\{\!|m|\!\}_K$ if and only if K^{-1} is available. In this definition, the keys K and K^{-1} are interchangeable. A major feature is that K cannot be obtained from K^{-1} or vice versa. Encryption is often not perfect when it is implemented, but Lowe's attack shows that designing a protocol that achieves its claimed goals is not trivial even if perfect cryptography were available.

Below, we briefly review the basics of semiring-based SCSPs (§2), and then describe how to use them to model attacks to security protocols (§3). Then, we describe the asymmetric Needham-Schroeder protocol (§4), use our approach on that protocol (§5), and conclude (§6).

2 Soft constraints

Several formalisations of the concept of *soft constraints* are currently available [SFV95,DFP93,FW92,FL93]. In the following, we refer to one that is based on semirings [BMR95,BMR97,Bis01], which can be shown to generalise and express many of the others [BFM$^+$96,BFM$^+$99].

Let us first remind that a CSP is a tuple $\langle V, D, C, con, def, a\rangle$ where

- V is a finite set of variables, i.e., $V = \{v_1, \ldots, v_n\}$;
- D is a set of values, called the domain;
- C is a finite set of constraints, i.e., $C = \{c_1, \ldots, c_m\}$. C is ranked, i.e. $C = \bigcup_k C_k$, such that $c \in C_k$ if c involves k variables;
- con is called the connection function and it is such that $con : \bigcup_k (C_k \to V^k)$, where $con(c) = \langle v_1, \ldots, v_k\rangle$ is the tuple of variables involved in $c \in C_k$;

– def is called the definition function and it is such that $def : \bigcup_k (C_k \to \wp(D^k))$, where $\wp(D^k)$ is the powerset of D^k, that is, all the possible subsets of k-tuple in D^k;
– $a \subseteq V$, and represents the *distinguished* variables of the problem.

In words, function *con* describes which variables are involved in which constraint, while function def specifies which are the domain tuples permitted by the constraint. The set a is used to point out the variables of interest in the given CSP, i.e., the variables for which we want to know the possible assignments, compatibly with all the constraints (note that other classical definitions of constraint problems do not have the notion of distinguished variables, and thus it is as if all variables are of interest).

An example of CSP is depicted in figure 1, where variables are inside circles, constraints are represented by undirected arcs. Here we assume that the domain $\mathcal{D}$ of the variables contains only elements a and b.

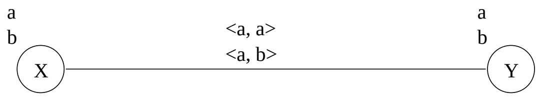

Fig. 1. A CSP

To transform a classical constraint into a soft one, we need to associate to each instantiation of its variables a value from a partially ordered set. Combining constraints will then have to take into account such additional values, and thus the formalism has also to provide suitable operations for combination ($\times$) and comparison ($+$) of tuples of values and constraints. This is why this formalisation is based on the concept of semiring, which is just a set plus two operations.

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 is a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that: $+$ is idempotent, $\mathbf{1}$ is its absorbing element and $\times$ is commutative.

Let us consider the relation $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$. Then it is possible to prove that (see [BMR97]):

– $\leq_S$ is a partial order;
– $+$ and $\times$ are monotone on $\leq_S$;
– $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum;
– $\langle A, \leq_S \rangle$ is a complete lattice and, for all $a, b \in A$, $a + b = lub(a, b)$.

Informally, the relation $\leq_S$ gives us a way to compare (some of the) tuples of values and constraints. In fact, when we have $a \leq_S b$, we will say that *b is better than a*. Below, $\leq$ will usually replace $\leq_S$.

A *constraint system* is a tuple $CS = \langle \mathcal{S}, \mathcal{D}, \mathcal{V} \rangle$ where $\mathcal{S}$ is a c-semiring, $\mathcal{D}$ is a finite set (the domain of the variables) and $\mathcal{V}$ is an ordered set of variables.

Given a semiring $\mathcal{S} = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and a constraint system $CS = \langle \mathcal{S}, \mathcal{D}, \mathcal{V} \rangle$, a *constraint* is a pair $\langle def, con \rangle$ where $con \subseteq \mathcal{V}$ and $def : \mathcal{D}^{|con|} \rightarrow A$. Therefore, a constraint specifies a set of variables (the ones in *con*), and assigns to each tuple of values of these variables an element of the semiring.

A soft *constraint problem* is a pair $\langle C, con \rangle$ where $con \subseteq \mathcal{V}$ 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*.

Figure 2 pictures a soft CSP, with the semiring values written to the right of the corresponding tuples, obtained from the classical one represented in figure 1 by using the fuzzy c-semiring [DFP93,Rut94,Sch92]:

$$S_{FCSP} = \langle [0,1], max, min, 0, 1 \rangle.$$

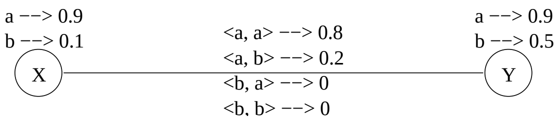

Fig. 2. A fuzzy CSP

Combining and projecting soft constraints Given two constraints $c_1 = \langle def_1, con_1 \rangle$ and $c_2 = \langle def_2, con_2 \rangle$, their *combination* $c_1 \otimes c_2$ is the constraint $\langle def, con \rangle$ defined by $con = con_1 \cup con_2$ and $def(t) = def_1(t \downarrow^{con}_{con_1}) \times def_2(t \downarrow^{con}_{con_2})$, where $t \downarrow^X_Y$ denotes the tuple of values over the variables in Y, obtained by projecting tuple t from X to Y. In words, combining two constraints means building a new constraint involving all the variables of the original ones, and associating to each tuple of domain values for such variables a semiring element that is obtained by multiplying the elements associated by the original constraints to the appropriate subtuples.

Given a constraint $c = \langle def, con \rangle$ and a subset I of $\mathcal{V}$, the *projection* of c over I, written $c \Downarrow_I$ is the constraint $\langle def', con' \rangle$ where $con' = con \cap I$ and $def'(t') = \sum_{t / t \downarrow^{con}_{I \cap con} = t'} def(t)$. Informally, projecting means eliminating some variables. 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 operation.

Solutions The *solution* of an SCSP problem $\mathsf{P} = \langle C, con \rangle$ is the constraint $Sol(\mathsf{P}) = (\bigotimes C) \Downarrow_{con}$. That is, we combine all constraints, and then project over the variables in *con*. In this way we get the constraint over *con* which is "induced" by the entire SCSP.

For example, each solution of the fuzzy CSP of figure 2 consists of a pair of domain values (that is, a domain value for each of the two variables) and an associated semiring element. Such an element is obtained by looking at the smallest value for all the subtuples (as many as the constraints) forming the pair. For example, for tuple $\langle a, a \rangle$ (that is, $x = y = a$), we have to compute the minimum between 0.9 (which is the value for $x = a$), 0.8 (which is the value for $\langle x = a, y = a \rangle$) and 0.9 (which is the value for $y = a$). Hence, the resulting value for this tuple is 0.8.

3 Using SCSPs for Protocol Analysis

We explain here how to formalise any network configuration arising from the execution of a protocol as an SCSP, and define confidentiality and confidentiality attacks as properties of the solution for that SCSP. The following, general treatment is demonstrated in §5.

3.1 The Security Semiring

We define the set L as to contain *unknown*, *private*, *debug* and *public*. Each of these elements represents a possible *security level* that a protocol associates to messages. The security levels regulate the agents' knowledge of messages. They may be seen as an extension of the two-valued property of known/unknown. For example, *unknown* will be assigned to those messages that a protocol does not encompass, and obviously to those messages that a given agent does not know; *debug* will be assigned to the messages that are created and then exchanged during a protocol session. The remaining levels are self-explanatory.

Figure 3 defines a multiplicative operator, $\times_{sec}$, and an additive one, $+_{sec}$. Theorem 1 introduces the *security semiring*.

Theorem 1 (Security Semiring).
$\mathcal{S}_{sec} = \langle L, +_{sec}, \times_{sec}, public, unknown \rangle$ is a c-semiring.

Proof. There exists an isomorphism between the fuzzy c-semiring (§2) and $\mathcal{S}_{sec}$: the security levels can be mapped into the values in the range $[0, 1]$ (unknown being mapped into 1, public being mapped into 0), $+_{sec}$ can be mapped into function max, and $\times_{sec}$ into function min.

Since $\times_{sec}$ is idempotent, it is also the *glb* operator in the total order of L. While the current four levels will suffice to model most protocols, it is understood that more complex protocols may require additional ones, such as *notice*, *warning*, *error*, *crit*. The security semiring can be easily extended by upgrading the definitions of $\times_{sec}$ and $+_{sec}$.

$\times_{sec}$	*unknown*	*private*	*debug*	*public*
unknown	*unknown*	*private*	*debug*	*public*
private	*private*	*private*	*debug*	*public*
debug	*debug*	*debug*	*debug*	*public*
public	*public*	*public*	*public*	*public*

$+_{sec}$	*unknown*	*private*	*debug*	*public*
unknown	*unknown*	*unknown*	*unknown*	*unknown*
private	*unknown*	*private*	*private*	*private*
debug	*unknown*	*private*	*debug*	*debug*
public	*unknown*	*private*	*debug*	*public*

Fig. 3. The definitions of $\times_{sec}$ and $+_{sec}$

3.2 The Network Constraint System

A computer network can be modelled as a constraint problem over a constraint system $CS_n = \langle \mathcal{S}_{sec}, \mathcal{D}, \mathcal{V} \rangle$ defined as follows:

- $\mathcal{S}_{sec}$ is the security semiring (§3.1);
- $\mathcal{V}$ is an unlimited set of variables, including SPY, each representing a network agent;
- $\mathcal{D}$ is an unlimited set of values including the empty message $\{\!|\,|\!\}$, all atomic messages, as well as all messages recursively obtained by concatenation and encryption. Intuitively, $\mathcal{D}$ represents all agents' possible knowledge.

We consider an unlimited number of atomic messages, which typically are agent names, timestamps, nonces and cryptographic keys. Concatenation and encryption operations can be applied an unlimited number of times. Also, each agent can initiate an unlimited number of protocol sessions with any other agent.

We name CS_n as *network constraint system*. Note that CS_n does not depend on any protocols, for it merely portrays the topology of a computer network on which any protocol can be implemented. Members of $\mathcal{V}$ will be indicated by capital letters, while members of $\mathcal{D}$ will be in small letters.

3.3 The Initial SCSP

Each security protocol $\mathcal{P}$ is associated with a policy that should, at least, state which messages are public, and which messages are private for which agents.

It is intuitive to capture these policy rules by means of our security levels (§3.1). Precisely, these rules can be translated into unary constraints. For each agent $A \in \mathcal{V}$, we define a unary constraint that states the security levels of A's knowledge as follows. It associates security level *public* to all agent names and to timestamps (if $\mathcal{P}$ uses them); level *private* to A's initial secrets,[1] such as keys (i.e.

[1] As opposed to the secrets created during the protocol execution.

Ka if $\mathcal{P}$ uses symmetric encryption, or Ka^{-1} if it uses asymmetric encryption) or nonces; level *unknown* to all remaining domain values (including, e.g., other agents' initial secrets, or A's secrets created during the protocol execution).

This procedure defines what we name *initial SCSP for $\mathcal{P}$*, which portrays a network where $\mathcal{P}$ can be executed but none of its sessions has started yet. The initial SCSP for $\mathcal{P}$ also highlights the agents' initial knowledge, which typically consists of atomic messages such as agent names, timestamps, some keys and some nonces.

Considerations on how official protocol specifications often fail to provide a satisfactory policy [BMPT00] exceed the scope of this paper. Nevertheless, having to define the initial SCSP for a protocol may pinpoint unknown deficiencies or ambiguities in the policy.

3.4 The Policy SCSP

The policy for a protocol $\mathcal{P}$ also establishes which messages must be exchanged during a session between a given pair of agents.

We extend the initial SCSP for $\mathcal{P}$ (§3.3) with further constraints. Each step of a session of $\mathcal{P}$ between any pair of agents can be translated into, at most, two constraints. Precisely, for each protocol step whereby A sends a message m to B, the following rules must be followed.

R_1) If A invents a new secret n (i.e. typically a new nonce) and uses it to build m, then add a unary constraint on A that assigns security level *debug* to n, and level *unknown* to all remaining messages.

R_2) Add a binary constraint between A and B that assigns security level *debug* to the tuple $\langle \{\!|\,|\!\}, m \rangle$, and level *unknown* to all other possible tuples.

The unary constraint advanced by R_1 corresponds to A's off-line creation of m, while the binary constraint stated by R_2 corresponds to A's sending m to B, and B's receiving m. The two rules yield what we name *policy SCSP for $\mathcal{P}$*. This SCSP formalises a network where each agent has successfully terminated an unlimited number of protocol sessions with every other agent, while the spy has performed no malicious activity.

3.5 The Imputable SCSP

A finite network history induced by a protocol $\mathcal{P}$ may be viewed as a repeated sequence of three steps in various order: agents' creating, sending and receiving messages. However, in the real world, not all messages that are sent are then received by the intended recipient or received at all. This is due to the malicious activity of the spy. Hence, to model the configuration of the network at a certain point in a possible history as an SCSP, we need a variation of rule R_2 (§3.4) in order to allow for the malicious activity of the spy. So, for each protocol step whereby A sends a message m to B, we constrain the initial SCSP for $\mathcal{P}$ (§3.3) additionally, as stated by rule R_1 and by the following variation of rule R_2.

R_2') If C receives m, then add a binary constraint between A and C that assigns security level *debug* to the tuple $\langle \{\!|\,|\!\}, m \rangle$, and level *unknown* to all other possible tuples.

Note that C could either be B (in case the spy as not interfered) or be the spy (no other agent can act maliciously). In particular, should the spy, in the latter case, also deliver m to B, the rule quoted above would apply to the triple SPY, m and B. Or, should the spy tamper with m obtaining m', and then deliver it to another agent D, then the rule would apply to the triple SPY, m', and D.

We name the originated soft constraint problem as *imputable SCSP* for the network of agents running protocol $\mathcal{P}$. There exist an unlimited number of imputable SCSPs for $\mathcal{P}$, which, in particular, include both the initial SCSP for $\mathcal{P}$ and the policy SCSP for $\mathcal{P}$.

3.6 Agents' knowledge as a Security Entailment

The agents' knowledge increases dynamically while they are running a protocol. This can be represented using a language like *cc* [Sar89] suitably extended to deal with soft constraints [Bis01]. The extension relies on the notion of α-consistency, rather than on the notion of consistency/inconsistency. We extend the entailment relation "$\vdash$" [Sco82], which captures from the store the constraints implied but not explicitly present, with four new rules. These model the operations — message encryption, decryption, concatenation and splitting — that the agents may perform on the messages that they see, hence increasing their knowledge. We name the obtained relation *security entailment*. Below, function *def* is associated to a generic constraint over a generic agent A.

Encryption

$$def(m_1) = v_1, \quad def(m_2) = v_2, \quad def(\{\!|m_1|\!\}_{m_2}) = v$$
$$\vdash \; def(\{\!|m_1|\!\}_{m_2}) = (v \times_{sec} (v_1 +_{sec} v_2))$$

Decryption

$$def(m_1) = v_1, \quad def(m_2) = v_2, \quad def(\{\!|m_1|\!\}_{m_2}) = v$$
$$\vdash \; def(m_1) = (v_1 \times_{sec} v), \quad def(m_2) = (v_2 \times_{sec} v)$$

Concatenation

$$def(m_1) = v_1, \quad def(m_2) = v_2, \quad def(\{\!|m_1, m_2|\!\}) = v$$
$$\vdash \; def(\{\!|m_1, m_2|\!\}) = (v \times_{sec} (v_1 +_{sec} v_2))$$

Splitting

$$def(m_1) = v_1, \quad def(m_2) = v_2, \quad def(\{\!|m_1, m_2|\!\}) = v$$
$$\vdash \; def(m_1) = (v_1 \times_{sec} v), \quad def(m_2) = (v_2 \times_{sec} v)$$

3.7 Formalising Confidentiality

In this section, l will denote a security level, and m a message. Moreover, given a security protocol, P will indicate the policy SCSP for it, and p and p' some imputable SCSPs for the same protocol. We define $Sol(\mathsf{P}) \Downarrow_{\{SPY\}} = \langle con_{\mathsf{P}}, def_{\mathsf{P}} \rangle$, $Sol(\mathsf{p}) \Downarrow_{\{SPY\}} = \langle con, def \rangle$, and $Sol(\mathsf{p}') \Downarrow_{\{SPY\}} = \langle con', def' \rangle$.

Definition 1 (l-Confidentiality).
l-confidentiality of m in $\mathsf{p} \iff \mathit{def}(m) > l$.

If l-confidentiality of m in p holds, then we say that m is *l-confidential* in p. Intuitively, this signifies that the spy does not know message m in the network configuration given by p with security level l but, rather, with a level that is better than l.

The definition becomes spurious if $\mathit{def}(m) = \mathit{public}$, in which case we say that *m has worst confidentiality in* p, or if $l = \mathit{unknown}$. Conversely, if $\mathit{def}(m) = \mathit{unknown}$, we say that *$m$ has best confidentiality in* p, as l-confidentiality holds for all $l < \mathit{unknown}$. If $l = \mathit{public}$, then m is certainly l-confidential in p unless $\mathit{def}(m) = \mathit{public}$.

Definition 2 (Confidentiality Attack).
Confidentiality attack on m in $\mathsf{p} \iff \mathit{def}(m) < \mathit{def}_{\mathsf{P}}(m)$

If confidentiality attack on m in p holds, then we say that *there is* a confidentiality attack on m in p. Intuitively, this signifies that the spy has lowered in p her security level for m w.r.t. that allowed by the protocol policy. Clearly, if there is a confidentiality attack on m in p such that $\mathit{def}(m) = l$, then m is not l-confidential in p.

Confidentiality attacks can be compared as follows. If there is a confidentiality attack on m in p such that $\mathit{def}(m) = l$, and a confidentiality attack on m in p' such that $\mathit{def}'(m) = l'$ and $l < l'$, then we say that p *hides a worse confidentiality attack on m than* p' *does.*

4 The Needham-Schroeder Protocol

We present the "asymmetric" protocol due to Needham and Schroeder, which is based on asymmetric cryptography (e.g. RSA [RSA76]) rather than on symmetric cryptography (e.g. DES [QUI77]). Each agent A is endowed with a *public* key Ka, which is known to all, and a *private* key Ka^{-1}, which should be known only to A. Note that limiting the knowledge of Ka^{-1} only to A is an assumption explicitly required by the protocol, rather than a property that is enforced.

Recall that a *nonce* is a "number that is used only once" [NS78]. The protocol assumes that agents can invent *truly-random* nonces, so that, given a nonce N invented by an agent P, the probability that agents other than P guess N is negligible.

The first step sees an *initiator* A initiate the protocol with a *responder* B. A invents a nonce Na and encrypts it along with her identity under B's public key. Upon reception of that message, B decrypts it and extracts A's nonce. Then, he invents a nonce Nb and encrypts it along with Na under A's public key. When A receives message 2, she extract Nb and sends it back to B, encrypted under his public key.

The goal of the protocol is *authentication*: at completion of a protocol session initiated by A with B, A should get evidence to have communicated with B and,

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

Fig. 4. The asymmetric Needham-Schroeder protocol

likewise, B should get evidence to have communicated with A. We emphasise that authentication here is achieved by means of confidentiality of the nonces. Indeed, upon reception of Na inside message 2, A would conclude that she is interacting with B, the only agent who could retrieve Na from message 1, since Na is a truly-random nonce and encryption is perfect. 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. But, what happens if the spy intercepts some messages?

4.1 Lowe's attack to the Needham-Schroeder Protocol

Recall that security protocols are implemented as distributed concurrent programs. Lowe discovers [Low95] that the Needham-Schroeder protocol allows the scenario depicted in figure 5, whereby a malicious agent C can interleave two of the protocol sessions, provided that some agent A initiates one session with C.

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

Fig. 5. Lowe's attack to the Needham-Schroeder Protocol

Note that C could be a registered user of the network, so no-one could suspect his tampering. Since A initiates with C, she encrypts her nonce and her identity under C's public key. Once obtained these data, C initiates another session (indicated by the primes) with another agent B, quoting A's data rather than his own. From this message, B deduces that A is trying to communicate with him. Therefore, B replies to A, quoting her nonce and his own, Nb. Since the entire network is under C's control, C intercepts this message before it is delivered to A but cannot decrypt it because encryption is perfect. So, C forwards it to A. The message is of the form that A was expecting, hence A extracts Nb and sends it to the agent with whom she had initiated the first session, C. This hinders the confidentiality of Nb, so C can use it to complete the session with B by issuing message 3', which is of the form that B was expecting.

As a result, B believes to have communicated with A, while A was in fact communicating with C. In other words, C impersonates A with B: the protocol has ultimately failed achieve authentication because it has failed to keep Nb confidential. The consequences of the attack are that B will consider every future messages quoting Nb as coming from A. If B is a bank and A and C are account holders, C could ask B to transfer money from A's account to C's without A realising it.

Lowe proposes to quote B's identity in message 2, so that A would address message 3 to the agent mentioned in that message rather than to the one with whom she initiated the session. This upgrade would prevent the attack in figure 5 because message 3 would be encrypted under B's public key, so C would not discover Nb and could not construct message $3'$ to complete the session with B.

5 Modelling Needham-Schroeder

We demonstrate our approach to protocol analysis on the Needham-Schroeder protocol. Note that all SCSPs presented below are in fact suitable fragments. As a start, we build the initial SCSP (figure 6) and the policy SCSP for the protocol (omitted here).

Fig. 6. Initial SCSP for the Needham-Schroeder protocol

Then, we build the imputable SCSP corresponding to the network configuration in which a protocol session between A and B has initiated and terminated without the spy's interference (figure 7). We observe that our definition of confidentiality attack does not hold in this imputable SCSP. This confirms that, if the spy does not interfere with the completion of a session between A and B, then she does not acquire more knowledge than that allowed by the policy. In particular, the nonce Nb has best confidentiality in this SCSP.

Hence, we build the imputable SCSP corresponding to Lowe's attack (figure 8). We add a suitable suffix to B's nonces to distinguish which agent they are meant to be used with. There is a confidentiality attack on nonce Nb_a: by the security entailment, the spy has lowered her security level for Nb_a from

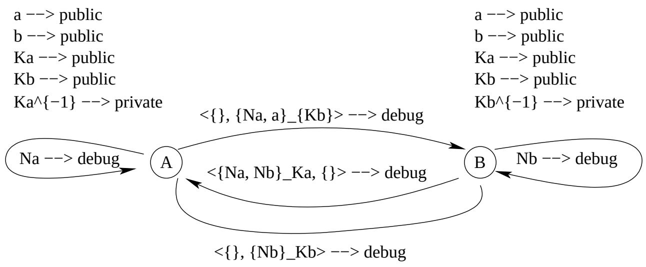

Fig. 7. Imputable SCSP for completion of protocol session between A and B

unknown, which was stated by the policy SCSP, to *debug*. Indeed, Nb_a is not even *debug*-confidential in this SCSP, but only *public*-confidential.

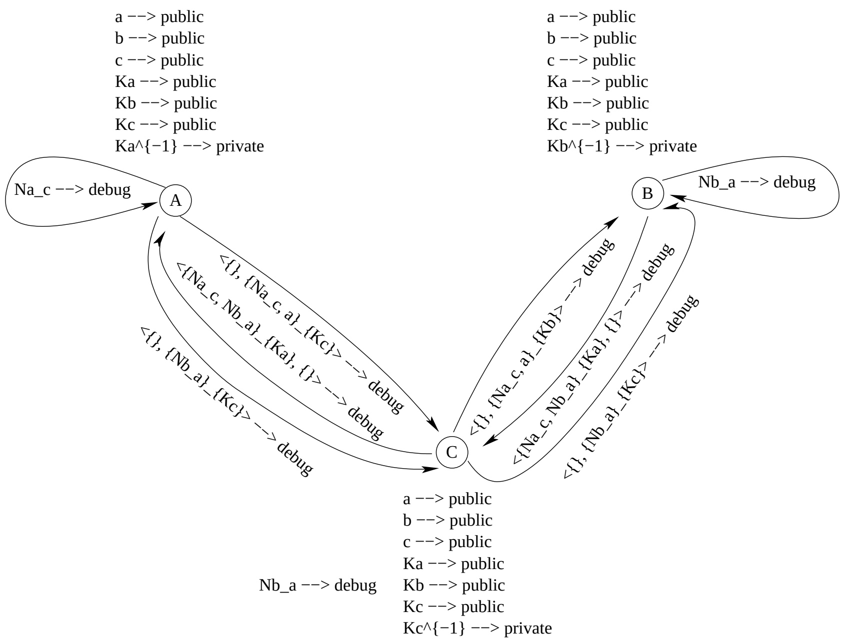

Fig. 8. Imputable SCSP corresponding to Lowe's attack

It is easy to verify that, if the protocol is amended as required by Lowe (§4.1), then the imputable SCSP corresponding to Lowe's attack is not generated by our procedure for building the imputable SCSPs (§3.5).

6 Conclusions and future work

A number of approaches for reasoning formally about security protocols are available (e.g. [Low95,Pau98,Bel99,BR97]). We have developed a new approach bases on soft constraints where confidentiality is not merely associated to a boolean value but to a discrete security level. This allows a finer reasoning on the confidentiality goals that one expects from a protocol. For example, let us consider the network configuration in which a message that is meant for a pair of agents becomes unexpectedly known to a third agent (i.e. *debug*). Some protocol policy might address this as a confidentiality attack, while some other might not unless the message becomes known to all (i.e. *public*). Yet another policy might consider as a confidentiality attack the mere fact that the spy has some private information of her own. We formally treat those different levels of confidentiality, and argue that this is novel to the field of protocol verification.

Our approach is not currently devoted to proving protocols attack-proof but, rather, to deciding whether a single configuration induced by a protocol is so. If the configuration is found to hide an attack, then our analysis also states its significance. Therefore, our approach might be used to complement the existing semi-automated ones based on model checking or theorem proving. Should these discover some crucial configuration, building the corresponding imputable SCSP would allow a deeper reasoning on that configuration by comparison with the policy SCSP. We expect this reasoning to be useful to the resolution of legal disputes where a configuration induced by a protocol is imputed as possibly illegal, and the judge must establish whether that configuration is admissible by the policy that comes with the protocol.

Our protocol analyses shall be mechanised. We intend to implement a tool that takes as input a protocol specification, a protocol policy, and a configuration to study. The tool should build the policy SCSP for the protocol, the imputable SCSP for the given configuration, compare their solutions, and finally output "OK" or a statement of the confidentiality attacks mounted in that configuration with what security levels. Although computing the solution of an SCSP is in general NP-complete, computing those two solutions will be efficient in practice, once a reasonable bound is stated on the number of agents and on the number of protocol sessions that each agent is entitled to initiate.

Another direction of research is developing a dynamic evolution of the initial SCSP by means of agents who *tell* [Sar89] new constraints, so to model the evolution of the network. All network configurations could now be checked against the policy, but, to do so, the solution should be computed of all possible imputable SCSPs. This might raise the computational costs. However, since each imputable SCSP differs from the previous one for at most two constraints (§3.5), previous computation could be reused and complexity could be managed.

Future research also includes formalising additional protocol goals such as agent authentication, message authenticity and session key distribution. Our findings, together with the research directions sketched above, support soft constraint programming as a significant and highly promising approach to protocol verification.

Acknowledgements. We are grateful to the anonymous referees for their useful comments on a draft of this paper. Giampaolo Bella acknowledges support from EPSRC grant code GR/R01156/01 "Verifying Electronic Commerce Protocols". Stefano Bistarelli acknowledge support from the MURST Project "Tipi, Ordine Superiore e Concorrenza (TOSCA)".

References

[AN96] M. Abadi and R. M. Needham. Prudent Engineering Practice for Cryptographic Protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.

[Bel99] G. Bella. Modelling Security Protocols Based on Smart Cards. In *Proc. of the International Workshop on Cryptographic Techniques & E-Commerce (CrypTEC'99)*, pages 139–146. City University of Hong Kong, 1999.

[BFM+96] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and Valued CSPs: Basic Properties and Comparison. In *Over-Constrained Systems*. Springer-Verlag, 1996.

[BFM+99] S. Bistarelli, 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(3), 1999.

[Bis01] S. Bistarelli. *Soft Constraint Solving and programming: a general framework.* PhD thesis, Dipartimento di Informatica, 2001.

[BMPT00] G. Bella, F. Massacci, L. C. Paulson, and P. Tramontano. Formal Verification of Cardholder Registration in SET. In *Proc. of European Symposium on Research in Computer Security (ESORICS 2000)*, LNCS. Springer-Verlag, 2000. In press.

[BMR95] S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*. Morgan Kaufman, 1995.

[BMR97] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, pages 201–236, 1997.

[BP98] G. Bella and L. C. Paulson. Kerberos Version IV: Inductive Analysis of the Secrecy Goals. In *Proc. of European Symposium on Research in Computer Security (ESORICS'98)*, volume 1485 of *LNCS*, pages 361–375. Springer-Verlag, 1998.

[BR97] G. Bella and E. Riccobene. Formal Analysis of the Kerberos Authentication System. *Journal of Universal Computer Science*, 3(12):1337–1381, 1997.

[DFP93] D. Dubois, H. Fargier, and H. Prade. The Calculus of Fuzzy Restrictions as a Basis for Flexible Constraint Satisfaction. In *Proc. of IEEE International Conference on Fuzzy Systems*, pages 1131–1136. IEEE Press, 1993.

[FL93] H. Fargier 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)*, pages 97–104. Springer-Verlag, 1993.

[FW92] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *AI Journal*, 1992.

[Low95] G. Lowe. An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[Low96] G. Lowe. Some New Attacks upon Security Protocols. In *In Proc. of Computer Security Foundations Workshop (CSFW96)*, pages 139–146. IEEE Press, 1996.

[LR97] G. Lowe and B. Roscoe. Using CSP to Detect Errors in the TMN Protocol. *IEEE Transactions on Software Engineering*, 3(10), 1997.

[Mac92] A. K. Mackworth. Constraint Satisfaction. In *Encyclopedia of AI (second edition)*, pages 285–293. John Wiley & Sons, 1992.

[NS78] R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978.

[Pau98] L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6:85–128, 1998.

[QUI77] CHE AUTORE METTIAMO QUI? *Data Encryption Standard*. National Bureau of Standards, Jan 1977. Federal Information Processing Standards Publications, FIPS Pub. 46.

[RSA76] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1976.

[Rut94] Zs. Ruttkay. Fuzzy Constraint Satisfaction. In *Proc. of 3rd IEEE International Conference on Fuzzy Systems*, pages 1263–1268, 1994.

[Sar89] V.A. Saraswat. *Concurrent constraint programming language*. PhD thesis, Carnegie-Mellon University, 1989.

[Sch92] T. Schiex. Possibilistic Constraint Satisfaction Problems, or "How to Handle Soft Constraints?". In *Proc. of 8th Conference on Uncertainty in AI*, pages 269–275, 1992.

[Sco82] D. S. Scott. Domains for denotational semantics. In *Proc. ICALP*. Springer-Verlag, 1982.

[SFV95] T. Schiex, 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)*, pages 631–637. Morgan Kaufmann, 1995.

[Wal96] M. Wallace. Practical Applications of Constraint Programming. *Constraints: An International Journal*, 1996.