Especificación Formal de Protocolos Criptográficos en Cálculo de Situaciones†

José Hernández-Orallo[1], Javier Pinto[2]

Resumen: *Presentamos un modelo genérico en Cálculo de Situaciones (C.S.) para formalizar protocolos criptográficos. Se distingue de otras lógicas para la modelización criptográfica que usan los conceptos de creencia o secreto porque nuestra propuesta está basada en el concepto de conocimiento y no es necesaria ninguna construcción adicional fuera del C.S. Como muestra, presentamos la especificación formal del protocolo de autentificación del ISO/IEC 9798-2 y se ilustra cómo demostrar diferentes propiedades de corrección sobre él.*

A la vista de una posible automatización de las demostraciones, el resultado más prometedor hasta ahora es que la especificación de los protocolos se mantiene de una manera extremadamente clara y explícita, facilitando la tarea de diseño y verificación, y permitiendo una implementación directa.

1. Introducción

Aunque muchos problemas de seguridad se pueden dar en la implementación de algoritmos o protocolos, es en el diseño de éstos donde más trabajo previo y riguroso se puede hacer, antes de la prueba "en real" de los mismos. Aunque siguen apareciendo algunos algoritmos criptográficos, la frenética evolución de aplicaciones de comunicación y de comercio electrónico en redes abiertas requiere la introducción de nuevos protocolos, generalmente basados en algoritmos de cifrado clásicos ya probados. En este momento, las 'dudas' y 'ataques' recaen sobre el nuevo protocolo y cualquier aumento en el grado de certeza de su funcionamiento correcto supone para la entidad que lo desarrolla una garantía (económica y de prestigio) antes de poner en funcionamiento el nuevo protocolo, como muchos casos recientes, y tristemente famosos, corroboran.

En este artículo, se presenta un modelo lo suficientemente genérico en cálculo de situaciones (C.S.) [McCarthy 1963] para formalizar distintos protocolos criptográficos. Partimos del modelo de conocimiento indéxico introducido por Scherl y Levesque [Scherl et al. 1993] y ampliado en artículos posteriores [Scherl et al. 1995], y vamos añadiendo los predicados y acciones necesarios para dar forma a la teoría. Como nota destacable, y a diferencia de otras lógicas para la modelización criptográfica, podemos apuntar que se basa en el concepto de conocimiento (y no en el de creencia ni en el de secreto) y no se hará necesaria ninguna construcción especial fuera del C.S.

Para los fundamentos criptográficos, seguiremos las líneas, notaciones y algunos ejemplos clásicos de la literatura, fundamentalmente basadas en [Abadi et al. 1996] y [Gürgens 1996]. Del primero se han intentado extraer algunos principios informales, haciéndolos requisitos en nuestra formalización, lo que puede facilitar el diseño de protocolos utilizando la misma. Del segundo se ha extraído una formalización extendida y mejorada de la lógica BAN [Burrows et al. 1989], que, a su vez, hemos adaptado y ampliado a nuestras necesidades.

Como ejemplo se introduce el protocolo de autentificación 5.1.2. del ISO/IEC 9798-2, que especificamos formalmente para demostrar su corrección, asumiendo ciertos supuestos. Según [Gürgens 1996], acerca del mismo protocolo, éste puede dar problemas cuando existen distintas series del mismo; aunque no se incluye aquí por falta de espacio, nuestro modelo permite demostrar la existencia de estos problemas.

Las construcciones y axiomas presentados en este artículo son lo más significativo para ilustrar la teoría; una exposición más completa y detallada de la misma se puede encontrar en la versión preliminar de este texto [Hernandez-Orallo & Pinto 1996].

2. Ejemplo

Para ver la necesidad de los diferentes formalismos y construcciones que se van a ir introduciendo, vamos a presentar un protocolo sencillo, el ISO 9798-2 descrito en [ISO9798-2 1994] y se va a intentar hacer un análisis similar al hecho por [Gürgens 1996].

2.1. Especificación Informal

La ***descripción informal*** de una serie o *run* (ejecución) del protocolo, tal y como especifica el estándar, es:

0. Ambos interlocutores, A y B, comparten una clave simétrica denominada K_{AB}.
1. B envía un número aleatorio N_B y, opcionalmente, un campo de texto $text_1$ a A.
2. Cuando A recibe el mensaje, A calcula un número aleatorio N_A y envía un texto $text_3$, seguido del mensaje (N_A, N_B, B, $text_2$) cifrado con K_{AB}.
3. Al recibir el mensaje 2, B verifica el texto cifrado, descifrándolo y comprobando la corrección del identificador B que le distingue y que el número aleatorio N_B enviado en el paso 1, coincide con el que contiene el texto cifrado.

† La mayor parte de esta investigación se originó y desarrolló durante la cooperación establecida a raíz de una beca intercampus del Instituto de Cooperación Iberoamericana (ICI) entre el Depto. de Lógica de la Universidad de Valencia y el Depto. de Ciencia de Computación de la Pontificia Universidad C. de Chile.

[1] Universitat Politècnica de València, Departament. de Sistemes Informàtics i Computació, Camí de Vera 14, Aptat. 22.012 E-46071, València. jorallo@dsic.upv.es

[2] Pontificia Universidad Católica de Chile, Escuela de Ingeniería, Departamento de Ciencia de Computación, Casilla 306, Santiago 22, Chile. jpinto@ing.puc.cl

4. B envía un texto $text_5$, seguido del mensaje (N_B, N_A, $text_4$) cifrado con K_{AB}.
5. A la llegada del mensaje 3, A verifica el mensaje cifrado, descifrándolo y comprobando que el número aleatorio N_B recibido de B en el mensaje 1 coincide con el número aleatorio contenido en el mensaje y que el número aleatorio N_A enviado a B en el mensaje 2 coincide con el número aleatorio contenido en el mensaje.
6. El objetivo del protocolo es que, sin usar un servidor de claves, ambos estén convencidos al final de su ejecución de que han hablado el uno con el otro.

Utilizando la ***notación clásica*** frecuente en la literatura criptográfica [Abadi et al. 1996], que clarifica un poco más el proceso, el protocolo es el siguiente:

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

La primera parte (antes de los dos puntos) indica el origen y destino de los mensajes, a continuación se muestra una serie de campos empaquetados y posiblemente cifrados (entre llaves y denotándose la clave con un subíndice).

El problema de esta notación es que no expresa lo que verifican los interlocutores, sólo lo que se envía.

En su lugar presentamos una ***notación extendida*** con la que podemos expresar las precondiciones o comprobaciones antes de poder realizar cada acción:

0. PRECONDICIONES INICIALES:

K_{AB} es una clave simétrica compartida por A y B.

1. PRECONDICIONES de B:

Ninguna

ACCIONES:

$B \to A{:}\, N_B, text_1$

2. PRECONDICIONES de A:

Recibe un mensaje (x_1,x_2)

ACCIONES:

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

3. PRECONDICIONES de B:

Recibe un mensaje (x_3,x_4)

Descifra x_4 con K_{AB} en (x_5,x_6,x_7,x_8)

Verifica que x_7=B

Verifica que x_6= N_B

ACCIONES:

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

4. PRECONDICIONES de A:

Recibe un mensaje (x_9,x_{10})

Descifra x_{10} con K_{AB} en (x_{11},x_{12},x_{13})

Verifica que x_{11}= x_1

Verifica que x_{12}= N_A

5. OBJETIVOS:

A sabe que ha hablado con B

B sabe que ha hablado con A

6. RESTRICCIONES DE EJECUCION:

A y B no deben comunicar K_{AB} en otro mensaje.

A y B no deben comunicar N_A y N_B en ningún mensaje salvo los que aparecen aquí.

A y B no deben comunicar $text_2$ y $text_4$ en ningún otro mensaje.

Y con esto hemos completado la especificación informal de una serie (o *run*) del protocolo ISO 9798-2.

2.2. Objetivos

Ahora nos preguntamos si realmente el protocolo cumple su función y si no, a qué tipo de ataques es vulnerable ¿Tenemos alguna forma de asegurar cuándo ocurren o no ocurren este tipo de cosas? La única manera de estar completamente seguro de que cierto hecho no va a suceder consiste en formalizar el protocolo y realizar la pregunta formalmente en este marco, mostrando que no puede ser el caso.

Para ello es necesario poder expresar el esquema anterior con una notación formal. Como ya hemos dicho, aparte de otras aproximaciones, que lo han conseguido parcialmente (demostrando algunas propiedades) nosotros vamos a utilizar para ello el cálculo de situaciones, donde no es necesaria, como veremos, ninguna construcción fuera del mismo y en el que, posteriormente podemos ensayar una semiautomatización de las demostraciones.

Deberemos, en resumen, formalizar cómo se envían y reciben los mensajes, cómo se cifran, qué tipos de claves se pueden usar, qué es una serie de un protocolo e incluso introducir formalmente conceptos raramente formalizados en la literatura como *nonces* (marcas aleatorias) o *timestamps* (marcas temporales).

3. Supuestos

Antes de comenzar es necesario asumir ciertos supuestos con los que vamos a estudiar los protocolos, muchos de ellos habituales ya en la literatura criptográfica.

1. Algoritmo de Cifrado: la modelización presentada funciona independientemente de qué algoritmos de cifrado se utilicen, por ejemplo DES, RSA u otros. No se contempla la posibilidad de que estos algoritmos se venzan por fuerza bruta o 'timing-attacks' [Goldberg et al. 1996] [Koder 1995] al cabo de un tiempo, sino que supondremos que las claves tienen suficientes bits para que esto sea probabilísticamente imposible. En el modelo presentado se supone que sólo hay un único algoritmo de cifrado, pero podría ampliarse fácilmente para contemplar varios algoritmos diferentes.

2. Intercalado de Protocolos: se va a estar especialmente concienzado con los problemas que pueden aparecer del hecho de que varias series (*runs*) del mismo o diferentes protocolos estén discurriendo al mismo tiempo. El protocolo presentado y otros muchos han mostrado fallos en el caso de series intercaladas del mismo protocolo. El problema es que el conjunto de posibles mensajes y su momento de aparición es infinito. Éste es uno de los problemas latentes en [Gürgens 1996].

3. Seguridad y Fiabilidad: se supone un único canal de comunicación 'privado' o punto a punto, con posibilidad de pérdida, duplicación o llegada errónea de los mensajes. Ya que tampoco hay ninguna seguridad sobre la autenticidad del origen de un mensaje, el mejor modo de modelar este canal es, paradójicamente, suponer que se trata de un canal distribuido (*broadcast*), en el que todos los participantes reciben y pueden leer los mensajes. También, cualquier participante puede enviar el mensaje que le plazca en cualquier momento. Aunque ésta es la aproximación clásica y la que nosotros usaremos en general, hemos introducido adicionalmente un predicado especial ***SecureComm(a,b)*** para representar aquellos casos ideales en los que podemos asegurar tajantemente que la comunicación entre dos interlocutores puede realizarse privadamente, sin escuchas. También usaremos, del mismo modo, un predicado ***ReliableComm(a,b)*** cuando queramos expresar que la comunicación no falla nunca, es decir, independientemente de que haya o no escuchas, un mensaje enviado implica un mensaje recibido sin ningún error. Por último, los agentes pueden ser personas o más comúnmente algoritmos y pueden fallar o no seguir la especificación. Cuando queramos indicar que son infalibles y completamente fieles a la especificación lo denotaremos con el predicado ***Compliant***.

4. Claves y Privacidad: no se asumirá sin embargo ciertas propiedades habituales de claves públicas y privadas, no porque querramos hacer un modelo muy genérico, sino porque gracias al fluente ***Knows***, podemos expresar perfectamente qué claves conoce cada agente y cuáles desconoce, así como las que son compartidas. Por el mismo motivo, tampoco es necesario la introducción de predicados para modelar lo que es secreto.

5. Mensajes: los mensajes no contienen más información que la que realmente se explicita (éste es más o menos el principio 1 de [Abadi et al. 1996]). Por tanto, cualquier información que se suponga implícitamente no será considerada en el modelo, ignorando por tanto su orden, su serie, su origen, su destino, si está codificado o no, etc. Todo ello deberá saberse o deducirse por el destinatario del mensaje porque conoce el protocolo.

4. Fundamentos

4.1 El Lenguaje

Aunque posteriormente presentaremos el C.S., de donde provendrán las construcciones lógicas básicas, vamos a introducir las clases de objetos con los que vamos a trabajar. Para no confundir el envío de mensajes (→) con la implicación, usaremos ⊃ para esta última.

Con la letra mayúscula ***A*** denominaremos a la clase de agentes (también llamados interlocutores). Los agentes son los individuos que se comunican por el canal. En nuestro modelo van a coincidir con los agentes tal y como se conocen en C.S. Usaremos las letras minúsculas **a**, **b** y **c**, para elementos de dicha clase.

Del mismo modo ***M*** será la clase de mensajes. Un mensaje es el objeto que puede ser enviado/recibido por un agente. ***P*** es la clase de protocolos. ***R*** es la clase de series (*runs*). Una serie es la realización de un protocolo desde el principio hasta el fin. Como hemos dicho, permitiremos que varias series puedan discurrir secuencial o concurrentemente. ***E*** es la clase de pasos (*steps*). Cada serie de un protocolo se compone de varios pasos, numerados con números naturales. ***K*** es la clase de claves (*keys*) utilizadas para cifrar o descifrar mensajes. Hay varios tipos de claves, como veremos. ***S*** es la clase de situaciones. Son los momentos o estados del cálculo de situaciones y aquí van a hacer la misma función que los puntos temporales en otras formalizaciones. ***Q*** es la clase de acciones, tal y como se conocen en cálculo de situaciones, permitiendo pasar de unas situaciones a otras. ***N*** es la clase de marcas (*nonces*), muy usuales en protocolos de autentificación. Las marcas son números aleatorios generados por un agente y que supondremos imposibles de repetir por otro agente a no ser que se comunique. ***W*** es la clase de proposiciones para el tipo de los parámetros de los metapredicados, como ***Knows***.

4.2. Cálculo de Situaciones y Conocimiento

Por razones de espacio y del objetivo de este artículo, no se va a hacer una introducción sobre qué es el cálculo de situaciones (C.S.). Recordemos a grandes rasgos sus elementos: el C.S. se trata de una extensión temporal de la lógica de primer orden que consta de una situación inicial s_0, a la que es posible aplicar ciertas acciones y llegar a otras situaciones. La relación de accesibilidad entre situaciones se representa por el signo **<** y por ejemplo $s_1 < s_2$ quiere decir que s_2 es accesible desde s_1. Aquellos predicados cuyo valor de verdad varíen según las situaciones (y por tanto tendrán como último parámetro un elemento de la clase ***S***) se les denomina fluentes.

Los acciones tienen unas precondiciones para poderse aplicar, que se denotan con ***Poss***. Si la acción es posible, el resultado de realizar (***do***) una acción ***a*** en una situación ***s*** es una nueva situación ***s'***. Es decir,

$$Poss(acc(\ldots),s) \wedge s' = do(acc(\ldots),s) \supset \text{ Fluentes que se deben cumplir en s'}$$

Para aquellos que no estén familiarizados con el C.S. pueden consultar la bibliografía, especialmente [Scherl et al. 1993]. En dicho artículo se presenta un conocimiento basado en mundos posibles según [Moore 1985] que es el que vamos a utilizar. Adicionalmente se introduce la posibilidad de conocimiento indéxico que, aunque en lo sucesivo no aparecerá, creemos que puede ser necesario para modelar algunos protocolos más complejos.

Al modelo anterior se le añadieron los agentes, siguiendo la formalización presentada en [Scherl et al. 1995] que combina ambos mediante la notación ***Knows(a, P, s)***, entendido como "el agente ***a*** conoce que ***P*** en la situación s". Esto permite expresar algunas sentencias muy útiles para la modelización criptográfica, como por ejemplo: "a_1 sabe (en s_1) que a_2 sabía (en s_2) algo (w)"

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

La formalización y la resolución del problema del marco (*frame problem*) para esta notación se encuentra en el mismo artículo de Scherl et al. y también pueden ser muy útiles a la hora de demostrar propiedades de protocolos.

Para simplicar la tarea subsiguiente vamos a extraer y citar a continuación un conjunto reducido de fórmulas provenientes de la definición de Knows de [Scherl et al. 1995]:

$$Knows(a, w, s) \supset w \text{ es verdadero en la situación } s$$

Es decir, el conocimiento como lo estamos viendo es infalible; un agente *conoce* un subconjunto (generalmente propio) de todo lo cierto en una situación, pero en ningún caso *conoce* algo falso. En la definición de **Knows** de [Scherl et al. 1995] se incluye la propiedad de cierre, es decir, no hay conocimiento cuyas consecuencias lógicas no sean conocidas también.

Para conocer algo que no sean enunciados, por ejemplo mensajes, claves, etc., se introduce el fluente *KRef* apartir de *Knows* de la manera siguiente:

$$Kref(a,t,s) \overset{def}{=} (\exists x) Knows(a, t = x, s)$$

donde x no aparece en t

Ahora vamos añadir un axioma de marco que nos dice que no sabemos nada nuevo si no lo hemos recibido (***Received***) dentro de (***FullIn***) algún mensaje:

$$\begin{aligned}&\neg KRef(a,m',s) \wedge \\ &\wedge \left[(\forall m)(\forall s')(s'' > s' > s) Received(a,m,s') \supset \neg FullIn(m',m)\right] \supset \\ &\supset (\forall s' < s'') \neg KRef(a,m',s')\end{aligned}$$

El definición del predicado ***FullIn(m',m)*** se encuentra en [Hernández-Orallo & Pinto 1996] pero, escuetamente, significa que el mensaje ***m'*** esté en ***m***, pudiendo haber cifrados anidados.

5. Acciones

A partir del modelo de conocimiento presentado en el punto anterior, vamos a introducir las acciones necesarias dentro del C.S.:

5.1. Empaquetamiento

Las acciones ***pack*** y ***unpack*** nos permiten dar forma a los mensajes, y que el orden de sus elementos tenga importancia a la hora de comparar mensajes. Así no es necesario introducir el concepto de *tipo* de los mensajes (según su estructura).

Veamos en primer lugar los axiomas de precondición y efecto de ***pack*** y ***unpack***, que son bastante sencillos:

$$Poss(pack(a,m_1,m_2,m),s) \equiv KRef(a,m_1,s) \wedge KRef(a,m_2,s)$$

$$Poss(pack(a,m_1,m_2,m),s) \supset KRef(a,m,do(pack(a,m_1,m_2,m),s))$$

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

$$\begin{aligned}&Poss(unpack(a,m,m_1,m_2),s) \wedge s' = do(unpack(a,m,m_1,m_2),s) \supset \\ &\supset KRef(a,m_1,s') \wedge KRef(a,m_2,s')\end{aligned}$$

Más adelante, para abreviar, utilizaremos la misma notación para empaquetar más de dos mensajes.

5.2. Cifrado

Entremos ahora en materia con las acciones ***encrypt*** y ***decrypt***, que van a ser la base, junto con *Knows* de nuestra modelación. Ambas van a tomar el mismo tipo y número de argumentos, veamos:

- ***encrypt(a, m, m', k)*** significa "el agente ***a*** cifra ***m*** utilizando la clave ***k*** dando como resultado el mensaje ***m'*** cifrado".

De modo similar,

- ***decrypt(a, m', m, k)*** significa "el agente a descifra ***m'*** utilizando la clave ***k*** dando como resultado el mensaje m en claro".

En primer lugar, veamos las precondiciones para poder ejecutar ambas acciones. Para cifrar requerimos la información en claro y la clave con la que se va a codificar.

$$\begin{aligned}&Poss(encrypt(a,m,m',k),s) \equiv \\ &\equiv KRef(a,m,s) \wedge KRef(a,k,s) \wedge EncodingKey(k)\end{aligned}$$

Fijémonos que no es necesario que el agente ***a*** sepa que ***k*** es una clave de codificación, así contemplamos la posibilidad de pruebas por ensayo y error (legítimas o no).

Para decodificar requerimos algo parecido, pero es necesario una precondición más; el descifrado sólo será posible en el caso de que la información que nos suministren haya sido codificada anteriormente y que la clave de descifrado sea compatible con la clave de cifrado (es decir, formen un par de claves).

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

Visto esto, el resultado de la acción de cifrar es bien simple, el agente pasa a saber *lo que acaba de hacer*, es decir, que ***m'*** es el resultado de cifrar ***m*** con la clave ***k***:

$$\begin{aligned}&Poss(encrypt(a,m,m',k),s) \wedge s' = do(encrypt(a,m,m',k),s) \supset \\ &\supset Knows(a, Encrypted(m,m',k,s),s')\end{aligned}$$

Y la de descifrar es la que tiene resultados más palpables de cara al conocimiento:

$$\begin{aligned}&Poss(decrypt(a,m',m,k),s) \supset (\exists s',k') s' = do(decrypt(a,m',m,k),s) \wedge \\ &\wedge (\exists s'' < s) Knows(a, Encrypted(m',m,k',s''),s') \wedge KRefs(a,m,s') \wedge \\ &\wedge Knows(a, Decrypted(m',m,k,s'),s') \wedge Knows(a, KeyPair(k,k'),s')\end{aligned}$$

Es decir, ocurre que, aparte de saber la información ***m*** en claro (que es lo que se pretende), el agente sabe (cosa que es posible que no supiera en ***s***, habiendo hecho un ensayo como hemos comentado antes) que dicha información fue codificada en ***m*** utilizando una ***k'*** (también probablemente desconocida) y que es un par junto con ***k***.

5.3. Envío y Recepción de Mensajes

Ahora vamos a introducir la acción ***send(a₁, m)*** que representa que "el agente $\boldsymbol{a_1}$ envía el mensaje ***m***". Nótese que no se especifica el destino por lo que ya apuntamos en los supuestos de un canal distribuido (*broadcasted*). Tenemos:

$$Poss(send(a,m),s) \equiv KRef(a,m,s)$$

Como ya dijimos, la acción *receive* es completamente independiente de *send* porque el canal generalmente no es seguro ni fiable, por tanto la acción send no tiene ningún efecto aparte de que se ha enviado:

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

La acción ***receive(a,m)*** que quiere decir que "el agente ***a*** recibe el mensaje ***m***" queda de la siguiente forma:

$$Poss(receive(a,m),s) \wedge s' = do(receive(a,m),s) \supset$$
$$\supset KRef(a,m,s') \wedge Knows(a, Received(a,m,s'),s')$$

$$Poss(receive(a,m),s) \equiv$$
$$\equiv (\exists a',s',s'' < s) KRefs(a',m,s') \wedge (s'' = do(send(a',m),s'))$$

5.4. Marcas

Ahora vamos a introducir una acción que permite a un agente crear marcas aleatorias (*nonces*):

$$Poss(createNonce(a,n),s) \equiv (\forall a')\neg KRef(a',n,s)$$

Es decir, un *nonce* se crea nuevo, sin la posibilidad de que el mismo número hubiera existido en el conocimiento de otro agente (o de él mismo), considerando el número de bits del *nonce* lo suficientemente alto para que la probabilidad de casualidad o fuerza bruta sea despreciable. Ahora vemos el resultado de la acción, bastante obvio:

$$Poss(createNonce(a,n),s) \supset KRef(a,n,do(createNonce(a,n),s))$$

5.5. Verificaciones

Por último vamos a añadir una de las acciones más importantes, la acción ***verify*** que nos permitirá introducir verificaciones por parte de los agentes. Éstas son necesarias para comprobar que ciertos datos son correctos (utilizando un predicado de igualdad). De este modo, el agente aborta el protocolo si no conoce dicho predicado (por ejemplo no sabe la igualdad de un texto y un paquete recibido por lo que no es auténtico).

A pesar de su importancia, se define de una forma muy sencilla, una precondición y ningún resultado de dicha acción. Es decir, sólo se puede ejecutar si se sabe aquello a verificar.

$$Poss(verify(a,w),s) \equiv Knows(a,w,s)$$

6. La Especificación de los Protocolos

Una vez introducidas las acciones que pueden realizar los agentes, queda por especificar una parte crucial, cómo se especifican los protocolos, de qué acciones constan, qué precondiciones necesitan, etc.

Independientemente de que ciertas acciones se hagan o no, que el agente se comporte bien o no, el protocolo debe especificar ciertas acciones y qué precondiciones son necesarias en cada paso de cada protocolo, para cualquier serie y agente. Para ello, introduciremos un enunciado ***CPPreconds(p, a, r, e)*** que nos expresa las precondiciones del protocolo ***p*** para que ***a***, en el run ***r***, puede comenzar el paso ***e*** (***a*** y ***r*** serán, generalmente, las variables libres).

Una vez especificadas las precondiciones de cada etapa, debemos indicar cuáles son las acciones que puede realizar. Esto lo haremos mediante un vector de acciones denominado ***CPActions***, relacionando precondiciones y acciones de la manera siguiente:

$$Poss(CPActions(p,a,r,e),s) \equiv CPPreconds(p,a,r,e)$$

$$Poss(CPActions(p,a,r,e),s) \supset$$
$$\supset AtStep(a,p,r,e,Do(CPActions(p,a,r,e),s)$$

Siendo *CPActions* un vector de acciones que se deben realizar secuencialmente, utilizando la notación clásica en C.S.:

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

Para que las acciones se realicen se deben cumplir las precondiciones de cada una de ellas según se han ido definiendo. Lo que diferencia un vector de acciones de una secuencia de los mismos, es que cuando una acción del vector no se pueda realizar no se continúa y se tomará como que el vector de acciones ***CPActions*** no se ha podido realizar *ninguna* de ellas (o todo o nada). Para ello, se introduce[3] un ***Do*** similar al GOLOG [Levesque et al. 1997]:

$$Do([q_1; q_2; \ldots; q_n], s, s_n) \equiv$$
$$\equiv Poss(q_1,s) \wedge s_1 = do(q_1,s_1) \wedge Poss(q_2,s) \wedge s_2 = do(q_2,s_1) \wedge$$
$$\ldots \wedge Poss(q_n,s_{n-1}) \wedge s_n = do(q_n,s_{n-1})$$

Por último, vamos a especificar un fluente auxiliar ***Finished*** a partir de las acción ***end***, que utilizaremos más adelante.

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

$$Poss(end(a,p,r),s) \supset Finished(a,p,r,do(end(a,p,r),s)$$

Obsérvese, que para poder terminar el protocolo se debe estar en el *LastStep*, lo cual se debe indicar al especificar el protocolo. La acción ***end*** no se introduce en la especificación del protocolo. También se podría contemplar una acción ***begin*** para complementar la existencia de precondiciones para que un protocolo pudiera comenzarse, ligándose a un fluente ***Started*** de manera similar a como se liga ***end*** con ***Finished***.

7. Especificación del ISO 9798-2

Por fin vamos a pasar a dar ya la especificación formal de nuestro ejemplo, el ISO 9798-2.

En primer lugar, ya podemos establecer los axiomas referentes a la clave compartida:

[3] Una definición bien formada debería ser recursiva.

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

También sería sencillo especificar quién conoce y quién no conoce estas claves.

Finalmente, la especificación del protocolo, hecha a partir de la notación extendida del punto 2.

ETAPA 1:

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

$$CPActions(ISO9798.2, b, r, 1) = \begin{bmatrix} createNonce(b, n_b); \\ pack(b, n_b, text_1, m_1); \\ send(b, m_1) \end{bmatrix}$$

ETAPA 2:

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

$$CPActions(ISO9798.2, a, r, 2) = \begin{bmatrix} unpack(a, m_2, m_3, m_4); \\ createNonce(a, n_a); \\ pack(a, n_a, m_3, b, text_2, m_5); \\ encrypt(a, m_5, m_6, K_{ab}); \\ pack(a, text_3, m_6, m_7); \\ send(a, m_7) \end{bmatrix}$$

ETAPA 3:

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

$$CPActions(ISO9798.2, b, r, 3) = \begin{bmatrix} unpack(b, m_8, m_9, m_{10}); \\ decrypt(b, m_{10}, m_{11}, K_{ab}'); \\ unpack(b, m_{11}, m_{12}, m_{13}, m_{14}, m_{15}); \\ verify(b, m_{14} = b); \\ verify(b, m_{13} = nb); \\ pack(b, n_b, n_a, text_4, m_{16}); \\ encrypt(b, m_{16}, m_{17}, K_{ab}); \\ pack(b, text_5, m_{17}, m_{18}); \\ send(b, m_{18}) \end{bmatrix}$$

ETAPA 4:

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

$$CPActions(ISO9798.2, a, r, 4) = \begin{bmatrix} unpack(a, m_{19}, m_{20}, m_{21}); \\ decrypt(a, m_{21}, m_{22}, K_{ab}'); \\ unpack(a, m_{22}, m_{23}, m_{24}, m_{25}); \\ verify(a, m_{23} = m_3); \\ verify(a, m_{24} = n_a) \end{bmatrix}$$

RESTRICCIONES DE EJECUCIÓN:

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

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

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

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

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

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

$$(\forall m, s) Sent(a, m, s) \wedge FullIn(text2, m) \supset m = m_7$$

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

Por último, queda especificar cuáles son los últimos pasos del protocolo:

$$LastStep(ISO9798.2, a, r, 4)$$

$$LastStep(ISO9798.2, b, r, 3)$$

8. Formalización de Propiedades. Objetivos y Seguridad

En este momento deberíamos ser capaces de demostrar si se cumplen o no ciertas propiedades.

8.1. Autoría

En primer lugar, preguntaremos si se cumple el objetivo, es decir, si se certifica la autoría de dos de los mensajes:

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

En definitiva, que se hayan comunicado entre sí asegurando el origen de los textos. Evidentemente, el resto de textos se envían en claro y no se puede saber si alguien interceptó el mensaje y pudo cambiar los textos en claro.

8.2. Privacidad

También nos planteamos si es seguro, es decir, si se mantienen secretos los textos privados:

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

8.3. Fiabilidad

Aunque $\boldsymbol{m_{25}}$ debería corresponder a $\boldsymbol{text_4}$ y $\boldsymbol{m_{14}}$ a $\boldsymbol{text_2}$, no se puede demostrar ambas correspondencias, porque puede ser que ambos mensajes provengan del interlocutor auténtico, pero podrían estar mal colocados o erróneos. En cambio, sí que prodríamos en el caso de que la comunicación fuera fiable (***ReliableComm***). Aún más, si una serie del protocolo se empieza y los interlocutrores se ajustan al protocolo (***Compliant***), ¿la serie termina?

$$Compliant(A, ISO9798.2) \wedge Compliant(B, ISO9798.2) \wedge$$
$$\wedge Started(B, ISO9798.2, r, s) \wedge ReliableComm(A, B) \wedge$$
$$\wedge ReliableComm(B, A) \supset (\exists s') Finished(A, ISO9798.2, r, s)$$

Para poder demostrar dichas propiedades, como se hace en [Hernandez-Orallo & Pinto 1996], se han usado, al igual que otras teorías en C.S., algunos axiomas más (alrededor de 20), la mayoría obvios o genéricos en criptografía, que relacionan ciertos predicados o fluentes y que dan el verdadero cuerpo a la teoría.

9. Conclusiones

El verdadero alcance y utilidad del modelo presentando no se podrá estimar hasta la aplicación a un número amplio de protocolos y hasta la automatización de las demostraciones de corrección (por ahora manuales) utilizando el razonador automático SCDBR [Bertossi et al. 1996], usado ya para demostrar propiedades en otros ámbitos. Tanto nuestra especificación como la herramienta son lo suficientemente flexibles y genéricas para poder demostrar muchas de las propiedades de corrección e incorrección de otros protocolos.

Quizás el resultado más prometedor por el momento es que la especificación de los protocolos queda de una manera extremadamente sencilla y explícita, *justificando por sí sólo la tarea de formalización*, ya que usando cualquier intérprete de Prolog (codificando las extensiones para C.S.) pueden ser implementados y probados en él, sirviendo como herramienta de diseño, verificación o incluso para normativas o enseñanza de los protocolos.

En definitiva, independientemente de que el modelo pueda ser mejorado y refinado con trabajos y extensiones futuros, lo más importante de lo aquí expuesto es que se ha dado un planteamiento de cómo y qué es necesario para realizar la formalización, viendo dónde aparecen las dificultades y mostrando que, en ningún caso, parece ser posible modelar protocolos criptográficos con una docena de axiomas. La cuestión (y nuestro trabajo futuro) se centra en conseguir que la complejidad se mantenga a un nivel razonable para permitir que el modelo tenga un uso práctico para el diseño y comprobación de algoritmos criptográficos antes de su puesta en real.

Bibliografía

[Abadi et al. 1996] Abadi, M.; Needham, R. "Prudent Engineering Practice for Cryptographic Protocols" IEEE Transactions on Software Engineering, Vol. 22, no. 1, Jan. 1996.

[Bertossi et al. 1996] Bertossi, L.; Pinto, J.; Saez, P.; Kapur, D.; Subramaniam, M. "Automating Proofs of Integrity Constraints in Situation Calculus" in Z.Pawlak and Z.W. Ras (eds.) Proceedings of the 9th International Symposium on Methodologies for Intelligent Systems, 212-222, Lecture Notes in AI, Springer-Verlag 1996.

[Bertossi et al. 1996] Bertossi, L.; Arenas, M.; Ferretti, C.; Delaporte, A.; Sáez, P.; Siu, B.; Strello, M. "El Razonador SCDBR: Manual de Uso e Instalación. Versión 4.0". LYRCC, D. Ciencia de la Computación, Pontificia Universidad C. de Chile, marzo de 1996.

[Burrows et al. 1989] M. Burrows, M. Abadi, R. Needham "A Logic of Authentication" Report 39 Digital Systems Research Center, Palo Alto, Ca., 1989. in Royal Soc. London A, v. 426, pp. 233-271, 1989.

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

[Gong et al. 1990] 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, 1996.

[Gürgens 1996] Gürgens, Sigrid "A Formal Analysis Technique for Authentication Protocols" TR GMD 988, April 1996.

[Hernandez-Orallo & Pinto 1996] Hernández-Orallo, J.; Pinto, J. "Modelando Protocolos Criptográficos en Cálculo de Situaciones" Departamento de Computación, P. Universidad Cat. de Chile, 1996.

[ISO9798-2 1994] ISO/IEC 9798-2: 1994(E) "Information technology - Security techniques - Entity authentication - Part 2: Mechanism using encipherment algorithms" 1994.

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

[Levesque et al. 1997] Levesque, H.J.; Reiter, R.; Lespérance, Y.; Lin, F.; Scherl, R.B. "GOLOG: A Logic Programming Language for Dynamic Domains", J. of Logic Programming, v. 31, pp. 59-84, 1997.

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

[Moore 1985] R.C. Moore "A formal theory of knowledge and action" in Hobbs, J.R.; Moore, R.C. (eds), Formal Theories of the Commonsense World, Norwood, NJ. 319-358, 1985.

[Reiter 1991] R. Reiter "The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, ed., AI and Mathematical Theory of Computation: Papers in Honor of J. McCarthy, pages 359-380. Academic Press, San Diego, 1991.

[Reiter 1994] R. Reiter "Proving Properties of States in the Situation Calculus", Artificial Intelligence, vol. 64, n.2, 337-351, Dec. 1997.

[Scherl et al. 1993] R.B. Scherl and H.J. Levesque "The Frame Problem and Knowledge-Producing Actions" in Proceedings, 11th National Conference on AI. 689-695

[Scherl et al. 1995] R.B. Scherl, H.J. Levesque and Y. Lespérance "The Situation Calculus with Sensing and Indexical Knowledge" in M.Koppel & E.Shamir, Proceedings of BISFAI'95: The 4th Bar-Ilan Symp. on Foundations of Artificial Intelligence", pp. 86-95, 1995.

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