Formal Modelling of Cryptographic Protocols in Situation Calculus*

José Hernández†

Universitat de València
Dpt. de Lògica
Aptat. 22109, E-46071 València, Spain
E-mail: Jose.Hernandez@uv.es

Javier Pinto

Pontificia Universidad Católica de Chile
Escuela de Ingeniería, Dpto. de Ciencia de Computación
Casilla 306, Santiago 22, Chile
E-mail: jpinto@ing.puc.cl

Draft
March 20, 1997

Abstract

We present here a generic model in Situation Calculus (S.C.) to formalise several cryptographic *protocols*. It distinguishes from other logics of cryptographic modelling using the concepts of believing or secret because it is based on the concepts of knowledge and it will not be required any special construct out of S.C. As a working example, we take the ISO/IEC 9798-2 5.1.2 protocol, which we specify formally. Subsequently we prove some correction properties of it.

Expecting the range of applications and a possible semi-automatisation of proofs, maybe the most promising result till now is that the protocol specification remains in an extremely plain and explicit manner, *justifying per se the formalisation*.

In the end, we comment a *set of requirements* that are necessary to the general goal of fully formalising cryptographic protocols.

Key words: Cryptography, Situation Calculus, Security Protocols, Knowledge Modelling, ISO/IEC 9798-2 5.1.2. protocol

1 Introduction

We present here a generic model in Situation Calculus (S.C.) to formalise several cryptographic *protocols*[1]. To attain it, we start from the indexical knowledge model introduced by Scherl and Levesque [17] and elaborated in following articles [18]. Then we add the necessary

*Most of this research was established and developed during the cooperation originated by a grant from the Iberoamerican Cooperation Institute (ICI).

†Current address is Universitat Politècnica de València, Departament de Sistemes Informàtics i Computació, Camí de Vera 14 Aptat. 22.012 E-46071 València, Spain. Email: jorallo@dsic.upv.es

[1]We shall not undertake here problems with the encryption *algorithm*, such as RSA, DSS or any others, which we shall suppose infallible, although cases like [5] and studies like [11] about Timing Attacks may have called

predicates and actions to build a theory that allows to constitute a theory that will make possible to formalise cryptographic protocols. As a distinguishing remark from other logics of cryptographic modelisation, we can point out that it is based on the concepts of knowledge (and not on believing or secret) and it will not be required any special construct out of S.C.

For the cryptographic fundamentals, we shall follow the lines, notations and some classical examples from the literature, mainly [1] and [7]. From the former, we have extracted some informal principles, making them requisites of our formalisation, what may facilitate the design of protocols using it. From the latter we have taken out ideas from her extended and improved formalisation, in our view, of BAN logic [4], which, at the same time, we have adapted and extended to our needs.

As an example we introduce the ISO/IEC 9798-2 5.1.2. protocol, which we specify formally and from here we prove some correction properties, adopting some assumptions. As is commented in [7] about the same protocol, this one may bring problems if there are several runs of it; even though it is not presented here, the model we introduce allows to prove these cases adapting Gürgens's axioms.

The constructs and axioms that we present in this article are an extract of what we have considered most significant to illustrate the theory; a more thorough and detailed exposition can be found in the extended preliminary version of this text [8].

2 Example

To see the necessity of the different formalisms and constructs that will be introduced, we are going to present a simple protocol, the ISO/IEC 9798-2 5.1.2 protocol, described in [10]. We shall try to make a similar analysis of the one made by [7].

2.1. The **informal description** of a *run* of the protocol is:

0. Both principals, A and B, share a symmetric key called K_{AB}.
1. B sends to A a *nonce* (random number) N_B and, optionally, a textfield $text_1$.
2. When A receives the message, A calculates the nonce N_A and sends a text $text_3$, followed by the message $(N_A, N_B, B, text_2)$ encrypted with K_{AB}.
3. On receiving message 2, B verifies the cyphered text, decoding it and checking the correction of the identifier B that distinguishes it and that the nonce N_B sent in step 1, matches the one that is in the cyphered text.
4. B sends a text $text_5$, followed by the message $(N_B, N_A, text_4)$ encrypted with K_{AB}.
5. On the arrival of message 3, A verifies the cyphered message, decyphering it and checking that the nonce N_B received from B in message 1 matches the nonce inside the message and that the nonce N_A sent to B in message 2 matches the nonce inside the message.
6. The goal of the protocol is that, both principals —not using a key server— would be convinced that, once finished the run, they have talked to each other.

2.2. Using the **classical notation**, which clarifies quite the process, the protocol stands as follows:

1. $B \rightarrow A: \ N_B, text_1$
2. $A \rightarrow B: \ text_3, \{N_A, N_B, B, text_2\}_{K_{AB}}$
3. $B \rightarrow A: \ text_5, \{N_B, N_A, text_4\}_{K_{AB}}$

them into question. It is possible that in order to avoid these lacks, it may be necessary considering jointly the algorithm, the protocol and the code implementation of both.

The problem of this notation is that there is no indication of the checkings made by the principals.

2.3. Thereto we present an **extended notation** introducing some preconditions or checkings before taking any action. So it is:

0. **INITIAL PRECONDITIONS:**

   K_{AB} is a symmetric key shared by A and B.

1. **PRECONDITIONS of B:**

   None

   ACTIONS:

   $B \rightarrow A:\ N_B, text_1$

2. **PRECONDITIONS of A:**

   Receives a message (x_1, x_2)

   ACTIONS:

   $A \rightarrow B:\ text_3, \{N_A, N_B, B, text2\}_{K_{AB}}$

3. **PRECONDITIONS of B:**

   Receives a message (x_3, x_4)

   Decrypts x_4 with K_{AB} into a message (x_5, x_6, x_7, x_8)

   Verifies that $x_7 = B$

   Verifies that $x_6 = N_B$

   ACTIONS:

   $B \rightarrow A:\ text_5, \{N_B, N_A, text_4\}_{K_{AB}}$

4. **PRECONDITIONS of A:**

   Receives a message (x_9, x_{10})

   Decrypts x_{10} with K_{AB} into a message (x_{11}, x_{12}, x_{13})

   Verifies that $x_{11} = x_1$

   Verifies that $x_{12} = N_A$

5. **GOALS:**

   A knows that she has talked with B

   B knows that she has talked with A

6. **RESTRICTIONS OF EXECUTION:**

   A and B must not communicate K_{AB} in any other message.

   A and B must not communicate N_A and N_B in any other message but these.

   A and B must not communicate $text_2$ and $text_4$ in any other message.

With this we have completed the informal especification of an ISO 9798-2 protocol run.

Now we may ask ourselves whether the protocol carry out its function and if not, to which attacks it is vulnerable. Do we have any way of making sure when this kind of things may or not happen? The only way to be completely sure that some fact cannot happen consists in formalising the protocol and posing the question in this frame, showing that is not the case.

On that account it is necessary to be able to express the previous scheme with a formal notation. As we have already pointed out, apart from other approaches that have achieved it partially (proving some properties) we are going to make use of situation calculus. S.C. has two advantages: it will not required —as we shall see— any construct out of it and, afterwards we may essay a semi-automatisation of proofs.

To sum up, we shall have to formalise how the messages are sent and received, how they are encrypted, what kind of keys can be used, what a protocol run is, and even we shall have to introduce formally concepts that have been vaguely formalised in the literature such as *nonces* or *timestamps*.

3 Assumptions

Before starting with, it is necessary supposing some assumption within which we are going to study the protocols, many of them habitual in the cryptographical literature.

1. **Encryption Algorithm**: the presented modelisation works independently of which encryption algorithms might be used, for instance DES, RSA or others. It is not taken account of the possibility that these algorithms may be broken into by brute force after some time, since we shall suppose that keys have enough bits to render this probabilisticly impossible. In fact, we have made a simplification in the presented model, since it is supposed that there is only one encryption algorithm, so an agent can decrypt a message just knowing the key and that that message is encrypted with that key. The model could be extended easily to cover different algorithms.
2. **Protocol Interleaving**: we shall be specially aware with the problems that may appear from the fact that several *runs* of the same or different algorithms may go on at the same time. The ISO 9798-2 protocol and many others have shown failures in the case of interleaved runs of the same protocol. The main obstacle for studying these questions is that the set of possible messages and their occurences is infinite. This is one of the latent problems in [7].
3. **Communication Stream**: we shall suppose a single communication stream or channel, with possibility of message loss, duplication o erroneus arrival. Since there is no certainty about the authenticity of the origin of a message, the better way results in modeling the stream as a *broadcasted channel*, in which all principals receive and can read the messages. Besides, any principal may send the message she wants in any moment. Although this is the classical approach, we shall make out a special predicate $SecureComm(a, b)$ to represents the cases where we can sharply affirm that the communication between two principals can be made privately, without tappling. We shall use, in the same way, a predicate $ReliableComm(a, b)$ when we would want express that communication never fails; i.e., with independence from whether there have been tappling or not, a sent message implies a received message without any error.
4. **Keys and Privacity**: we shall not assume however certain common properties of public and private keys, not because we want to make a very general model, but because by the means of the fluent $Knows$, we can perfectly express which keys each agent knows and which ones does not, as well as those that are shared. For the same reason, we do not need the introduction of predicates to model secretness.
5. **Messages**: messages do not contain more information than that which is explicit (this is more or less the first principle of [1]). Therefore, any information that is supposed implicitly will not be considered in the model, so ignoring its order, run, origin, destination and whether it is encrypted or not. All of this should be known or deduced by the addressee of the message. The only thing that could be noted will be the type of the components that go packed according to their order and structure.

4 Fundamentals

4.1 The Language

Next we shall present briefly Situation Calculus (S.C.), where all the logic constructs come from. For the moment, we introduce the classes of objects which we are going to work with. We shall use lowercase letters for the variables belonging to these classes in order to avoiding using permanently the belongs-to symbol (i.e. when we indicate a variable t, it is supposed from class $\mathcal{T}$).

- $\mathcal{A}$: class of agents (also known as principals). The agents are the individuals who communicate through the channel. In our model, they match with agents as known in S.C. The lowercase letters b and c will also be used as belonging to this class.
- $\mathcal{M}$: class of messages. A message is the object that may be sent or received by an agent.
- $\mathcal{P}$: class of protocols.
- $\mathcal{R}$: class of runs. A run is the execution of a protocol from the beginning to the end. As we have said, we shall allow several runs occurring sequentially or concurrently.
- $\mathcal{E}$: class of steps. Each protocol run is composed of some steps, identified with natural numbers.
- $\mathcal{K}$: class of keys. Used to encrypt or decrypt messages. There are several kinds of keys, as we shall see.
- $\mathcal{S}$: class of situations. They are the moments or states in S.C. and here they will take the same role that time points in other formalisations.
- $\mathcal{Q}$: class of actions. They are also those from S.C. and allow moving between two situations.
- $\mathcal{N}$: class of nonces. A nonce is a random number generated by an agent and we shall suppose impossible that any other agent could ever repeat it provided it is not communicated before. They are used frequently in authentification protocols, as we shall see.
- $\mathcal{W}$: class of propositions. The are useful for the parameters type of metapredicates, like $Knows$.

To make easier the use of messages, we shall assume that $\mathcal{M}$ is a non-strict superset of any other class; this would allow to pack keys, agents, nonces, etc., directly inside the messages. It is the agent's matter to guess the type of each component when receiving a message. This would be implemented easily identifying the parts of the messages but not doing so allows considerating the errors produced by intruders changing fields of order, even though they would be of different type.

4.2 Situation Calculus and Knowledge

Due to the objectives and extension of this text, we shall not make a proper introduction to Situation Calculus (S.C.). We shall recall superficially its elements. The first one is the initial situation s0, to which it is possible to apply certain actions to arrive to other situations. The accessibility relation between situations is represented by $<$ sign (e.g. $s_1 < s_2$ means s_2 is accessible from s_1). Those predicates whose truth value may vary according to the situation are known as *fluents*, and consequently they have a last parameter of type $\mathcal{S}$.

The actions have some preconditions to be applied, noted by $Poss$. If the action is possible, the result of doing an action a in a situation s is a new situation s'. That is,

$$Poss(acc(\cdots), s) \wedge s' = do(acc(\cdots)) \supset \text{Fluents that must be true in } s'$$

Those who are not familiarised with S.C. can consult the bibliography, specially [17], since this one does a quick summary and introduces the model of knowledge based on possible worlds of Moore [14], which we shall work with. It is also introduced the possibility of indexical knowledge which may be necessary to model more complex protocols.

The concept of agent was added afterwards, following the formalisation presented in [18] which combines both into the notation $Knows(A, P(now), s)$, read as "the agent A knows P in situation s". This allows to express some very useful sentences for cryptographic modelling, like: "a_1 knows (in s_1) that a_2 knew (in s_2) something (w)":

$$Knows(a_1, Knows(a_2, w, s_2), s_1)$$

The formalisation and the resolution of the frame problem for this notation can also be found in [17] and may also be very useful when proving properties of protocols.

To ease the following task, we are extracting a reduced set of formulae from the definition of $Knows$ in [18]:

$$Knows(a, w, s) \supset w \text{ is true in situation } s \tag{1}$$

That is to say, the knowledge is infallible; an agent *knows* a subset (generally a strict subset) of all that is true in a given situation, but she never *knows* anything that is false.

In the definition of $Knows$ of [18], it is supposed the closure property, i.e., there is no knowledge whose logic consequences are not known as well. From this property we are specially interested in these axioms:

$$\begin{array}{c} Knows(a, p_1 \vee p_2 \vee \cdots \vee p_n, s) \\ \wedge Knows(a, \neg p_1 \wedge \neg p_2 \wedge \cdots \wedge \neg p_{i-1} \wedge p_{i+1} \wedge \cdots \wedge \neg p_n, s) \\ \supset Knows(a, p_i, s) \end{array} \tag{2}$$

$$Knows(a, p, s) \wedge (p \supset q) \supset Knows(a, q, s) \tag{3}$$

To avoid a type problem when trying to know something that is not of propositional type, (e.g. messages, keys, etc.), it is useful to create a new fluent $KRef$ from $Knows$ in the following manner:

$$KRef(a, t, s) \stackrel{\text{def}}{=} (\exists x) \quad Knows(a, t = x, s) \quad \text{where } x \text{ does not appear in } t \tag{4}$$

From here, obviously,

$$Knows(a, x = y, s) \supset KRef(a, x, s) \wedge KRef(a, y, s) \tag{5}$$

Now we are adding a frame axiom which tells that we do not know anything new if we do not have received it in some message:

$$\begin{array}{c} \neg KRef(a, m', s) \wedge [(\forall m)(\forall s')(s'' > s' > s) Received(a, m, s') \supset \neg FullIn(m', m)] \\ \supset (\forall s' < s'') \neg KRef(a, m', s') \end{array} \tag{6}$$

The definition of the predicate $FullIn(m', m)$ can be found in [8] but, succintly, it means that the message m' is in m, allowing nested encryptions.

5 Actions

From the model of knowledge presented in the previous point, we introduce the following actions inside S.C.:

- $pack(a, m_1, m_2, m)$
- $unpack(a, m, m_1, m_2)$
- $encrypt(a, m_1, m_2, k)$
- $decrypt(a, m_1, m_2, k)$
- $send(a, m)$
- $receive(a, m)$
- $createNonce(a, n)$
- $verify(a, w)$

5.1 Packing

The actions *pack* and *unpack* allow us to give form to the messages and allow distinguishing the order of the components when comparing messages. Thus it is not necessary introducing the concept of *type* of messages (according to their structure).

Let's begin with the precondition and effect axioms of *pack* and *unpack*:

$$Poss(pack(a, m_1, m_2, m), s) \equiv KRef(a, m_1, s) \wedge KRef(a, m_2, s) \tag{7}$$

$$Poss(pack(a, m_1, m_2, m), s) \supset KRef(a, m, do(pack(a, m_1, m_2, m), s)) \tag{8}$$

$$Poss(unpack(a, m, m_1, m_2), s) \equiv Packed(m_1, m_2, m) \wedge KRef(a, m, s) \tag{9}$$

$$\begin{array}{c} Poss(unpack(a, m, m_1, m_2), s) \wedge s' = do(unpack(a, m, m_1, m_2), s) \\ \supset KRef(a, m_1, s') \wedge KRef(a, m_2, s') \end{array} \tag{10}$$

In the following, to shorten, we shall use the same predicate with more parameters to pack more than two messages.

5.2 Encryption

Now let's get in touch with actions *encrypt* and *decrypt*, which will be the ground, jointly with $Knows$, of our model. Both will take the same type and number of arguments:

- $encrypt(a, m, m', k)$ means "agent a encrypts m using key k resulting in the cyphered message m'".

In a similar way,

- $decrypt(a, m', m, k)$ means "agent a decrypts m' using key k resulting in the plain message m".

Firstly, we shall see the preconditions that allow to execute both actions. In order to begin encryption we require the plain information and the key which we are going to code with.

$$\begin{array}{c} Poss(encrypt(a, m, m', k), s) \\ \equiv KRef(a, m, s) \wedge KRef(a, k, s) \wedge EncodingKey(k) \end{array} \tag{11}$$

Let's remark that it is not necessary that agent a knows that k is a coding key, therefore allowing the possibility that one may be doing essays withe her set of keys.

We need something similar to decode, but an additional condition is required; decryption will only be possible in the case the information we are supplied has been encrypted previously and that the decryption key is compatible with the encryption key (i.e., they both form a key pair).

$$\begin{array}{c} Poss(decrypt(a,m',m,k),s) \\ \equiv KRef(a,m',s) \wedge KRef(a,k,s) \\ \wedge(\exists s',s''<s,a',k')do(encrypt(a',m,m',k'),s') = s'' \wedge KeyPair(k,k') \end{array} \tag{12}$$

Seen these both preconditions of encryption and decryption, the results from the encryption action are very simple: the agent gets knowledge of *what she has just done*, i.e., that m' is the result of encrypting m with key k:

$$\begin{array}{c} Poss(encrypt(a,m,m',k),s) \wedge s' = do(encrypt(a,m,m',k),s) \\ \supset Knows(a,Encrypted(m,m',k,s),s') \end{array} \tag{13}$$

And those from decryption are more apparent concerning knowledge:

$$\begin{array}{c} Poss(decrypt(a,m',m,k),s) \\ \supset (\exists s',k')s' = do(decrypt(a,m',m,k),s) \\ \wedge(\exists s''<s)Knows(a,Encrypted(m',m,k',s''),s') \wedge KRef(a,m,s') \\ \wedge Knows(a,Decrypted(m',m,k,s'),s') \wedge Knows(a,KeyPair(k,k'),s') \end{array} \tag{14}$$

That is to say, it happens that, besides knowing the plain information m (what she pretends), the agent knows —and it is possible that this was not known in s, because we allow the possibility of essays, as we have pointed out before— that the information was encrypted in m using a k' —most of cases unknown— and that it constitutes a pair with k.

5.3 Sending and Receiving Messages

We introduce straightaway the action $send(a_1,m)$ representing "agent a_1 sends message m". It should be noticed that the destination is not specified so following what we indicated in the assumption of a broadcasted channel. So we have:

$$Poss(send(a,m),s) \equiv KRef(a,m,s) \tag{15}$$

As we said, the action $receive$ is completely independent to $send$ because generally the channel is neither sure nor reliable[2], therefore $send$ has no effect besides knowing that it has been sent:

$$Poss(send(a,m),s) \wedge s' = do(send(a,m),s) \supset Knows(a,Sent(a,m,s'),s) \tag{16}$$

Action $receive(a,m)$ means "agent a receives message m" and happens in the following way:

$$\begin{array}{c} Poss(receive(a,m),s) \\ \equiv (\exists a',s',s''<s)KRef(a',m,s') \wedge s'' = do(send(a',m),s')) \end{array} \tag{17}$$

$$\begin{array}{c} Poss(receive(a,m),s) \wedge s' = do(receive(a,m),s) \\ \supset KRef(a,m,s') \wedge Knows(a,Received(a,m,s'),s') \end{array} \tag{18}$$

[2]The result in a reliable stream would be:

$$Poss(send(a_1,a_2,m),s) \supset Knows(a_2,m,do(send(a_1,a_2,m)))$$

But this definition is far from suitable because it does not allow to model the main problem which cryptographic protocols must face up to: the possibility of intruders or message errors.

5.4 Nonces

Nonces are some random marks that are necessary to identify messages in many protocols. To give an agent the possibility to create them we need a new action:

$$Poss(createNonce(a, n), s) \equiv (\forall a')\neg KRef(a', n, s) \tag{19}$$

That is, a nonce is newly created, without the possibility that the same number would have existed in other agent's knowledge (or in hers), considering the number of bits of the nonce great enough to desestimate a casual probability or brute-force hit. The action results are quite obvious:

$$Poss(createNonce(a, n), s) \supset KRef(a, n, do(createNonce(a, n), s)) \tag{20}$$

5.5 Checks

Finally we add an action $verify$ that will make possible to introduce checkings carried out by agents. Despite its importance, it is defined in a very simple way, one precondition and no action results.

$$Poss(verify(a, w), s) \equiv Knows(a, w, s) \tag{21}$$

That is, it only can be executed if it is known what is to be verified.

6 Protocol Specification

Once we have introduced the actions that agents can make, it remains to be concretised a crucial part: how do we specify protocols, which actions they are composed of, which preconditions they need, etc.

Independently of whether certain actions would be done or not, whether the agent would behave properly or not, the protocol must specify what actions and preconditions are required in every step of each protocol, for any run and agent. Hence, we shall introduce a predicate $CPPreconds(p, a, r, e)$ that expresses the preconditions of protocol p in order that a, in run r, can begin step e (a and r will be, generally, free variables).

Given the preconditions of every step, we must indicate which are the actions that can be done. This will be achieved through an action array named $CPActions$, relating preconditions and actions in the following way:

$$Poss(CPActions(p, a, r, e), s) \equiv CPPreconds(p, a, r, e) \tag{22}$$

$$Poss(CPActions(p, a, r, e), s) \supset AtStep(a, p, r, e, Do(CPActions(p, a, r, e), s) \tag{23}$$

The actions in $CPActions$ must be fulfilled sequentially. We shall use the classic notation in S.C.:

$$CPActions(p, a, r, e) \equiv [q_1; q_2; \cdots; q_n]$$

What makes the difference between an action array and a sequence of actions is that whenever an action precondition does not hold, the process is stopped and it will be assumed that the action array $CPActions$ has not tried any of them (i.e. all or naught). For that reason, we define a Do similar to GOLOG [12]:

$$\begin{gathered} Do([q_1; q_2; \cdots; q_n], s, s_n) \equiv \\ Poss(q_1, s) \wedge s_1 = do(q_1, s) \wedge Poss(q_2, s_1) \wedge s_2 = do(q_2, s_1) \wedge \cdots \\ \cdots \wedge Poss(q_n, s_{n-1}) \wedge s_n = do(q_n, s_{n-1}) \end{gathered} \tag{24}$$

Finally, we specify an auxiliar fluent $Finished$ from a new action end, which we shall use later.

$$Poss(end(a,p,r),s) \equiv (\exists e) LastStep(a,p,e) \wedge DoneStep(a,p,r,e,s) \tag{25}$$

$$Poss(end(a,p,r),s) \supset Finished(a,p,r,do(end(a,p,r),s) \tag{26}$$

Let's regard, that in order to finish the protocol we must be at he $LastStep$, which must be indicated in the protocol. The action end is not introduced in the protocol specification.[3]

7 Specification of the ISO 9798-2 Protocol

To illustrate the formalism we give a formal specification of our example, the ISO 9798-2 protocol. Firstly, in view of the preconditions, we can establish the axioms referring to the shared key:

INITIAL PRECONDITIONS:

$$Symmetric(K_{ab}, K'_{ab}) \tag{27}$$

Later we shall indicate who knows and who does not these keys, but now we shall confine to the protocol specification, which we shall call ISO9798.2, constructed from the extended notation seen in point 2.3.

[3]It also may be heeded the introduction of an action $begin$ to complement an eventual set of required preconditions to start a run, binding it to a fluent $Started$ in a similar way as how is bound end with $Finished$.

STEP 1:

$$CPPreconds(ISO9798.2, b, r, 1) \equiv (\exists s) Started(b, ISO9798.2, r, s)$$

$$CPActions(ISO9798.2, b, r, 1) = \left[\begin{array}{l} createNonce(b, N_b); \\ pack(b, N_b, text_1, m_1); \\ send(b, m_1) \end{array} \right]$$

STEP 2:

$$CPPreconds(ISO9798.2, a, r, 2) \equiv (\exists s) Received(a, m_2, s)$$

$$CPActions(ISO9798.2, a, r, 2) = \left[\begin{array}{l} unpack(a, m_2, m_3, m_4); \\ createNonce(a, N_a); \\ pack(a, N_a, m_3, b, text_2, m_5); \\ encrypt(a, m_5, m_6, K_{ab}); \\ pack(a, text_3, m_6, m_7); \\ send(a, m_7) \end{array} \right]$$

STEP 3:

$$CPPreconds(ISO9798.2, b, r, 3) \equiv (\exists s, s', s < s') DoneStep(b, ISO9798.2, r, 1, s) \wedge Received(a, m_8, s')$$

$$CPActions(ISO9798.2, b, r, 3) = \left[\begin{array}{l} unpack(b, m_8, m_9, m_{10}); \\ decrypt(b, m_{10}, m_{11}, Kab'); \\ unpack(b, m_{11}, m_{12}, m_{13}, m_{14}, m_{15}); \\ verify(b, m_{14} = b); \\ verify(b, m_{13} = N_b); \\ pack(b, N_b, N_a, text_4, m_{16}); \\ encrypt(b, m_{16}, m_{17}, K_{ab}); \\ pack(b, text_5, m_{17}, m_{18}); \\ send(b, m_{18}) \end{array} \right]$$

STEP 4:

$$CPPreconds(ISO9798.2, a, r, 4) \equiv (\exists s, s', s < s') DoneStep(a, ISO9798.2, r, 2, s) \wedge Received(a, m_{19}, s')$$

$$CPActions(ISO9798.2, a, r, 4) = \left[\begin{array}{l} unpack(a, m_{19}, m_{20}, m_{21}); \\ decrypt(a, m_{21}, m_{22}, Kab'); \\ unpack(a, m_{21}, m_{22}, m_{23}, m_{24}); \\ verify(a, m_{23} = m_3); \\ verify(a, m_{34} = N_a); \end{array} \right]$$

EXECUTION RESTRICTIONS:

$$(\forall m, s) Sent(a, m, s) \supset \neg FullIn(K_{ab}, m) \tag{28}$$

$$(\forall m, s) Sent(b, m, s) \supset \neg FullIn(K_{ab}, m) \tag{29}$$

$$(\forall m, s) Sent(a, m, s) \wedge FullIn(N_a, m) \supset m = m_7 \tag{30}$$

$$(\forall m, s) Sent(b, m, s) \wedge FullIn(N_a, m) \supset m = m_{18} \tag{31}$$

$$(\forall m, s) Sent(a, m, s) \wedge FullIn(N_b, m) \supset m = m_7 \tag{32}$$

$$(\forall m, s) Sent(b, m, s) \wedge FullIn(N_b, m) \supset m = m_1 \vee m = m_{18} \tag{33}$$

$$(\forall m, s) Sent(a, m, s) \wedge FullIn(text_2, m) \supset m = m_7 \tag{34}$$

$$(\forall m, s) Sent(b, m, s) \wedge FullIn(text_4, m) \supset m = m_{18} \tag{35}$$

Finally, we specify the last steps of the protocol:

LAST STEPS:

$$LastStep(a, ISO9798.2, 4) \tag{36}$$

$$LastStep(b, ISO9798.2, 3) \tag{37}$$

8 Protocol Instances

Let's observe that variables are in fact shared (e.g. messages and nonces). This is open to the implementation whenever it is introduced in a logic or computational system. One suggerence would be to parametrise the protocol and then specify which are its variables. In this case, the parameters are:

$a, b, Kab, Kab', text_1, text_2, text_3, text_4, text_5$

and the variables:

$m_1 \cdots m_{24}, N_a, N_b$

This may be solved in a simpler way using *parametrising* predicates in the following way:

$Par(ISO9798.2, r, a)$ would be placed instead of agent a.

$Par(ISO9798.2, r, K_{ab})$ would be placed instead of key K_{ab}.

...

And the same for variables:

$Var(ISO9798.2, r, m_1)$ would be placed instead of m_1.

$Var(ISO9798.2, r, N_a)$ would be placed instead of N_a.

...

The result would stay then a less clear specification but completely formalised in S.C., without any additional construction.

For instance, this allows to distinguish between $Var(ISO9798.2, 3, m_3)$ — the m_3 from protocol ISO9798.2 in its run 3 — from $Var(D\&S, 2, m_3)$ — another m_3 of protocol D&S in its run 2 —. This permits asking and showing properties for a protocol in general, for a concrete case of a protocol, for two parallel runs of the same protocol, for the case where two given protocols coexist, etc.

Let's see, enfin, a case of behaviour of this protocol.

8.1 Case 1

In order to use the preceding specification we must create an instance, that is, we must say who is effectively a, b, r and texts $text_1$, $text_2$, ... Choosing run 1, the rest of parameters will be as follows:

$Par(ISO9798.2, 1, a) = A$
$Par(ISO9798.2, 1, b) = B$
$Par(ISO9798.2, 1, K_{ab}) = K_{AB}$
$Par(ISO9798.2, 1, K'_{ab}) = K'_{AB}$
$Par(ISO9798.2, 1, text_1) =$"Hi man"
$Par(ISO9798.2, 1, text_2) =$"bet for horse 13"
$Par(ISO9798.2, 1, text_3) =$"I insinuate you to "
$Par(ISO9798.2, 1, text_4) =$"you can manage that its jockey had an accident..."
$Par(ISO9798.2, 1, text_5) =$"I'll do if "

Let's suppose that everything begins in an ideal way, i.e., A and B know the keys and nobody else does.

$$KRef(A, K_{AB}, s_0) \tag{38}$$

$$KRef(B, K_{AB}, s_0) \tag{39}$$

$$(\forall a) KRef(a, K_{AB}, s_0) \supset a = A \vee a = B \tag{40}$$

9 Formalisation of Properties. Goals and Security

At this moment we should be able to pose the question about whether the goal is commited, i.e.:

$$\begin{array}{c} Finished(A, ISO9798.2, r, s) \supset \\ KRef(A, m_{25}, s) \wedge KRef(A, Author(B, m_{25})) \wedge \\ KRef(B, m_{15}, s) \wedge KRef(B, Author(A, m_{15})) \end{array} \tag{41}$$

that is to say, that the two important texts had been communicated and that their source and authorship can be ensured. Whereas m_{25} should correspond to $text_4$ and m_{15} to $text_2$, we shall not prove here both correspondences because it is possible that both messages come from the authentic principal, but they may be misplaced or wrong. Obviously, the rest of texts are sent plainly and it cannot be known if someone intercepted the message and changed simply the plain texts.

Also we would like to question if it is sure, i.e. if what should be kept secret is actually kept secret:

$$\begin{array}{c} Finished(A, ISO9798.2, r, s) \supset \\ \neg(\exists a, a \neq A, a \neq B) KRef(a, K_{AB}, s) \vee KRef(a, K'_{AB}, s) \\ \vee KRef(a, text_2), s) \vee KRef(a, text_4), s) \end{array} \quad (42)$$

Ultimately, we would like to prove even more; given a started protocol run with a reliable communication and if the principals are complying to the protocol, does the run end?

$$\begin{array}{c} Compliant(A, ISO9798.2) \wedge Compliant(B, ISO9798.2) \wedge \\ Started(B, ISO9798.2, r, s) \wedge ReliableComm(A, B) \wedge ReliableComm(B, A) \\ \supset (\exists s' > s) Finished(A, ISO9798.2, r, s') \end{array} \quad (43)$$

At last, is it possible, within our formalism, to prove these questions?

10 Domain Specific Axioms

As any other theory in S.C., aside from the introduction of the preconditions for the actions and their effects, it is required further axioms to relate the predicates or fluents and at length give body to the theory. Since the whole set is very extense, we present here only those axioms that we shall use thereafter in the proof of (41). We neither give more than the necessary explanations for their comprehension[4]. For a more complete set of axioms and more detailed introduction of them, we refer to a preparatory and extended version of this work [8].

$$Packed(m_1, m_2, m) \wedge Packed(m_3, m_4, m') \wedge (m_1 \neq m_3 \vee m_2 \neq m_4) \supset m \neq m' \quad (44)$$

$$Encryptable(m'_1, m_1, k) \wedge Encryptable(m'_2, m_2, k_2) \wedge m_1 \neq m_2 \supset m'_1 \neq m'_2 \quad (45)$$

$$\neg SameType(m_1, m_2) \supset m_1 \neq m_2 \quad (46)$$

$$\begin{array}{c} [\neg(\exists(m, a, s' < s'') FullIn(m_1, m) \wedge Sent(a, m, s') \wedge \neg InsideEncryption(m_1, m, k) \ ] \\ \neg(\exists s''' < s'') KRef(a_1, m_1, s''') \wedge KeyPair(k, k') \supset \\ [(\exists s < s'') KRef(a_1, m_1, s) \supset KRef(a, k', s) \ ] \end{array} \quad (47)$$

Equation (47), though it may seem very tough, comes to express the following: "if every message sent containing m_1, did it inside an encryption with key k (i.e., every message holding m_1, holds it protected with k) and a_1 did not know m_1 before s'' then in order to a_1 could get to know m_1 before s'', necessarily a_1 had to know the key to decrypt some of the preceeding messages". Another axiom that may look uncomfortable:

$$\begin{array}{c} Received(a_3, m, s) \wedge FullIn(m', m) \wedge \\ (\exists a_1, a_2) \, [(\forall a, s', s'')(s'' < s' < s) KRef(a, m's, s') \supset (a = a_1) \vee (a = a_2) \ ] \supset \\ [(\exists s_1 < s) Sent(a_1, m_1, s_1) \wedge FullIn(m', m_1) \ ] \vee \\ [(\exists s_2 < s) Sent(a_2, m_2, s_2) \wedge FullIn(m', m_2) \ ] \end{array} \quad (48)$$

[4]This makes that the axioms may seem as introduced *ad-hoc*. Although their intuitive correspondence should be an indication that this is not the case, the necessary conviction would come from making the proofs of (41), (42), (43) and other properties of this and other protocols. The extended set of axioms could be used to classify different protocols and situations in which they may happen, according to the subset of axioms that are observed.

that means something quite obvious but formally necessary: "a message can only be sent by someone who knew its content", particularising to only two agents. Finally we shall require too some axioms to express authorship:

$$(\forall m, m', k, s, a) Encrypted(m, m', k, s) \wedge \neg KRef(a, k, s) \supset \neg Author(a, m', s) \tag{49}$$

$$[(\forall m, s) Sent(a, m, s) \supset m = m_1] \wedge \neg FullIn(m_2, m_1) \supset \neg Author(a, m_2, s) \tag{50}$$

$$Author(a, m, s) \wedge FullIn(m', m) \supset Author(a, m', s) \tag{51}$$

11 Proving Properties

As a matter of sample, we shall prove the first of proprieties exposed in point 9, which matches with the protocol goal. To ease the following proof, we shall suppose that there will not be any other protocol run.

We start from $Finished(A, ISO9798.2, r, s_f)$ and with (26) and (25) results:

$$(\exists e, s_1 < s_f, r) LastStep(A, ISO9798.2, e) \wedge DoneStep(A, ISO9798.2, r, e, s_1)$$

using (36) we have:

$$DoneStep(A, ISO9798.2, r, 4, s_1) \tag{52}$$

From (23) we know that $Do(CPActions(ISO9798.2, A, r, 4), s_2')$ took place in a $s_2' < s_1$, and from (24) we know that the actions have been accomplished one by one and that their preconditions have been satisfied. Going backwards we have that the last action is $verify(a, m_{24} = N_a)$ and as it has been carried out there must have been the case, from (21) that:

$$Knows(A, m_{24} = N_a, s_2) \text{ in } s_2 < s_1 \tag{53}$$

From (5) we have:

$$KRef(A, m_{24}, s_3) \wedge KRef(A, N_a, s_3)$$

If we go on with actions we see that if $verify(a, m_{23} = m_3)$ has been done in s_3, from (21) we get:

$$Knows(A, m_{23} = m_3, s_3) \text{ in } s_3 < s_2 \tag{54}$$

From (5) we have:

$$KRef(A, m_{23}, s_3) \wedge KRef(A, m_4, s_3)$$

From the previous action $unpack(A, m_{22}, m_{23}, m_{24}, m_{25})$ in s_4 we know with (10) and (9) that:

$$\begin{gathered} KRef(A, m_{25}, s_3) \\ KRef(A, m_{22}, s_4) \text{ in } s_4 < s_3 \\ Packed(A, m_{22}, m_{23}, m_{24}, m_{25}) \end{gathered} \tag{55}$$

From $decrypt(A, m_{21}, m_{22}, K'_{AB})$ in s_5 we know with (14) that:

$$(\exists s < s_5) Knows(A, Encrypted(m_{21}, m_{22}, k, s), s_4) \text{ in } s_5 < s_4$$

and from (12) we get:

$$(\exists s', s'' < s_4, a', k) s'' = do(encrypt(a', m_{22}, m_{21}, k), s') \wedge KeyPair(k, K'_{AB})$$

From (27) we can deduce that $k = K_{AB}$ and therefore we obtain:

$$(\exists s, k') Knows(A, Encrypted(m_{21}, m_{22}, K_{AB}, s), s_4)$$

Since initially the key K_{AB} are only known by A and B —see (38), (39) and (40)— and since nobody might know it because it will never go in any message —see (28) and (29)— and since from (6), (17) and (16) nobody but A and B know it, we have:

$$(\forall a, s) KRef(a, K_{AB}, s) \supset a = A \vee a = B$$

this implies with (49) that:

$$(\forall s) Author(A, m_{21}, s) \vee Author(B, m_{21}, s)$$

Moreover we suppose that A and B both know (38), (39) and (40). From here and the preceeding implication, with (3) we get:

$$(Knows(A, Author(A, m_{21}, s_4) \vee Author(B, m_{21}, s_4), s_4)$$

The only message that A sends in the whole protocol is:

$$A \rightarrow B : \ text_3, \{N_A, x_1, B, text_2\}_{K_{AB}}$$

Assuming that we work with this run and that there are not preceeding runs, it will suffice to check that m_{23} —the first element of the encrypted part— is not N_A to know sure that A has not sent it in this run.

From (54) we have $Knows(A, m_{23} = m_3, s_4)$, and, since $m_3 \neq N_A$ —because the action $unpack(a, m_2, m_3, m_4)$ goes before $createNonce(a, N_a)$ in $CPActions(ISO9798.2, a, r, 2)$— with (19) we have:

$$Knows(A, m_{23} \neq N_A, s_3)$$

From (44) we have:

$$Knows(A, m_{22} \neq m_5, s_3)$$

From (45) we have:

$$Knows(A, m_{21} \neq m_6, s_3)$$

And all together we have:

$$\neg FullIn(m_{21}, m_7)$$

Since m_6 was the only message that A has sent in this run, i.e.,

$$(\forall m, s) Sent(a, m, s) \supset (m = m_7)$$

we have, from (50):

$$Knows(A, \neg Author(A, m_{21}, s))$$

considering, as we said, only the messages from A in this run and this protocol.

So we have, from (2):

$$Knows(A, Author(B, m_{21}, s_4))$$

and from (51) since $FullIn(m_{25}, m_{21})$ we have:

$$Knows(A, Author(B, m_{25}), s_4))$$

that jointly with (55) corresponds to one half of (41) which is what we pretend to prove. $\sqcap$

The other side will be proved in a similar way if we begin from the fact:

$$DoneStep(B, ISO9798.2, r, 3, s)$$

which comes out immediately from $Finished(B, ISO9798.2, r, s)$, as the previous case. The problem is that we have only supposed $Finished(A, ISO9798.2, r, s)$ and we need:

$$Finished(A, ISO9798.2, r, s) \supset Finished(B, ISO9798.2, r, s) \tag{56}$$

Let's depart as before from (26) and (25) in (52):

$$DoneStep(A, ISO9798.2, r, 4, s_1)$$

From here we see that $DoneStep(A, ISO9798.2, r, 2, s)$ must be also true because it is a precondition of step 4 of A. The action $createNonce(a, N_a)$ is inside step 2, so we know that A has generated N_A, i.e., just after its creation A is the only one who knows N_A. We know that A has not sent N_A in any other message from (28), i.e.:

$$(\forall s) FullIn(N_A, m) \wedge Sent(A, m) \supset m = m_6$$

and we know:

$$InsideEncryption(N_A, m_6, K_{AB})$$

From (29) we know that B has not sent N_A in any other message, i.e., the only message B has sent containing N_A is message m_{18}:

$$(\forall s) FullIn(N_A, m) \wedge Sent(B, m) \supset m = m_{18}$$

and we know

$$InsideEncryption(N_A, m_{18}, K_{AB})$$

and from (38), (39), (40) and (47) we have:

$$(\forall s, a) KRef(a, N_A, s) \supset a = A \vee a = B$$

Knowing that:

$$Received(A, m_{19}, s) \wedge FullIn(N_A, m_{19})$$

from (48) we get:

$$(\exists s' < s) Sent(A, m', s') \wedge FullIn(N_A, m', s') \vee$$
$$(\exists s'' < s) Sent(B, m'', s'') \wedge FullIn(N_A, m'', s'')$$

Which only can be m_{18} or m_7. But the message received in the precondition of step 4 is m_{19}, and since $m_{19} \neq m_7$ because they have not the same shape, with (46), we have:

$$(\exists s) Sent(B, m_{18}, s)$$

and therefore the action $sent(B, m_{18})$ was done and this is the last one of step 3, so we have:

$$DoneStep(B, ISO9798.2, r, 3, s)$$

and from here and from $Finished(B, ISO9798.2, r, s)$, we show (56). As we said we did in the first half of this proof we would do in the second one, proving that:

$$KRef(B, m_{15}, s)$$
$$(\exists s) Knows(B, Author(A, m_{15}, s))$$

and with this we have both parts of (41). □

12 Conclusion

The applications and real range of the model we have presented will not be estimated till it would be applied to a broader number of protocols and once an automatisation of the theory would be tried. Nonetheless, the theory is flexible and generic enough to make possible proofs of most of the correction (or incorrection) properties of cryptographic protocols. Maybe the most promising result is that the protocol specification remains in an extremely plain and explicit manner, *justifying per se the formalisation task.*

In a wider manner, this work has needed the introduction of certain aspects and formalisms. These could be summarised into a *set of requirements* that are necessary to the objection of formalising cryptographic protocols:

- Assumptions: First of all, as it has been done here, it should be clarified the environment where we would work in, the communication channel, what implicit information (if it is) is known about a message beyond its content, what reliability about keys and cryptographic protocol we shall have and, mainly, how far we want to arrive with the formalization. It is also necessary see what kind of problems we want to detect or treat with our model and consequently whether it is necessary to contemplate the possibility that some protocols or some runs of the same one befall concurrently or sequentially.
- Situational Structuration: To study the behaviour of a cryptographic protocol one must evaluate every possible situation we may come upon, realising that one leads to another through certain actions. This idea is practically identical to Situation Calculus. This does not mean that it is indispensable to use S.C. to model cryptographic protocols (in fact there are many previous models) but it is something very similar to S.C. if it is pretended to study in detail and in a dynamic way the situations in which the different principals may find as long as actions occur.
- Fallible or Infallible Knowledge: Obviously, to model the principals we need independent agents with respective independent knowledges and that each knowledge could evolve along the situations. The model presented here is based on the concept of infallible knowledge, i.e., it is no possibility of mistake or falsity in agents' knowledge. This makes impossible to model situations where an agent thinks something and other agents thinks the contrary or even, the very frequent case where an agent cheats another and make her think that something is not true. This is, in our opinion, one of the great problems presented here. It is also difficult to introduce the concept of trustness, from which if A trusts B and A knows that some information comes from

B, A gets to believe the given information, but if A does not trust B, A will reject the information, knowing it but not believing it. This suggests that the model could be extended, adding a fluent $Believes$ in addition to $Knows$[5]. Possibly, most of the found difficulties, may be resolved with a model based on believe (fallible knowledge). Thus in the ISO protocol example, if a knows that b is the author of a message implies that b is the author of the message, but if we would want to prove that it is possible that a would think that b is the author of a message while b is not, we would have proven a failure in the protocol, what cannot be done exclusively with $Knows$.

- Protocol Specification: The protocol specification must establish much more than the habitual informal description of it, showing explicitly everything implicit in the latter. As we have seen, we start up describing step by step which are the actions of each agent involved, their preconditions to commit them, as well as certain restrictions that must be preserved by the messages involved in the protocol (e.g. revealing a key or some crucial message in the protocol). Concretely, it has been seen the need of introducing actions for encryption/decryption, packing/unpacking, sending/receiving, as well as how to model the creation of nonces, essential in most of cryptographic protocols. One of the actions that has been shown necessary to include as long as we have been working on the matter, is the one that verifies that an agent knows something in a given moment or she is sure of something, because if she is not, she cannot go on executing the rest of actions of a protocol step —in our models this action is called $verify$—. It is possible that for more complex protocols, in which different actions are possible, would be necessary to introduce some king of model similar to programming languages, including bifurcations, loops, etc.[6]

  Aside from $nonces$, many protocols include the so-called $timestamps$ to order sequentially the messages. Accordingly it would be introduced the action $createTimestamp$ whose precondition and result axioms would be:

$$Poss(createTimestamp(a,t),s)$$
$$\equiv (\forall s' < s) do(createTimestamp(a,t'),s') \supset (t' < t)$$

  The result of $createTimestamp$ is similar to that of $createNonce$:

$$Poss(createTimestamp(a,t),s) \supset KRef(a,t,do(createTimestamp(a,t),s))$$

- Case Instantiation: Once the protocol has been specified, it must be created an instance that asserts that some real agent is going to make the role of agent a, b or c (or whoever she would be) of the protocol, that certain parameters have some value and that the agent is going to behave in a reliable way or not, i.e. whether she will follow rigorously the protocol steps and, even if it is necessary that the agent will not be engaged in any other task or that she will not send any other message. In this point, it would be interesting the inclusion of some extensions to cover obligation in S.C, like those proposed by J. Pinto. It is also substantive to specify in which moment the agent begins a protocol —by means of an action in some situation— and in which moment cease following the steps of the protocol —also by means of an action— even though the protocol had not successfully finished yet.
- Protocols Interleaving: Due to the multiple instantiation of the same or different protocols, we face the possibility that several runs may be taking place simultaneously with some of the agents making different roles in each of them. This means that it should be considered every kind of message that may have been sent in the current protocol and

[5]The consequences of this and other knowledge extensions to S.C. are superficially treated in [9].

[6]Fortunately, the extension is already done in S.C. and is known as GOLOG [12].

any other, because if we do not heed these situations, we get the more common errors in cryptographic protocols (identity supplantation, cheatings, etc.). This question, that we have left outlined, is what [7] tries to undertake formally.

- Axiomatisation: the protocol actions must be related with the predicates that allow expressing the protocol properties and it should be considered those axioms which, precisely because of their obvious character, need to be introduced to carry out proofs. One must be very careful when including new predicates or fluents, because thereafter the number of axioms required blows up and the model gets harder in a non-linear progression, making even more difficult the semiautomatisation of proofs.

In conclusion, albeit that the model may be refined and extended in the future, the most remarkable thing of what we have exposed here is that it has been given a framework of how and what is required to attain the formalisation, seeing where the pitfalls appear and showing that, in any case, it seems impossible to model cryptographic protocols in detail with a dozen axioms. The question rests on whether complexity can be maintained at a reasonable level to let the model be used in the design and checking of cryptographic protocols.

References

[1] Abadi, Martin; Needham, Roger *"Prudent Engineering Practice for Cryptographic Protocols"* IEEE Transactions on Software Engineering, Vol. 22, No. 1, January 1996.

[2] Bertossi, Leopoldo; Pinto, Javier; Sáez, Pablo; Kapur, Deepak; Subramaniam, Mahadevan *"Automating Proofs of Integrity Constraints in Situation Calculus"*.

[3] Bertossi, Leopoldo; Arenas, Marcelo; Ferretti, Cristian; Delaporte, Alejandra; Sáez, Pablo; Siu, Bernardo; Strello, Mauricio *"El Razonador SCDBR: Manual de Uso e Instalación. Versión 4.0"* LYRCC, Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile, marzo de 1996.

[4] Burrows, M.; Abadi, M.; Needham, R. *"A Logic of Authentication"* Report 39 Digital Systems Research Center, Palo Alto, California, 1989. Also in Proc. Royal Soc. Londo A, vol. 426, pp. 233-271, 1989.

[5] Goldberg D.; Wagner, D. *"Randomness and the Netscape Browser"* Dr Dobb's J., Jan. 1996, pp. 66-70.

[6] Gong, L.; Needham, R.; Yahlom, R. *"Reasoning about Belief in Cryptographic Protocols"* Proc. 1990 IEEE Symp. on Security and Privacy (Oakland, California), pp. 234-248.

[7] Gürgens, Sigrid *"A Formal Analysis Technique for Authentication Protocols"* Arbeitspapier der GMD 988, April 1996.

[8] Hernández-Orallo, José *"Modelando Protocolos Criptográficos en Cálculo de Situaciones"* Departamento de Computación, Pontificia Universidad Católica de Chile, agosto 1996. Available in Spanish on "`http://www.dsic.upv.es/~jorallo/escritos/`".

[9] Hernández-Orallo, José *"Viabilidad de un Modelo de Conocimiento Falible en Cálculo de Situaciones"* Available in Spanish on "`http://www.dsic.upv.es/~jorallo/-escritos/`".

[10] ISO/IEC 9798-2: 1994(E) *"Information technology - Security techniques - Entity authentication - Part 2: Mechanism using encipherment algorithms"* 1994.

[11] Kocher, P. *"Cryptanalysis of Diffie-Hellman, RSA, DSS, and Other Systems Using Timing Attacks"* extended abstract, "`http://www.cryptography.com`", Dec. 1995.

[12] Levesque, Hector J.; Reiter, Raymond; Lespérance, Yves; Lin, Fangzhen; Scherl, Richard B. *"GOLOG: A Logic Programming Language for Dynamic Domains"*

[13] McCarthy, J. *"Situations, actions and causal laws"* TR, Stanford University 1963. Reprinted in Semantic Information Processing (M. Minsky ed.), MIT Press, Cambridge, Mass., 1968, pp. 410-417.

[14] Moore, Robert C. *"A formal theory of knowledge and action"* in Hobbs, J.R. and Moore, R.C. editors 1985, Formal Theories of the Commonsense World Ablex, Norwood, NJ. 319-358.

[15] Reiter, R. *"The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression"* in Vladimir Lifschitz, editor, Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, pages 359-380. Academic Press, San Diego, CA, 1991.

[16] Reiter, R. *"Proving Properties of States in the Situation Calculus"* 1994.

[17] Scherl, Richard B.; Levesque, Hector J. *"The Frame Problem and Knowledge-Producing Actions"* in Proceedings, Eleventh National Conference on Artificial Intelligence. 689-695

[18] Scherl, Richard B.; Levesque, Hector J.; Lespérance, Yves *"The Situation Calculus with Sensing and Indexical Knowledge"* 1996?.

[19] Schneier, B. *"Applied cryptography"* 2nd ed., John Wiley & Sons, New York, 1996.