UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Implementación de una interfaz gráfica para Maude-NPA

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Damián Aparicio Sánchez

Tutor: Santiago Escobar Román

Septiembre 2016

Resumen

Maude-NPA es una herramienta de verificación de protocolos criptográficos desarrollada por la University of Illinois at Urbana-Champaign (EE.UU.), el Navy Research Laboratory en Washington D.C. (EE.UU.) y la Universitat Politècnica de València. Maude-NPA es un poderoso analizador, es flexible ya que puedes crear tu propia notación y permite una larga variedad de propiedades criptográficas como homomorfismo o or-exclusivo. Sin embargo, la interfaz existente para modelar protocolos y luego verificar propiedades de seguridad de éstos es compleja y difícil de entender, por lo que hace a la herramienta Maude-NPA poca atractiva.

Andrew Russel Cholewa, estudiante de máster en la University of Illinois at Urbana-Champaign propuso un nuevo lenguaje de modelado de protocolos y de verificación de sus propiedades de seguridad, denominado Maude-NPA Protocol Specification Language (Maude-PSL). Maude-PSL utiliza la notación Alice y Bob estándar para definir protocolos de forma directa: la interpretación de cada uno de los mensajes enviados o recibidos por cada participante (rol) utilizando tanto la información asumida al comienzo de la ejecución del protocolo como la información de los participantes al final de la ejecución.

En este proyecto de fin der grado durante una estancia en la University of Illinois at Urbana-Champaign de cinco meses, me he centrado en modelar y verificar la mayor cantidad de protocolos criptográficos descritos en Maude-PSL, incluyendo protocolos con homomorfismo, y conseguir que tengan el mismo resultado que con Maude-NPA original y, en el caso de no ser así, modificar los protocolos para confirmar que se obtiene el mismo resultado que en Maude-NPA.

Palabras clave: Maude-NPA, Maude-PSL, protocolo criptográfico, homomorfismo.

Índice

Tabla de ilustraciones

1. Introducción

Maude-NPA es una herramienta de verificación de protocolos criptográficos desarrollada por la University of Illinois at Urbana-Champaign (EE.UU.), el Navy Research Laboratory en Washington D.C. (EE.UU.) y la Universitat Politècnica de València. Maude-NPA es un poderoso analizador, es flexible ya que puedes crear tu propia notación y permite una larga variedad de propiedades criptográficas como homomorfismo o ó-exclusivo. Sin embargo, la interfaz existente para modelar protocolos y luego verificar propiedades de seguridad de éstos es compleja y difícil de entender, por lo que hace a la herramienta Maude-NPA poca atractiva.

Andrew Russel Cholewa, estudiante de máster en la University of Illinois at Urbana-Champaign propuso en 2015 durante su tesis de máster un nuevo lenguaje de modelado de protocolos y de verificación de sus propiedades de seguridad, denominado Maude-NPA Protocol Specification Language (Maude-PSL). Maude-PSL utiliza la notación Alice y Bob estándar para definir protocolos de forma directa: la interpretación de cada uno de los mensajes enviados o recibidos por cada participante (rol) utilizando tanto la información asumida al comienzo de la ejecución del protocolo como la información de los participantes al final de la ejecución.

Conseguí una beca, con código PF/US/0124/1, del programa Faro del Ministerio de Educación, Cultura y Deporte para realizar prácticas de empresa en EE.UU. Estoy disfrutando de dicha beca en la University of Illinois at Urbana-Champaign (EE.UU.) de abril a septiembre de 2016. Durante este tiempo me he centrado en modelar y verificar la mayor cantidad de protocolos criptográficos descritos en Maude-PSL, incluyendo protocolos con homomorfismo, y conseguir que tengan el mismo resultado que con Maude-NPA original y, en el caso de no ser así, modificar los protocolos para confirmar que se obtiene el mismo resultado que en Maude-NPA.

Esta tesina de fin de grado recoge los resultados de dicho proceso de modelado y verificación de protocolos en Maude-PSL. Las secciones 2 y 3 describen Maude-PSL de forma breve. La sección 4 se centra en los protocolos que han funcionado correctamente, mientras que la sección 5 se centra en protocolos que no han funcionado en su totalidad y que requieren modificar Maude-PSL para adecuarlo a la herramienta Maude-NPA. De hecho, los protocolos de la sección 5 son protocolos con propiedades criptográficas de homomorfismo.

2. Preliminares

2.1. Reescribiendo Lógicas y Maude NPA

Teorías de reescritura

Maude es un lenguaje de programación declarativo, donde los programas usan lógica de reescritura. Maude-PSL utiliza un fragmento de Maude como un sub-lenguaje. Cuando especificamos protocolos en Maude-PSL estamos interesados en las teorías ecuacionales.

Las teorías ecuacionales son pares (Σ, E), donde Σ es un conjunto de símbolos llamados operadores y E es un conjunto de ecuaciones entre términos. Un ejemplo sencillo: la teoría ecuacional de los números naturales consiste en una constante 0 representando el número "cero", el operador s representando la función sucesor y un operador + representando la suma en la ecuación, ej:

$$X + 0 = x$$

$$0 + x = x$$

$$s(x) + y = s(x + y)$$

Se pueden usar estas ecuaciones para llegar al razonamiento ecuacional (reemplazando las igualdades). Pero por desgracia este razonamiento es difícil de automatizar. Las ecuaciones pueden aplicarse ya sea de derecha a izquierda y pueden llegar a no saber qué dirección deben ser aplicada en cualquier punto dado. Sin embargo, si orientamos las ecuaciones de dos direcciones a las normas unidireccional de tal manera que las reglas son confluentes y terminantes, entonces podemos utilizar con seguridad las normas unidireccionales en vez de las ecuaciones de dos direcciones para automatizar el razonamiento ecuacional. Las teorías ecuacionales cuyas ecuaciones se han orientado son llamadas teorías de lógica de reescritura, denotandose (Σ, R), donde R es un conjunto de reglas.

Con esta introducción intento explicar de forma muy breve los conceptos de lógica de reescritura, ya que pueden llegar a ser complejos. A continuación intento abarcar los conceptos necesarios para la comprensión de Maude-PSL.

Reglas

Las reglas en lógica de reescritura consisten en un conjunto de pares ordenados {(U1 → V1,..., (Un → Vn)}. El cálculo se lleva a cabo mediante el uso de estas reglas para poder reescribir términos en otros términos. Suponemos que para cada regla u → v, vars(v) ⊆ vars(u).

Dados dos términos t, t ', decimos que se reescribe t a t' con la reglao $u \rightarrow v \in R$ en la posición p con sustitución θ, denotado $t \rightarrow_{p,\theta,u \rightarrow v} t'$ (o $t \rightarrow t'$ si las reglas, la posición y la sustitución se entienden) si y sólo si $u\theta = t_p$, y $v\theta = t'_p$. En este caso, decimos que u es igual a t, que se combina con sustitución θ. Se denota la clausura transitiva por $t \rightarrow^+ t'$ y el cierre reflexivo transitivo por $t \rightarrow^* t'$.

Un término t se llama R-normalizado (o simplemente normalizado) si y sólo si no hay una regla u → v, un término t' y ninguna sustitución θ tal que t → t'. Por lo general, el cómputo en una teoría de reescritura consiste en la normalización de un término t, es decir, volver a escribir t hasta que el término esté R-normalizado.

Concluencia y Terminación

Una de las mayores virtudes de Maude-NPA es la capacidad de razonar acerca de la teoría ecuacional de módulos (mod) en los protocolos criptográficos. Por lo tanto, uno de los propósitos de Maude-PSL es permitir y especificar fácilmente esos módulos. Esto se obtiene gracias a la teoría de lógica de reescritura y al estrechamiento. Sin embargo, con el fin de asegurar los tipos ordenados (order-sorted) en los modulos de la teoría de lógica de reescritura, estos deben tener dos propiedades: terminación y confluencia.

Terminación

Una teoría de reescritura termina si para cada término t, no hay una secuencia infinita de reescritura t →t1→ → → ... → ti →.... En otras palabras, cada secuencia de reescritura a partir de t es de la forma t → → t1. .. → t_n, para algún n□N, y t_n está normalizado.

Confluencia

Una teoría de reescritura es confluente si para cualquier término t, tenemos que $t \rightarrow^*_{R/B} t_1$ y $t \rightarrow^*_{R/B} t_2$, implica que existe un término t' tal que $t_1 \rightarrow^*_{R/B} t'$ y $t_2 \rightarrow^*_{R/B} t'$. Si la teoría es terminante, entonces la teoría de reescritura es confluente si y sólo si cada término t tiene una forma normal única, llamada la forma canónica de t, denotado $t\downarrow_{R/B}$, o simplemente $t\downarrow$ si R y B son conocidos. Escribimos $t \rightarrow!_{R/B} t'$ para indicar que t se vuelve a escribir de forma canónica t'.

Maude NPA

Maude NPA es una herramienta para verificar protocolos criptográficos usando space model, en la que está construida en Maude y sus especificaciones están escritas también en Maude.

2.2. Análisis de Protocolos Criptográficos

El uso de Maude PSL se centra en especificar protocolos criptográficos de forma fácil para el usuario, que después serán verificados por Maude NPA, el objetivo de esta sección es dar una breve introducción a los conceptos criptográficos para entender Maude PSL.

Terminología

- Protocol: define la secuencia de pasos especificando con precisión las acciones de múltiples partes con el fin de alcanzar un objetivo.
- Nonce: Los nonces son normalmente usados para dar una marca de agua única a los mensajes del protocolo de ejecución por cada mensaje generado.
- Ciphertext/Plain: Ciphertext es un mensaje encriptado cuyo significado es imposible de entender y Plain es el mensaje desencriptado y entendible.
- Symmetric key: Se usa para la encriptación y desencriptación, para generar ciphertext desde un Plain.
- Asymmetric key: Es una pareja de keys (x, y) por lo que cada mensaje encriptado con x sólo puede ser desencriptado por y, y viceversa.

- Principal: persona/computador/server capacitado para enviar mensaje por la red, en el caso de PSL serán nuestros roles.

Alice y Bob

Estos dos nombres son los más populares para explicar la notación de especificación de protocolos y es la razón en la que Cholewa se basó para la generación del lenguaje in-put(PSL) para maude-NPA.

Estructura

1. $A \rightarrow B: M_1$

2. $B \rightarrow A: M_2$

3. $A \rightarrow S: M_3$

Donde podemos ver que tenemos tres diferentes Principals ("Alice", "Bob", "Server") y cada uno con su correspondiente mensaje (M_x). Por ejemplo: en la primera línea, Alice envía a Bob el mensaje M_1.

Protocolos Criptográficos

El objetivo principal de Maude-PSL es la especificación de protocolos criptográficos, que han sido verificados por Maude-NPA. Un protocolo criptográfico es un protocolo que garantiza una seguridad sobre datos enviados con los principales involucrados. Estos protocolos intentan garantizar la autentidad de los implicados y los secretos de los términos que son enviados durante el protocolo. Básicamente, si Alice se comunica con Bob, Alice espera que nadie pretenda robar la identidad de Bob, lo mismo al contrario, si Bob está recibiendo mensajes de Alice, él quiere estar seguro de que es Alice quien los envía. Por tanto, el éxito del protocolo se produce sólo en el caso de que los involucrados principales sean quienes dicen ser.

Cuando estudiamos protocolos criptográficos, asumimos dos hechos: la perfecta encriptación y que el intruso tenga control total sobre la red. La perfecta encriptación establece que la única manera de desencriptar y encriptar mensajes es con la clave/contraseña adecuada, dando al intruso control total sobre la red, pudiendo hacer

lo que quiera con los mensajes (recibir, enviar, crear y borrar). El intruso es capaz también de iniciar diferentes sesiones con los roles.

Estándares de los Protocolos

Sabemos la dificultad de verificar protocolos criptográficos, se necesita mucho trabajo para desarrollar herramientas que usen métodos formales (model checking, etc.). Sin embargo, el principal elemento que se precisa para analizar protocolos criptográficos son los modelos matemáticos en protocolos criptográficos. Uno de los más famosos es el π-calculus y Stand space model que es el usado por Maude-NPA. Ejemplo protocolo Needham-Schroeder

1. A → B: e (PKB, A; NA)

2. B → A: e (PKA, NA; NB)

3. A → B: e (PKB, NB)

En notación stand model

A : [+(e(PKB,A;NA)),−(e(PKA,NA;NB)),+(e(PKB,NB))]

B : [−(e(PKB,A;NA)),+(e(PKA,NA;NB)),−(e(PKB,NB))]

Este tipo de notación no contiene noción del tiempo, por lo que no puedes controlar en qué estado en particular está el protocolo, por eso Maude-NPA contiene el argumento "|" con el que se separan mensajes del pasado con mensajes del futuro.

A : [nil | + (e(PK B , A; NA)), −(e(PK A , NA ; NB)), +(e(PK B , NB))]

B : [nil | − (e(PK B , A; NA)), +(e(PK A , NA ; NB)), −(e(PK B , NB))]

En Maude-NPA "nill" representa una lista vacía.

3. Descripción del Lenguaje Maude PSL

Maude-PSL es una herramienta escrita en Python que traduce esta nueva notación para modelar protocolos donde luego verificará

sus propiedades a la notación estándar de la herramienta Maude-NPA.

El siguiente gráfico describe el funcionamiento de Maude-PSL de forma sencilla.

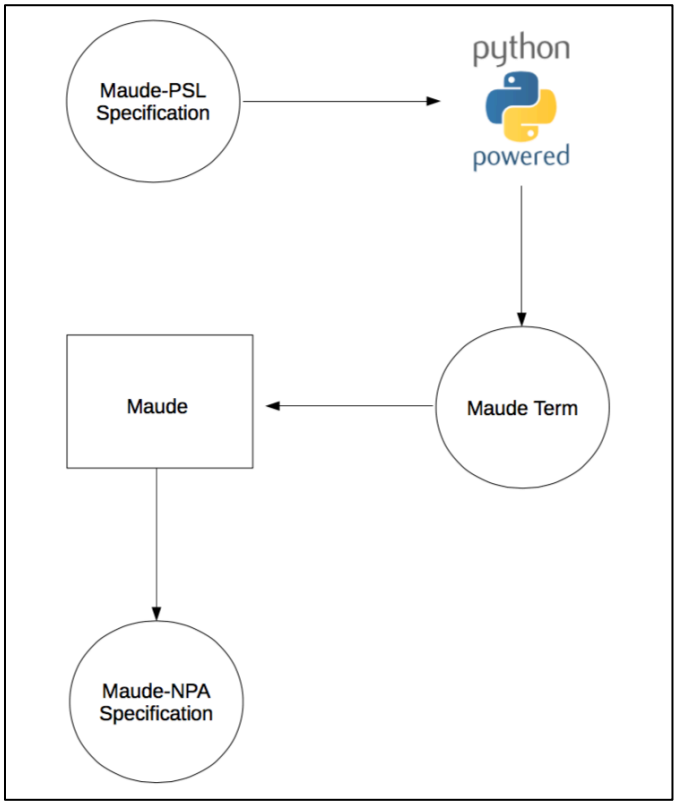

Ilustración 1 Gráfico explicativo Maude PSL

Un protocolo descrito en Maude PSL tiene una estructura formada por cuatro secciones. Cada seción contiene una secuencia de declaraciones, en la que cada declaración finaliza como en Maude, con espacio y punto. Las cuatro secciones son: Theory, para definir la teoría ecuacional; Protocol, usada para definir el protocol; Intruder, que define las capacidades del intruso; y Attack, que contiene los diferentes tipos de ataque.

3.1. Theory

En Theory definimos la teoría algebraica de los protocolos criptográficos. La teoría es un módulo funcional de Maude. En esta parte también diferenciamos el lenguaje en el que cada

especificación va a ser escrita, y también las propiedades de los operadores (p.e. el homomorfismo).

3.2. Protocol

La especificación de un protocolo en PSL costa de cuatro apartados, (e.j protocolo Kao Chow).

1. Protocol's input, especificado por los principals (rol), es el conocimiento inicial de cada rol. En este ejemplo todos saben quien es cada uno.

```
In(A) = ANAME, BNAME, SNAME .
In(B) = ANAME, BNAME, SNAME .
In(S) = ANAME, BNAME, SNAME .
```

Ilustración 2 Maude PSL Protocol

2. Protocol's itself, notación Alice-Bob para especificar el Protocolo (p.e. Kao Chow).

```
1 . A -> S :  ANAME ; BNAME ; na
              |- ANAME ; BNAME ; NAS  .

2 . S -> B :  e(kas, ANAME ; BNAME ; NAS ; kab) ; e(kbs, ANAME ; BNAME ; NAS ; kab)
              |- MA                               ; e(kbs, ANAME ; BNAME ; NAB ; SKB) .

3 . B -> A :  MA                          ; e(SKB, NAB)       ; nb
              |- e(kas, ANAME ; BNAME ; na ; SKA) ; e(SKA, na) ; NBA .

4 . A -> B :  e(SKA, NBA)
              |- e(SKB, nb) .
```

3. Protocol's output, muy similar al input, en este caso es la información importante que tiene que conocer cada rol.

```
Out(A) = na, NBA, SKA .
Out(B) = NAB, nb, SKB .
Out(S) = NAS, NBS, kab .
```

4. Definitions (opcional), usados para darnos una especificación mas rápida de escribir y de entender cuando tenemos términos complejos y complejos de leer.

```
Def(A) =  na := n(ANAME, r),    kas := mkey(ANAME, s) .
Def(B) =  nb := n(ANAME, r1),   kbs := mkey(BNAME, s) .
Def(S) = kas := mkey(ANAME, s), kbs := mkey(BNAME, s),
          kab := seskey(ANAME,BNAME,n(s,r2)) .
```

En protocol's input y output se especifican por "principals", mientras que ifself es la propia especificación del protocolo con la sintax Alice-Bob. (p.e. protocolo Kao-Chow).

Roles

Los roles son el nombre que nosotros usamos para representar cada rol. Por ejemplo, Alice (A) es el rol asociado normalmente al que inicia la comunicación, al igual que Bob (B) es asociado con el rol receptor. En ocasiones también tenemos el Server(S) que se le asocia el rol de neutral.

```
roles A B S .
```

3.3. Intruder

En la sección Intruder de Maude PSL es donde se definen para el intruso todas las capacidades que pueda tener sobre el control de la red, pudiendo interceptar mensajes, destruir mensajes, generar nuevos mensajes en la comunicación entre los roles, etcétera.

Ejemplo protocolo Kao-Chow en PSL

```
Intruder
    var D : Name .
    var r : Fresh .
    var K : Key .
    vars M N : Msg .

            => D, n(i, r), mkey(i, D), mkey(D, i) .
    K, M    => d(K, M), e(K, M) .
    M ; N <=> M, N .
```

Ilustración 3 Maude PSL Intruder

La estructura es sencilla: en la línea K, M => d(K,M), e(K,M), significa que si el intruso sabe K y M puede aprender d(K,M) y e(K,M).

En el caso de <=>, si el intruso sabe la declaración izquierda puede aprender la declaración derecha y viceversa. En la declaración en la que el intruso no tiene nada en el lado izquierdo, significa que el intruso puede generar sus propios nonces, ya que conoce los nombres de los roles.

3.4. Attacks

Es donde especificaremos los diferentes ataques hacia un protocolo, con estos ataques verificamos si el protocolo es seguro contra dicho ataque.

Con los siguientes ataques del protocolo Kao-Chow explicaremos la extructura:

```
Attacks
    0 .
      B executes protocol .
      Subst(B) = ANAME |-> a, BNAME |-> b, SNAME |-> s .

    1 .
      B executes protocol .
      Subst(B) = ANAME |-> a, BNAME |-> b, SNAME |-> s .
      Intruder learns SKB .

    2 .
      B executes protocol .
      Subst(B) = ANAME |-> a, BNAME |-> b, SNAME |-> s .
      without:
        A executes protocol .
        Subst(A) = ANAME |-> a, BNAME |-> b, SNAME |-> s .
```

Ilustración 4 Maude PSL Attacks

En este Attacks tenemos tres tipos distintos de ataque. Por ejemplo, en el 0. "B executes protocol" nos dice que Bob ejecutará su parte del protocolo, a esta línea se le puede especificar una ejecución en particular "B executes up to X", donde X es el número de pasos que deberá ejecutar.

En el attack 1 usa la declaración "Intruder learns SKB" es donde definimos lo que sabe el intruso, en este caso la clave de sesión.

En el attack 2 "without" es decirle qué ejecución específica no tiene que tener lugar durante dicho ataque.

4. Protocolos PSL

4.1. Amended Needham Schroeder

Especificación en notación Alice-Bob del protocolo

A -> B : A

B -> A : E(Kbs:A,Nb0)

A -> S : A,B,Na,E(Kbs:A,Nb0)

S -> A : E(Kas:Na,B,Kab,E(Kbs:Kab,Nb0,A))

A -> B : E(Kbs:Kab,Nb0,A)

B -> A : E(Kab:Nb)

A -> B : E(Kab:Nb-1)

4.1.1. Theory

Types y subtypes en PSL

```
types   SName Name Key Nonce Masterkey Sessionkey .
subtypes Masterkey  Sessionkey < Key .
subtype SName < Name .
subtype Name < Public .
```

Ilustración 5 Ilustraciones protocolo Amended Needham Schroeder

La transformación a Maude NPA cambia la declaración a sorts/subsorts, de hecho, Maude también acepta types/subtypes.

```
sorts SName  Name  Key  Nonce  Masterkey  Sessionkey .
subsorts Masterkey  Sessionkey < Key .
subsorts SName < Name .
subsorts Name < Public .
```

Declaración en PSL de los operadores.

```
op n : Name Fresh -> Nonce .
op a : -> Name . // Alice
op b : -> Name . // Bob
op i : -> Name . // Intruder
op s : ->  SName .
op mkey : Name Name -> Masterkey .
op seskey : Name Name Nonce -> Sessionkey .
op e : Key Msg -> Msg .
op d : Key Msg -> Msg .
op dec : Nonce -> Msg .
op null : -> Msg .
op _;_ : Msg Msg -> Msg [gather (e E)] .
```

La transformación a Maude NPA

```
op n : Name  Fresh -> Nonce [ frozen ] .
op a : -> Name .
op b : -> Name .
op i : -> Name .
op s : -> SName .
op mkey : Name  Name -> Masterkey [ frozen ] .
op seskey : Name  Name  Nonce -> Sessionkey [ frozen ] .
op e : Key  Msg -> Msg [ frozen ] .
op d : Key  Msg -> Msg [ frozen ] .
op dec : Nonce -> Msg [ frozen ] .
op null : -> Msg .
op _;_ : Msg  Msg -> Msg [ gather  (  e  E  )  frozen ] .
```

Ecuaciones algebraicas en PSL

```
eq d(K:Key, e (K:Key, Z:Msg )) = Z:Msg .
eq e(K:Key, d (K:Key, Z:Msg )) = Z:Msg .
```

Ecuaciones algebraicas tras la transformación a Maude.

```
eq d ( K:Key , e ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
eq e ( K:Key , d ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
```

4.1.2. Protocol

Declaración de las variables necesarias, especificación de Inputs de los diferentes roles y las definiciones que nos ayudan a comprender y leer mejor la notación Alice-bob.

```
vars ANAME BNAME SNAME A1NAME A2NAME B2NAME : Name .
var r r0 r1 r2 : Fresh .
vars M1 M2 : Msg .
vars KA KB : Key .
vars NA2 NB NB2 : Nonce .

roles A B S .

Def(A) =      na := n(ANAME, r),      keyAS := mkey(ANAME, SNAME) .
Def(B) =      nb := n(BNAME, r0),     keyBS := mkey(BNAME, SNAME),
                nb1 := n(BNAME, r2) .
Def(S) = keyA2S  := mkey(A2NAME, SNAME), keyB2S := mkey(B2NAME, SNAME),
          sesKey := seskey(A2NAME, B2NAME, n(SNAME, r1)) .

In(A) = ANAME, SNAME, BNAME .
In(B) = SNAME, BNAME .
In(S) = SNAME .
```

Declaración en PSL del protocolo con la especificación Alice-bob, al final del análisis los outputs de los roles.

```
1 . A -> B : ANAME |- A1NAME .

2 . B -> A : e(keyBS, A1NAME ; nb) |- M1 .

3 . A -> S : ANAME  ; BNAME  ; na  ; M1  |-
             A2NAME ; B2NAME ; NA2 ; e(keyB2S, A2NAME ; NB2) .

4 . S -> A : e(keyA2S, NA2 ; B2NAME ; sesKey ; e(keyB2S, sesKey ; NB2 ; A2NAME)) |-
             e(keyAS,  na  ; BNAME  ; KA     ; M2) .

5 . A -> B : M2 |- e(keyBS, KB ; nb ; A1NAME) .

6 . B -> A : e(KB, nb1) |- e(KA, NB) .

7 . A -> B : e(KA, dec(NB)) |- e(KB, dec(nb1)) .

Out(A) = KA .
Out(B) = KB .
Out(S) = sesKey .
```

Transformación del protocolo a Maude NPA: la lectura es difícil y las variables necesarias las declara en el momento con la declaración de tipos que ofrece Maude (variable: tipo), también dificulta la lectura el hecho de que el código haya sido generado automáticamente.

```
eq STRANDS-PROTOCOL =
:: r:Fresh ::
[ nil |
   +(ANAME:Name),
   -(M1:Msg),
   +(ANAME:Name ; BNAME:Name ; n(ANAME:Name, r:Fresh) ; M1:Msg),
   -(e(mkey(ANAME:Name, s), n(ANAME:Name, r:Fresh) ;
   BNAME:Name ; KA:Sessionkey ; M2:Msg)),
   +(M2:Msg),
   -(e(KA:Sessionkey, NB:Nonce)),
   +(e(KA:Sessionkey, dec(NB:Nonce))), nil]  &
:: r1:Fresh ::
[ nil |
   -(A2NAME:Name ; B2NAME:Name ; NA2:Nonce ; e(mkey(B2NAME:Name, s),
    A2NAME:Name ; NB2:Nonce)),
   +(e(mkey(A2NAME:Name, s), NA2:Nonce ; B2NAME:Name ;
   seskey(A2NAME:Name, B2NAME:Name, n(s, r1:Fresh)) ;
   e(mkey(B2NAME:Name, s), seskey(A2NAME:Name, B2NAME:Name,
    n(s, r1:Fresh)) ; NB2:Nonce ; A2NAME:Name))), nil]  &
:: r2:Fresh,r0:Fresh ::
[ nil |
   -(A1NAME:Name),
   +(e(mkey(BNAME:Name, s), A1NAME:Name ; n(BNAME:Name, r0:Fresh))),
   -(e(mkey(BNAME:Name, s), KB:Sessionkey ; n(BNAME:Name, r0:Fresh) ;
    A1NAME:Name)),
   +(e(KB:Sessionkey, n(BNAME:Name, r2:Fresh))),
   -(e(KB:Sessionkey, dec(n(BNAME:Name, r2:Fresh)))), nil] [nonexec].
var LIST : SMsgList-R . var K : IntruderKnowledge . var S : StrandSet .
```

4.1.3. Intruder

Declaración de variables y especificación, donde el intruso tiene el conocimiento de las mismas.

```
var A : Name .
var r : Fresh .
var K : Key .
vars M M1 : Msg .
var N : Nonce .

        => A, s, n(i, r) .
        => mkey(i, A), mkey(A, i) .
        => mkey(i, s), mkey(s, i) .
M, M1 <=> M ; M1 .
K, M   => d(K, M), e(K, M) .
N     <=> dec(N) .
```

Resultado transformación Intruder a Maude NPA a la especificación DOLEVYAO. La lectura es costosa ya que el código ha sido generado automáticamente.

```
eq STRANDS-DOLEVYAO =
:: nil ::
[ nil |
   +(s), nil]  &
:: nil ::
[ nil |
   +(A:Name), nil]  &
:: nil ::
[ nil |
   +(mkey(i, s)), nil]  &
:: nil ::
[ nil |
   +(mkey(i, A:Name)), nil]  &
:: nil ::
[ nil |
   +(mkey(s, i)), nil]  &
:: nil ::
[ nil |
   +(mkey(A:Name, i)), nil]  &
:: nil ::
[ nil |
   -(M1:Msg),
   -(M:Msg),
   +(M:Msg ; M1:Msg), nil]  &
:: nil ::
[ nil |
   -(N:Nonce),
   +(dec(N:Nonce)), nil]  &
:: nil ::
[ nil |
   -(K:Key),
   -(M:Msg),
   +(e(K:Key, M:Msg)), nil]  &
:: nil ::
[ nil |
   -(K:Key),
   -(M:Msg),
   +(d(K:Key, M:Msg)), nil]  &
:: nil ::
[ nil |
   -(dec(N:Nonce)),
   +(N:Nonce), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; M1:Msg),
   +(M:Msg), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; M1:Msg),
   +(M1:Msg), nil]  &
:: r:Fresh ::
[ nil |
   +(n(i, r:Fresh)), nil] [nonexec].
```

4.1.4. Attacks

Ataque 0. donde el intruso se hace pasar por el servidor y, por lo tanto, el protocolo es inseguro.

```
0 .
    B executes protocol .
    Subst(B) = A1NAME |-> a, BNAME |-> b, SNAME |-> s .
```

Ataque 0. en el Maude NPA generado.

```
eq ATTACK-STATE(0)=
:: r2:Fresh,r0:Fresh ::
[ nil,
   -(a),
   +(e(mkey(b, s), a ; n(b, r0:Fresh))),
   -(e(mkey(b, s), KA:Sessionkey ; n(b, r0:Fresh) ; a)),
   +(e(KA:Sessionkey, n(b, r2:Fresh))),
   -(e(KA:Sessionkey, dec(n(b, r2:Fresh)))) | nil]
|| empty
||
nil
||
nil
||
nil[nonexec].
```

4.1.5. Resultados

Protocolo Amended Needham Schroeder key generado por PSL

```
reduce in MAUDE-NPA : summary(5) .
rewrites: 149900532 in 143008ms cpu (142997ms real) (1048189 rewrites/second)
result Summary: States>> 58 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(6) .
rewrites: 802899317 in 632899ms cpu (632859ms real) (1268604 rewrites/second)
result Summary: States>> 140 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(7) .
rewrites: 465238481 in 4075412ms cpu (4085414ms real) (1300426 rewrites/second)
result Summary: States>> 330 Solutions>> 1
==========================================
```

Protocolo Amended Needham Schroeder key original Maude NPA

```
reduce in MAUDE-NPA : summary(5) .
rewrites: 150315237 in 124311ms cpu (124305ms real) (1209179 rewrites/second)
result Summary: States>> 58 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(6) .
rewrites: 803268426 in 600221ms cpu (600186ms real) (1338286 rewrites/second)
result Summary: States>> 140 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(7) .
rewrites: 4787129775 in 3969276ms cpu (3969024ms real) (1206046
    rewrites/second)
result Summary: States>> 330 Solutions>> 1
==========================================
```

4.2. Carlsen's Secret Key

La especificación del protocolo en notación Alice-Bob

(1) A -> B : A,Na

(2) B -> S : A,Na,B,Nb

(3) S -> B : E(Kbs:Kab,Nb,A),E(Kas:Na,B,Kab)

(4) B -> A : E(Kas:Na,B,Kab),E(Kab:Na),Nb'

(5) A -> B : E(Kab:Nb')

4.2.1. Theory

Types y subtypes

```
types Uname Sname Name Key Nonce Masterkey Sessionkey .
subtype Masterkey Sessionkey < Key .
subtype Sname Uname < Name .
subtype Name < Public .
```

Ilustración 6 Ilustraciones protocolo Carlsen's Secret Key

En Maude NPA los types/subtypes es sorts/subsorts

```
sorts Uname  Sname  Name  Key  Nonce  Masterkey  Sessionkey .
subsorts Masterkey  Sessionkey < Key .
subsorts Sname  Uname < Name .
subsorts Name < Public .
```

Propiedades operadores

```
op n : Name Fresh -> Nonce  .
ops a b i : -> Uname [ctor] .
op s : ->  Sname [ctor] .
op mkey : Name Name -> Masterkey [comm] .
op seskey : Name Name Nonce -> Sessionkey .
op e  : Key Msg -> Msg .
op d : Key Msg -> Msg .
op _;_ : Msg Msg -> Msg [gather (e E)] .
```

Con la transformación en Maude NPA

```
op n : Name  Fresh -> Nonce [ frozen ] .
ops a  b  i : -> Uname [ ctor ] .
op s : -> Sname [ ctor ] .
op mkey : Name  Name -> Masterkey [ comm  frozen ] .
op seskey : Name  Name  Nonce -> Sessionkey [ frozen ] .
op e : Key  Msg -> Msg [ frozen ] .
op d : Key  Msg -> Msg [ frozen ] .
op _;_ : Msg  Msg -> Msg [ gather  (  e  E  )  frozen ] .
```

Variables y ecuaciones algebraicas PSL

```
var K : Key .
var Z : Msg .
eq d(K:Key, e (K:Key, Z:Msg )) = Z:Msg .
eq e(K:Key, d (K:Key, Z:Msg )) = Z:Msg .
```

Ecuaciones algebraicas en Maude NPA

```
eq d ( K:Key , e ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
eq e ( K:Key , d ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
```

4.2.2. Protocol

Creación de roles, inputs y Definitions.

```
vars ANAME BNAME ANAME1 ANAME2 BNAME1 : Uname .
var SNAME : Sname .
vars M N MA : Msg .
vars NA NA1 NB NB1 : Nonce .
var K : Key .
var SKA SKB : Sessionkey .
vars r r1 r2 : Fresh .

roles A B S .

Def(A) = na := n(ANAME, r), kas := mkey(ANAME, s) .
Def(B) = nb := n(BNAME, r), kbs := mkey(BNAME, s),
         nb1 := n(BNAME, r1) .
Def(S) = ns := n(SNAME, r2), kab := seskey(ANAME2, BNAME1, ns),
         ksa := mkey(ANAME2, SNAME), ksb := mkey(BNAME1, SNAME) .

In(A) = ANAME, BNAME, SNAME .
In(B) = BNAME, SNAME .
In(S) = SNAME .
```

Especificación del protocolo usando la syntax Alice-Bob

```
1 . A -> B : ANAME  ; na
          |- ANAME1 ; NA  .

2 . B -> S : ANAME1 ; NA  ; BNAME  ; nb
          |- ANAME2 ; NA1 ; BNAME1 ; NB .

3 . S -> B : e(ksb, kab   ; NB ; ANAME2)   ; e(ksa, NA1 ; BNAME1 ; kab)
          |- e(kbs, SKB   ; nb ; ANAME1)   ; MA .

4 . B -> A : MA                        ; e(SKB, NA) ; nb1
          |- e(kas, na ; BNAME ; SKA) ; e(SKA, na) ; NB1 .

5 . A -> B : e(SKA, NB1)
          |- e(SKB, nb1) .

Out(A) = SKA, NB1, na .
Out(B) = SKB, nb1, nb, NA .
Out(S) = kab, NB, NA1 .
```

Resultado transformación protocolo

```
eq STRANDS-PROTOCOL =
:: r:Fresh ::
[ nil |
   +(ANAME:Uname ; n(ANAME:Uname, r:Fresh)),
   -(e(mkey(s, ANAME:Uname), n(ANAME:Uname, r:Fresh) ; BNAME:Uname ;
   SKA:Sessionkey) ; e(SKA:Sessionkey, n(ANAME:Uname, r:Fresh)) ; NB1:Nonce),
   +(e(SKA:Sessionkey, NB1:Nonce)), nil]  &
:: r2:Fresh ::
[ nil |
   -(ANAME2:Uname ; NA1:Nonce ; BNAME1:Uname ; NB:Nonce),
   +(e(mkey(BNAME1:Uname, SNAME:Sname), seskey(ANAME2:Uname,
   BNAME1:Uname, n(SNAME:Sname, r2:Fresh)) ; NB:Nonce ; ANAME2:Uname) ;
   e(mkey(ANAME2:Uname, SNAME:Sname), NA1:Nonce ; BNAME1:Uname ;
   seskey(ANAME2:Uname, BNAME1:Uname, n(SNAME:Sname, r2:Fresh)))), nil]  &
:: r:Fresh,r1:Fresh ::
[ nil |
   -(ANAME1:Uname ; NA:Nonce),
   +(ANAME1:Uname ; NA:Nonce ; BNAME:Uname ; n(BNAME:Uname, r:Fresh)),
   -(e(mkey(s, BNAME:Uname), SKB:Sessionkey ; n(BNAME:Uname, r:Fresh) ;
   ANAME1:Uname) ; MA:Msg),
   +(MA:Msg ; e(SKB:Sessionkey, NA:Nonce) ; n(BNAME:Uname, r1:Fresh)),
   -(e(SKB:Sessionkey, n(BNAME:Uname, r1:Fresh))), nil] [nonexec].
var LIST : SMsgList-R . var K : IntruderKnowledge . var S : StrandSet .
```

4.2.3. Intruder

```
vars ANAME : Uname .
var r : Fresh .
vars M N : Msg .
var K : Key .

=> ANAME, s, mkey(i, s), mkey(i, ANAME) .
K, M => d(K, M), e(K, M) .
M, N <=> M ; N .
```

Resultado transformación Intruder a Maude NPA a la especificación DOLEVYAO

```
eq STRANDS-DOLEVYAO =
:: nil ::
[ nil |
   +(s), nil]  &
:: nil ::
[ nil |
   +(ANAME:Uname), nil]  &
:: nil ::
[ nil |
   +(mkey(i, s)), nil]  &
:: nil ::
[ nil |
   +(mkey(i, ANAME:Uname)), nil]  &
:: nil ::
[ nil |
   -(M:Msg),
   -(N:Msg),
   +(M:Msg ; N:Msg), nil]  &
:: nil ::
[ nil |
   -(K:Key),
   -(M:Msg),
   +(e(K:Key, M:Msg)), nil]  &
:: nil ::
[ nil |
   -(K:Key),
   -(M:Msg),
   +(d(K:Key, M:Msg)), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; N:Msg),
   +(N:Msg), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; N:Msg),
   +(M:Msg), nil] [nonexec].
```

Attack 0 en PSL

```
0 .
   B executes protocol .
   Subst(B) = ANAME1 |-> a , BNAME |-> b, SNAME |-> s .
```

Ataque en Maude NPA

```
eq ATTACK-STATE(0)=
:: r:Fresh,r1:Fresh ::
[ nil,
   -(a ; NA:Nonce),
   +(a ; NA:Nonce ; b ; n(b, r:Fresh)),
   -(e(mkey(b, s), SKB:Sessionkey ; n(b, r:Fresh) ; a) ; MA:Msg),
   +(MA:Msg ; e(SKB:Sessionkey, NA:Nonce) ; n(b, r1:Fresh)),
   -(e(SKB:Sessionkey, n(b, r1:Fresh))) | nil]
|| empty
||
nil
||
nil
||
nil[nonexec].
```

4.2.5. Resultados

Para la comprobación de que el protocolo generado es igual al original usamos diferentes comandos, como "run" "summary" en Maude NPA.

Protocolo Carlsen's secret key generado

```
reduce in MAUDE-NPA : summary(3) .
rewrites: 22880079 in 26013ms cpu (26011ms real) (879542 rewrites/second)
result Summary: States>> 17 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(4) .
rewrites: 81490288 in 75524ms cpu (75519ms real) (1078988 rewrites/second)
result Summary: States>> 38 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(5) .
rewrites: 258674180 in 203088ms cpu (203075ms real) (1273700 rewrites/second)
result Summary: States>> 62 Solutions>> 1
==========================================
```

Protocolo Carlsen's secret key original

```
reduce in MAUDE-NPA : summary(3) .
rewrites: 22111119 in 23081ms cpu (23080ms real) (957960 rewrites/second)
result Summary: States>> 17 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(4) .
rewrites: 79369180 in 68260ms cpu (68258ms real) (1162743 rewrites/second)
result Summary: States>> 38 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(5) .
rewrites: 255339177 in 187283ms cpu (187269ms real) (1363381 rewrites/second)
result Summary: States>> 62 Solutions>> 1
==========================================
```

4.3. Denning-Sacco

Anotación Alice-Bob

(1) A -> B : A,B

(2) S -> A : E(Kas:B,Kab,T,E(Kbs:A,Kab,T))

(3) A -> B : E(Kbs:A,Kab,T)

4.3.1. Theory

Parte types/subtypes

```
types  UName SName Name Key Nonce Masterkey Sessionkey .
subtype Masterkey Sessionkey < Key .
subtype SName UName < Name .
subtype Name < Public .
```

Ilustración 7 Ilustraciones protocolo Denning Sacco

En la transformación a Maude NPA la notación types/subtypes será sorts/subsorts

```
sorts UName  SName  Name  Key  Nonce  Masterkey  Sessionkey .
subsorts Masterkey  Sessionkey < Key .
subsorts SName  UName < Name .
subsorts Name < Public .
```

Parte propiedades operaciones

```
op n : Name Fresh -> Nonce .
op t : Name Fresh -> Nonce .
ops a b i : -> UName [ctor] .
op s : ->  SName [ctor] .
op mkey : Name Name -> Masterkey .
op seskey : Name Name Nonce -> Sessionkey .
op e : Key Msg -> Msg .
op d : Key Msg -> Msg .
op _;_ : Msg Msg -> Msg [gather (e E)] .
```

Transformación operaciones a Maude NPA

```
op n : Name  Fresh -> Nonce [ frozen ] .
op t : Name  Fresh -> Nonce [ frozen ] .
ops a  b  i : -> UName [ ctor ] .
op s : -> SName [ ctor ] .
op mkey : Name  Name -> Masterkey [ frozen ] .
op seskey : Name  Name  Nonce -> Sessionkey [ frozen ] .
op e : Key  Msg -> Msg [ frozen ] .
op d : Key  Msg -> Msg [ frozen ] .
op _;_ : Msg  Msg -> Msg [ gather  (  e  E  )  frozen ] .
```

Variables y ecuaciones PSL

```
var K : Key .
var Z : Msg .
eq d(K, e(K, Z)) = Z .
eq e(K, d(K, Z)) = Z .
```

Resultado en Maude NPA

```
eq d ( K:Key , e ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
eq e ( K:Key , d ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
```

4.3.2. Protocol

Creación de roles, inputs y Definitions.

```
vars ANAME BNAME AS BS AB : UName .
var SNAME : SName .
vars r r' : Fresh .
vars TSA TSB : Nonce .
var K : Key .
vars SK SK1 SKA SKB : Sessionkey .
vars M N X Y : Msg .

roles A B S .

In(A) = ANAME, BNAME, SNAME .
In(B) = BNAME, SNAME .
In(S) = SNAME .

Def(A) = kas := mkey(ANAME, SNAME) .
Def(B) = kbs := mkey(BNAME, SNAME) .
Def(S) = kab := seskey(AS, BS, n(SNAME,r)), ts := t(SNAME, r'),
         ksa := mkey(AS, SNAME), ksb := mkey(BS, SNAME) .
```

Protocolo usando la syntax Alice-Bob

```
1 . A -> S : ANAME ; BNAME |- AS ; BS .

2 . S -> A : e(ksa, BS    ; kab ; ts  ; e(ksb, AS ; kab ; ts))
          |- e(kas, BNAME ; SKA ; TSA ; M) .

3 . A -> B : M |- e(kbs, AB ; SKB ; TSB) .


Out(A) = SKA, TSA .
Out(B) = SKB, TSB .
Out(S) = kab, ts  .
```

Resultado transformación protocolo

```
eq STRANDS-PROTOCOL =
:: nil ::
[ nil |
   +(ANAME:UName ; BNAME:UName),
   -(e(mkey(ANAME:UName, SNAME:SName), BNAME:UName ; SKA:Sessionkey ;
   TSA:Nonce ; M:Msg)),
   +(M:Msg), nil]  &
:: nil ::
[ nil |
   -(e(mkey(BNAME:UName, SNAME:SName), AB:UName ; SKB:Sessionkey ;
   TSB:Nonce)), nil]  &
:: r:Fresh,r':Fresh ::
[ nil |
   -(AS:UName ; BS:UName),
   +(e(mkey(AS:UName, SNAME:SName), BS:UName ; seskey(AS:UName, BS:UName,
   n(SNAME:SName, r:Fresh)) ; t(SNAME:SName, r':Fresh) ;
    e(mkey(BS:UName, SNAME:SName), AS:UName ;
    seskey(AS:UName, BS:UName, n(SNAME:SName, r:Fresh)) ;
    t(SNAME:SName, r':Fresh)))), nil] [nonexec].
var LIST : SMsgList-R . var K : IntruderKnowledge . var S : StrandSet .
```

4.3.3. Intruder

Declaración de las variables usadas por el intruso y la especificación del conocimiento que tiene del protocolo.

```
var D : Name .
var r : Fresh .
var K : Key .
vars M N : Msg .

        => D, n(i, r), t(i, r) .
        => mkey(i, D), mkey(D, i) .
K, M  => d(K, M), e(K, M) .
M, N <=> M ; N .
```

Transformación a Maude NPA, una de las ventajas de PSL es su syntax sobre DOLEVYAO que usa Maude NPA.

```
eq STRANDS-DOLEVYAO =
:: nil ::
[ nil |
   +(D:Name), nil]  &
:: nil ::
[ nil |
   +(mkey(i, D:Name)), nil]  &
:: nil ::
[ nil |
   +(mkey(D:Name, i)), nil]  &
:: nil ::
[ nil |
   -(M:Msg),
   -(N:Msg),
   +(M:Msg ; N:Msg), nil]  &
:: nil ::
[ nil |
   -(K:Key),
   -(M:Msg),
   +(e(K:Key, M:Msg)), nil]  &
:: nil ::
```

```
[ nil |
   -(K:Key),
   -(M:Msg),
   +(d(K:Key, M:Msg)), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; N:Msg),
   +(N:Msg), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; N:Msg),
   +(M:Msg), nil]  &
:: r:Fresh ::
[ nil |
   +(n(i, r:Fresh)), nil]  &
:: r:Fresh ::
[ nil |
   +(t(i, r:Fresh)), nil] [nonexec].
```

4.3.4. Attacks

Attack 0 PSL

```
0 .
    A executes protocol .
    Subst(A) = ANAME |-> a , BNAME |-> b, SNAME |-> s .
```

Transformación Attack 0 a Maude NPA

```
 eq ATTACK-STATE(0)=
:: nil ::
[ nil,
   +(a ; b),
   -(e(mkey(a, s), b ; SKA:Sessionkey ; TSA:Nonce ; M:Msg)),
   +(M:Msg) | nil]
|| empty
||
nil
||
nil
||
nil[nonexec].
```

4.3.5. Resultados

Para la comprobación que el protocolo generado es igual al original usamos diferentes comandos, como "run" "summary" en Maude NPA.

Protocolo Carlsen's secret key generado

```
reduce in MAUDE-NPA : summary(3) .
rewrites: 5842967 in 6396ms cpu (6397ms real) (913477 rewrites/second)
result Summary: States>> 3 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(4) .
rewrites: 8090208 in 8764ms cpu (8762ms real) (923060 rewrites/second)
result Summary: States>> 6 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(5) .
rewrites: 14551320 in 16349ms cpu (16348ms real) (890042 rewrites/second)
result Summary: States>> 9 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(6) .
rewrites: 20504803 in 21897ms cpu (21893ms real) (936404 rewrites/second)
result Summary: States>> 8 Solutions>> 2
==========================================
reduce in MAUDE-NPA : summary(7) .
rewrites: 23657896 in 26685ms cpu (26685ms real) (886539 rewrites/second)
result Summary: States>> 8 Solutions>> 2
==========================================
reduce in MAUDE-NPA : summary(8) .
rewrites: 17694367 in 19737ms cpu (19734ms real) (896496 rewrites/second)
result Summary: States>> 7 Solutions>> 3
==========================================
reduce in MAUDE-NPA : summary(9) .
rewrites: 6193637 in 6332ms cpu (6331ms real) (978087 rewrites/second)
result Summary: States>> 6 Solutions>> 4
```

Protocolo Carlsen's secret key original Maude NPA

```
reduce in MAUDE-NPA : summary(3) .
rewrites: 4915794 in 4900ms cpu (4899ms real) (1003160 rewrites/second)
result Summary: States>> 3 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(4) .
rewrites: 6336237 in 6488ms cpu (6489ms real) (976547 rewrites/second)
result Summary: States>> 6 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(5) .
rewrites: 11949168 in 12248ms cpu (12246ms real) (975540 rewrites/second)
result Summary: States>> 9 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(6) .
rewrites: 16965074 in 16441ms cpu (16438ms real) (1031874 rewrites/second)
result Summary: States>> 8 Solutions>> 2
==========================================
reduce in MAUDE-NPA : summary(7) .
rewrites: 18718665 in 19333ms cpu (19332ms real) (968213 rewrites/second)
result Summary: States>> 8 Solutions>> 2
==========================================
reduce in MAUDE-NPA : summary(8) .
rewrites: 14707631 in 14560ms cpu (14559ms real) (1010076 rewrites/second)
result Summary: States>> 7 Solutions>> 3
==========================================
reduce in MAUDE-NPA : summary(9) .
rewrites: 5361824 in 4700ms cpu (4701ms real) (1140742 rewrites/second)
result Summary: States>> 6 Solutions>> 4
```

4.4. Diffie Hellman

Anotación en Alice-Bob:

(1) A --> B: A ; B ; exp(g,N_A)

(2) B --> A: A ; B ; exp(g,N_A)

(3) A --> B: enc(exp(exp(g,N_B),N_A),secret(A,B))

4.4.1. Theory

Declaración types/subtypes

```
types Name Nonce NeNonceSet Gen Exp Key GenvExp Secret .
subtype Gen Exp < GenvExp .
subtype Exp < Key .
subtype Name < Public . // This is quite relevant and necessary
subtype Gen < Public . // This is quite relevant and necessary
subtype Nonce < NeNonceSet .
```

Ilustración 8 Ilustraciones protocolo Difiie Hellman

En la transformación a Maude NPA la notación types/subtypes será sorts/subsorts.

```
sorts Name  Nonce  NeNonceSet  Gen  Exp  Key  GenvExp  Secret .
subsorts Gen  Exp < GenvExp .
subsorts Exp < Key .
subsorts Name < Public .
subsorts Gen < Public .
subsorts Nonce < NeNonceSet .
subsorts Name  NeNonceSet  Key  GenvExp  Secret < Msg .
```

Parte propiedades operaciones

```
op sec : Name Fresh -> Secret .
op n : Name Fresh -> Nonce .
ops a b i : -> Name .
op e : Key Msg -> Msg .
op d : Key Msg -> Msg .
op exp : GenvExp NeNonceSet -> Exp .
op g : -> Gen .
op _*_ : NeNonceSet NeNonceSet -> NeNonceSet [assoc comm] .
op _;_ : Msg Msg -> Msg [gather (e E)] .
```

```
op sec : Name  Fresh -> Secret [ frozen ] .
op n : Name  Fresh -> Nonce [ frozen ] .
ops a  b  i : -> Name .
op e : Key  Msg -> Msg [ frozen ] .
op d : Key  Msg -> Msg [ frozen ] .
op exp : GenvExp  NeNonceSet -> Exp [ frozen ] .
op g : -> Gen .
op _*_ : NeNonceSet  NeNonceSet -> NeNonceSet [ assoc  comm  frozen ] .
op _;_ : Msg  Msg -> Msg [ gather  (  e  E  )  frozen ] .
```

Ecuaciones y variables PSL

```
var W : Gen .
vars Y Z : NeNonceSet .
var K : Key .
var M : Msg .
eq exp(exp(W,Y),Z) = exp(W, Y * Z) .
eq e(K,d(K,M)) = M .
eq d(K,e(K,M)) = M .
```

Transformación de las ecuaciones en Maude NPA

```
eq exp ( exp ( W:Gen , Y:NeNonceSet ) , Z:NeNonceSet ) =
exp ( W:Gen , Y:NeNonceSet * Z:NeNonceSet ) [ variant ] .
eq e ( K:Key , d ( K:Key , M:Msg ) ) = M:Msg [ variant ] .
eq d ( K:Key , e ( K:Key , M:Msg ) ) = M:Msg [ variant ] .
```

4.4.2. Protocol

Creación de roles, inputs y Definitions.

```
vars ANAME BNAME ANAME1 : Name .
vars r r' r1 : Fresh .
vars XEA XEB : Exp .
var S : Secret .

roles A B .

In(A) = ANAME, BNAME .
In(B) = BNAME .

Def(A) = na := n(ANAME, r), secret := sec(ANAME, r') .
Def(B) = nb := n(BNAME, r1) .
```

Especificación del protocolo usando Alice-Bob

```
1 . A -> B : ANAME  ; BNAME ; exp(g, na)
          |- ANAME1 ; BNAME ; XEB .
2 . B -> A : ANAME1 ; BNAME ; exp(g, nb)
          |- ANAME  ; BNAME ; XEA .
3 . A -> B : e(exp(XEA, na), secret)
          |- e(exp(XEB, nb), S) .

Out(A) = na, exp(g, na), XEB, secret .
Out(B) = nb, exp(g, nb), XEA, S .
```

Protocolo en Maude NPA

```
eq STRANDS-PROTOCOL =
:: r1:Fresh ::
[ nil |
   -(ANAME1:Name ; BNAME:Name ; XEB:Exp),
   +(ANAME1:Name ; BNAME:Name ; exp(g, n(BNAME:Name, r1:Fresh))),
   -(e(exp(XEB:Exp, n(BNAME:Name, r1:Fresh)), S:Secret)), nil]  &
:: r:Fresh,r':Fresh ::
[ nil |
   +(ANAME:Name ; BNAME:Name ; exp(g, n(ANAME:Name, r:Fresh))),
   -(ANAME:Name ; BNAME:Name ; XEA:Exp),
   +(e(exp(XEA:Exp, n(ANAME:Name, r:Fresh)),
   sec(ANAME:Name, r':Fresh))), nil] [nonexec].
var LIST : SMsgList-R . var K : IntruderKnowledge . var S : StrandSet .
```

4.4.3. Intruder

Declaración de las variables usadas por el intruso y la especificación del conocimiento que tiene del protocolo.

```
vars NAME : Name .
var K : Key .
vars M M1 M2 : Msg .
vars NS1 NS2 : NeNonceSet .
var GE : GenvExp .
var r  : Fresh .
          => n(i, r), g, NAME .
M1 ; M2 <=> M1, M2 .
K, M     => e(K, M), d(K, M) .
NS1, NS2 => NS1 * NS2 .
GE, NS1  => exp(GE, NS1) .
```

Transformación a Maude NPA, una de las ventajas de PSL es su syntax sobre DOLEVYAO que usa Maude NPA.

```
eq STRANDS-DOLEVYAO =
:: nil ::
[ nil |
   +(g), nil]  &
:: nil ::
[ nil |
   +(ANAME:Name), nil]  &
:: nil ::
[ nil |
   -(M2:Msg),
   -(M1:Msg),
   +(M1:Msg ; M2:Msg), nil]  &
:: nil ::
[ nil |
   -(K:Key),
   -(M:Msg),
   +(e(K:Key, M:Msg)), nil]  &
:: nil ::
[ nil |
   -(K:Key),
   -(M:Msg),
   +(d(K:Key, M:Msg)), nil]  &
:: nil ::
```

```
[ nil |
   -(NS2:NeNonceSet),
   -(NS1:NeNonceSet),
   +(NS1:NeNonceSet * NS2:NeNonceSet), nil]  &
:: nil ::
[ nil |
   -(GE:GenvExp),
   -(NS:NeNonceSet),
   +(exp(GE:GenvExp, NS:NeNonceSet)), nil]  &
:: nil ::
[ nil |
   -(M1:Msg ; M2:Msg),
   +(M1:Msg), nil]  &
:: nil ::
[ nil |
   -(M1:Msg ; M2:Msg),
   +(M2:Msg), nil]  &
:: r:Fresh ::
[ nil |
   +(n(i, r:Fresh)), nil] [nonexec].
```

4.4.4. Attack

Ataque 0

```
0 .
    B executes protocol .
    Subst(B) = ANAME1 |-> a, BNAME |-> b, S |-> sec(a, r') .
    without:
        Subst(A) = ANAME |-> a, BNAME |-> b .
        A executes protocol .
```

Ataque 0 en Maude NPA

```
eq ATTACK-STATE(0)=
:: r1:Fresh ::
[ nil,
   -(ANAME1:Name ; b ; XEB:Exp),
   +(ANAME1:Name ; b ; exp(g, n(b, r1:Fresh))),
   -(e(exp(XEB:Exp, n(b, r1:Fresh)), sec(a, r':Fresh))) | nil]
|| empty
||
nil
||
nil
|| never
((:: r:Fresh,r':Fresh ::
[ nil,
   +(a ; b ; exp(g, n(a, r:Fresh))),
   -(a ; b ; XEA:Exp),
   +(e(exp(XEA:Exp, n(a, r:Fresh)), sec(a, r':Fresh))) | nil] ) & S || K )[nonexec].
```

4.4.5. Resultados

Ejecución (Diffie Hellman) del Maude generado

```
reduce in MAUDE-NPA : summary(9) .
rewrites: 29116998 in 136049ms cpu (139159ms real) (214017 rewrites/second)
result Summary: States>> 7 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(10) .
rewrites: 16650019 in 54521ms cpu (55581ms real) (305385 rewrites/second)
result Summary: States>> 4 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(11) .
rewrites: 6587773 in 19421ms cpu (19872ms real) (339203 rewrites/second)
result Summary: States>> 3 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(12) .
rewrites: 2105773 in 5921ms cpu (6036ms real) (355610 rewrites/second)
result Summary: States>> 2 Solutions>> 2
==========================================
```

Ejecución (Diffie Hellman) del Maude NPA original

```
reduce in MAUDE-NPA : summary(9) .
rewrites: 85010122 in 113431ms cpu (113715ms real) (749439 rewrites/second)
result Summary: States>> 7 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(10) .
rewrites: 20270674 in 25790ms cpu (25860ms real) (785985 rewrites/second)
result Summary: States>> 4 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(11) .
rewrites: 7951351 in 11962ms cpu (12002ms real) (664667 rewrites/second)
result Summary: States>> 3 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(12) .
rewrites: 2621579 in 4482ms cpu (4499ms real) (584830 rewrites/second)
result Summary: States>> 2 Solutions>> 2
==========================================
```

4.5. ISO-5 pass Authentication

Protocolo en syntax Alice-Bob

(1) A -> B : A,Na

(2) B -> S : A,Na,B,Nb'

(3) S -> B : E(Kbs:Nb',Kab,A), E(Kas:Na,Kab,B)

(4) B -> A : E(Kas:Na,Kab,B),E(Kab:Nb,Na)

(5) A -> B : E(Kab:Na,Nb)

4.5.1. Theory

Types/subtypes en PSL.

```
types UName SName Name Key Nonce Masterkey Sessionkey .
subtype Masterkey Sessionkey < Key .
subtype SName UName < Name .
subtype Name < Public .
```

Ilustración 9 Ilustraciones protocolo ISO-5

En la transformación a Maude NPA la notación types/subtypes será sorts/subsorts.

```
sorts UName  SName  Name  Key  Nonce  Masterkey  Sessionkey .
subsorts Masterkey  Sessionkey < Key .
subsorts SName  UName < Name .
subsorts Name < Public .
```

Operadores PSL.

```
op n : Name Fresh -> Nonce  .
ops a b i : -> UName [ctor] .
op s : ->  SName [ctor]  .
op mkey : Name Name -> Masterkey [comm] .
op seskey : Name Name Nonce -> Sessionkey .
op e : Key Msg -> Msg .
op d : Key Msg -> Msg .
op _;_ : Msg Msg -> Msg [gather (e E)] .
```

Operadores Maude NPA generado.

```
op n : Name  Fresh -> Nonce [ frozen ] .
ops a  b  i : -> UName [ ctor ] .
op s : -> SName [ ctor ] .
op mkey : Name  Name -> Masterkey [ comm  frozen ] .
op seskey : Name  Name  Nonce -> Sessionkey [ frozen ] .
op e : Key  Msg -> Msg [ frozen ] .
op d : Key  Msg -> Msg [ frozen ] .
op _;_ : Msg  Msg -> Msg [ gather  (  e  E  )  frozen ] .
```

Variables y ecuaciones

```
var K : Key .
var Z : Msg .
eq d(K, e(K, Z)) = Z .
eq e(K, d(K, Z)) = Z .
```

Ecuaciones generadas

```
eq d ( K:Key , e ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
eq e ( K:Key , d ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
```

4.5.2. Protocol

Declaración de variables, input y definiciones.

```
vars ANAME BNAME AB AS BS : UName .
var SNAME : SName .
vars K : Key .
vars SKA SKB : Sessionkey .
vars r r' : Fresh .
vars NAB NAS NBA NBS : Nonce .
var MA : Msg .

roles A B S .

Def(A) = na := n(ANAME, r), kas := mkey(ANAME, s) .

In(A)  = ANAME, BNAME, SNAME .

Def(B) = nb := n(BNAME, r), nb1 := n(BNAME, r'), kbs := mkey(BNAME,s) .
In(B)  = BNAME, SNAME .

Def(S) = ksa := mkey(AS, s), ksb := mkey(BS, s),
         kab := seskey(AS, BS, n(s,r)) .
In(S) = SNAME .
```

Especificación del protocolo usando Alice-Bob

```
1 . A -> B : ANAME  ; na
       |- AB     ; NAB .

2 . B -> S : AB ; NAB ; BNAME  ; nb1
       |- AS ; NAS ; BS     ; NBS .

3 . S -> B : e(ksb, NBS ; kab  ; AS)  ; e(ksa, NAS ; kab ; BS)
       |- e(kbs, nb1 ; SKB  ; AB)     ; MA .

4 . B -> A : MA                        ; e(SKB, nb  ; NAB)
       |- e(kas, na ; SKA ; BNAME ) ; e(SKA, NBA ; na) .

5 . A -> B : e(SKA, na  ; NBA)
       |- e(SKB, NAB ; nb) .

Out(A) = na, NBA, SKA .
Out(B) = NAB, nb, nb1, SKB .
Out(S) = NAS, NBS, kab .
```

Especificación del protocolo transformado a Maude NPA

```
eq STRANDS-PROTOCOL =
:: r:Fresh ::
[ nil |
   +(ANAME:UName ; n(ANAME:UName, r:Fresh)),
   -(e(mkey(s, ANAME:UName), n(ANAME:UName, r:Fresh) ; SKA:Sessionkey ;
    BNAME:UName) ; e(SKA:Sessionkey, NBA:Nonce ; n(ANAME:UName, r:Fresh))),
   +(e(SKA:Sessionkey, n(ANAME:UName, r:Fresh) ; NBA:Nonce)), nil]  &
:: r:Fresh ::
[ nil |
   -(AS:UName ; NAS:Nonce ; BS:UName ; NBS:Nonce),
   +(e(mkey(s, BS:UName), NBS:Nonce ; seskey(AS:UName, BS:UName, n(s, r:Fresh)) ;
    AS:UName) ; e(mkey(s, AS:UName), NAS:Nonce ; seskey(AS:UName, BS:UName,
    n(s, r:Fresh)) ; BS:UName)), nil]  &
:: r:Fresh,r':Fresh ::
[ nil |
   -(AB:UName ; NAB:Nonce),
   +(AB:UName ; NAB:Nonce ; BNAME:UName ; n(BNAME:UName, r':Fresh)),
   -(e(mkey(s, BNAME:UName), n(BNAME:UName, r':Fresh) ; SKB:Sessionkey ;
   AB:UName) ; MA:Msg),
   +(MA:Msg ; e(SKB:Sessionkey, n(BNAME:UName, r:Fresh) ; NAB:Nonce)),
   -(e(SKB:Sessionkey, NAB:Nonce ; n(BNAME:UName, r:Fresh))), nil] [nonexec].
var LIST : SMsgList-R . var K : IntruderKnowledge . var S : StrandSet .
```

4.5.3. Intruder

Declaración de las variables usadas por el intruso y la especificación del conocimiento que tiene del protocolo.

```
vars C D : Name .
var K : Key .
vars M N : Msg .
var r : Fresh .

=> C, s, mkey(i, D), mkey(i, s) .
K, M => d(K, M), e(K, M) .
M, N <=> M ; N .
```

Transformación a Maude NPA, una de las ventajas de PSL es su syntax sobre DOLEVYAO que usa Maude NPA.

```
eq STRANDS-DOLEVYAO =
:: nil ::
[ nil |
   +(s), nil]  &
:: nil ::
[ nil |
   +(C:Name), nil]  &
:: nil ::
[ nil |
   +(mkey(i, s)), nil]  &
:: nil ::
[ nil |
   +(mkey(i, D:Name)), nil]  &
:: nil ::
[ nil |
   -(M:Msg),
   -(N:Msg),
   +(M:Msg ; N:Msg), nil]  &
:: nil ::
```

```
[ nil |
   -(K:Key),
   -(M:Msg),
   +(e(K:Key, M:Msg)), nil]  &
:: nil ::
[ nil |
   -(K:Key),
   -(M:Msg),
   +(d(K:Key, M:Msg)), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; N:Msg),
   +(N:Msg), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; N:Msg),
   +(M:Msg), nil] [nonexec].
```

4.5.4. Attack

Nos centramos en el Attack-0 que es el necesario para saber si el protocolo funciona correctamente.

```
0 .
     B executes protocol .
     Subst(B) = AB |-> a , BNAME |-> b, SNAME |-> s .
```

Attack 0 del Maude NPA generado.

```
eq ATTACK-STATE(0)=
:: r:Fresh,r':Fresh ::
[ nil,
   -(a ; NAB:Nonce),
   +(a ; NAB:Nonce ; b ; n(b, r':Fresh)),
   -(e(mkey(b, s), n(b, r':Fresh) ; SKB:Sessionkey ; a) ; MA:Msg),
   +(MA:Msg ; e(SKB:Sessionkey, n(b, r:Fresh) ; NAB:Nonce)),
   -(e(SKB:Sessionkey, NAB:Nonce ; n(b, r:Fresh))) | nil]
|| empty
||
nil
||
nil
||
nil[nonexec].
```

4.5.5. Resultados

Ejecución (Kao Chow) del Maude NPA generado por PSL

```
reduce in MAUDE-NPA : summary(3) .
rewrites: 17766988 in 20193ms cpu (20192ms real) (879847 rewrites/second)
result Summary: States>> 16 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(4) .
rewrites: 68663591 in 65000ms cpu (64996ms real) (1056361 rewrites/second)
result Summary: States>> 33 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(5) .
rewrites: 232799706 in 172006ms cpu (171996ms real) (1353433 rewrites/second)
result Summary: States>> 68 Solutions>> 1
==========================================
```

Ejecución (Kao Chow) del Maude NPA original

```
reduce in MAUDE-NPA : summary(3) .
rewrites: 17778258 in 20461ms cpu (20463ms real) (868873 rewrites/second)
result Summary: States>> 16 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(4) .
rewrites: 67564186 in 63395ms cpu (63395ms real) (1065749 rewrites/second)
result Summary: States>> 33 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(5) .
rewrites: 228903618 in 168730ms cpu (168721ms real) (1356622 rewrites/second)
result Summary: States>> 68 Solutions>> 1
==========================================
```

4.6. Needham- Schroeder public key

Anotación Alice-Bob:

(1) A -> B : pk(B, A ; na)

(2) B -> A : pk(A, N1 ; nb)

(3) A -> B : pk(B, N2)

4.6.1. Theory

Types y subtypes PSL.

```
types Name Nonce Enc .
subtype Name < Public .
```

Ilustración 10 Ilustraciones Needham-schroeder public key

En la transformación a Maude NPA la notación types/subtypes será sorts/subsorts.

```
sorts Name  Nonce  Enc .
subsorts Name < Public .
```

También se puede utilizar en Maude NPA los tipos types y subtypes.

Parte propiedades operaciones

```
op pk : Name Msg -> Enc .
op sk : Name Msg -> Enc .
op n : Name Fresh -> Nonce .
op _;_ : Msg Msg -> Msg [ctor gather(e E)] .
ops a b i : -> Name [ctor] .
```

Transformación a Maude NPA.

```
op pk : Name  Msg -> Enc [ frozen ] .
op sk : Name  Msg -> Enc [ frozen ] .
op n : Name  Fresh -> Nonce [ frozen ] .
op _;_ : Msg  Msg -> Msg [ ctor  gather  (  e  E  )  frozen ] .
ops a  b  i : -> Name [ ctor ] .
```

Ecuaciones y variables PSL.

```
var X : Name .
var Z : Msg .
eq pk(X, sk(X,Z)) = Z .
eq sk(X, pk(X,Z)) = Z .
```

Ecuaciones transformación a Maude NPA.

```
eq pk ( X:Name , sk ( X:Name , Z:Msg ) ) = Z:Msg [ variant ] .
eq sk ( X:Name , pk ( X:Name , Z:Msg ) ) = Z:Msg [ variant ] .
```

4.6.2. Protocol

Declaración de variables, roles, input y definitions.

```
vars AName BName : Name .
vars N1 N2 : Nonce .
vars r1 r2 : Fresh .

roles A B .

In(A) = AName, BName .
In(B) = AName, BName .

Def(A) = na := n(AName, r1) .//na := na1 , na1 := na,
 na2 := pk(BName, sk(AName, na1)) . //na := n(AName, r1) .
Def(B) = nb := n(BName, r2) .
```

Especificación del protocolo usando la syntax Alice-Bob

```
 1 . A -> B : pk(BName, AName ; na)  |- pk(BName, AName ; N1) .
 2 . B -> A : pk(AName, N1 ; nb)     |- pk(AName, na    ; N2) .
 3 . A -> B : pk(BName, N2)          |- pk(BName, nb) .

 Out(A) = na, N2 .
 Out(B) = nb, N1 .
```

Resultado en Maude NPA

```
eq STRANDS-PROTOCOL =
:: r1:Fresh ::
[ nil |
   +(pk(BName:Name, AName:Name ; n(AName:Name, r1:Fresh))),
   -(pk(AName:Name, n(AName:Name, r1:Fresh) ; N2:Nonce)),
   +(pk(BName:Name, N2:Nonce)), nil]  &
:: r2:Fresh ::
[ nil |
   -(pk(BName:Name, AName:Name ; N1:Nonce)),
   +(pk(AName:Name, N1:Nonce ; n(BName:Name, r2:Fresh))),
   -(pk(BName:Name, n(BName:Name, r2:Fresh))), nil] [nonexec].
var LIST : SMsgList-R . var K : IntruderKnowledge . var S : StrandSet .
```

4.6.3. Intruder

Variables y estructura de lo que conoce el intruso.

```
vars X Y : Msg .
var A : Name .
var r1 : Fresh .

       => n(i,r1) .
X ; Y <=> X, Y .
X      => sk(i,X) .
X, A   => pk(A, X) .
```

Transformación a Maude NPA, una de las ventajas de PSL es su syntax sobre DOLEVYAO que usa Maude NPA.

```
eq STRANDS-DOLEVYAO =
:: nil ::
[ nil |
   -(X:Msg),
   +(sk(i, X:Msg)), nil]  &
:: nil ::
[ nil |
   -(Y:Msg),
   -(X:Msg),
   +(X:Msg ; Y:Msg), nil]  &
:: nil ::
```

```
[ nil |
   -(A:Name),
   -(X:Msg),
   +(pk(A:Name, X:Msg)), nil]  &
:: nil ::
[ nil |
   -(X:Msg ; Y:Msg),
   +(X:Msg), nil]  &
:: nil ::
[ nil |
   -(X:Msg ; Y:Msg),
   +(Y:Msg), nil]  &
:: r1:Fresh ::
[ nil |
   +(n(i, r1:Fresh)), nil] [nonexec].
```

4.6.4. Attack

Ataque 0. En el que es ejecutado por Bob y el intruso averigua el nonce de bob.

```
0 .
    B executes protocol .
    Subst(B) = AName |-> a, BName |-> b .
    Intruder learns nb .
```

Ataque 0 en Maude NPA.

```
eq ATTACK-STATE(0)=
:: r2:Fresh ::
[ nil,
   -(pk(b, a ; N1:Nonce)),
   +(pk(a, N1:Nonce ; n(b, r2:Fresh))),
   -(pk(b, n(b, r2:Fresh))) | nil]
||
n(b, r2:Fresh) inI
||
nil
||
nil
||
nil[nonexec].
```

4.6.5. Resultado

Ejecución (Needham–Schroeder public key) del Maude NPA generado por PSL.

```
reduce in MAUDE-NPA : summary(7) .
rewrites: 826413 in 3126ms cpu (3242ms real) (264311 rewrites/second)
result Summary: States>> 4 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(8) .
rewrites: 1143618 in 4499ms cpu (4709ms real) (254174 rewrites/second)
result Summary: States>> 4 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(9) .
rewrites: 1421462 in 5882ms cpu (6129ms real) (241641 rewrites/second)
result Summary: States>> 2 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(10) .
rewrites: 421299 in 1874ms cpu (1954ms real) (224788 rewrites/second)
result Summary: States>> 1 Solutions>> 1
==========================================
```

Ejecución (Needham–Schroeder public key) del Maude NPA original.

```
reduce in MAUDE-NPA : summary(7) .
rewrites: 908919 in 784ms cpu (784ms real) (1159263 rewrites/second)
result Summary: States>> 4 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(8) .
rewrites: 1278820 in 1160ms cpu (1157ms real) (1102362 rewrites/second)
result Summary: States>> 4 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(9) .
rewrites: 1665305 in 1756ms cpu (1757ms real) (948291 rewrites/second)
result Summary: States>> 2 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(10) .
rewrites: 677206 in 636ms cpu (636ms real) (1064722 rewrites/second)
result Summary: States>> 1 Solutions>> 1
==========================================
```

4.7. Otway Rees

Sintasis en Alice-Bob:

(1) A -> B : M , A,B , E(Kas:Na,M,A,B)

(2) B -> S : M , A , B , E(Kas:Na,M,A,B) , E(Kbs:Nb,M,A,B)

(3) S ->B : M,E(Kas:Na,Kab),E(Kbs:Nb,Kab)

(4) B -> A : M,E(Kas:Na,Kab)

4.7.1. Theory

Types y subtypes.

```
types UName SName Name Key Nonce Masterkey Sessionkey .
subtypes Masterkey Sessionkey < Key .
subtypes SName UName < Name < Public .
```

Ilustración 11 Ilustraciones Otway Rees

Transformación a Maude NPA.

```
sorts UName  SName  Name  Key  Nonce  Masterkey  Sessionkey .
subsorts Masterkey  Sessionkey < Key .
subsorts SName  UName < Name < Public .
```

Operadores PSL

```
op n : Name Fresh -> Nonce .
ops a b i : -> UName .
op s : ->  SName .
op mkey : Name Name -> Masterkey .
op seskey : Name Name Nonce -> Sessionkey .
op e  : Key Msg -> Msg . //encryption
op d : Key Msg -> Msg . //decryption
op _;_ : Msg Msg -> Msg [gather (e E)] .
```

Transformación operadores

```
op n : Name  Fresh -> Nonce [ frozen ] .
ops a  b  i : -> UName .
op s : -> SName .
op mkey : Name  Name -> Masterkey [ frozen ] .
op seskey : Name  Name  Nonce -> Sessionkey [ frozen ] .
op e : Key  Msg -> Msg [ frozen ] .
op d : Key  Msg -> Msg [ frozen ] .
op _;_ : Msg  Msg -> Msg [ gather  (  e  E  )  frozen ] .
subsorts UName  SName  Name  Key  Nonce  Masterkey  Sessionkey < Msg .
op _$;_ : Msg Msg -> Msg [ctor gather(e E) frozen].
```

Declaración variables y equaciones

```
var K : Key .
var Z : Msg .
eq d(K, e(K, Z)) = Z .
eq e(K, d(K, Z)) = Z .
```

Equaciones en Maude NPA

```
eq d ( K:Key , e ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
eq e ( K:Key , d ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
```

4.7.2. Protocol

Declaración de variables, roles, input y definitions.

```
vars ANAME BNAME : UName .
var SNAME : SName .
vars r r' r'' rM : Fresh .
vars RA CB CS RB : Nonce .
vars M1 MA : Msg .
var KCA KCB : Sessionkey .

roles A B S .

Def(A) = c   := n(ANAME, rM), ra := n(ANAME, r),
         skA := mkey(ANAME, SNAME) .
Def(B) = rb  := n(BNAME, r'),
         skB := mkey(BNAME, SNAME) .
Def(S) = skA := mkey(ANAME, SNAME),
         skB := mkey(BNAME, SNAME),
          kc := seskey(ANAME, BNAME,
                       n(SNAME, r'')) .


In(A) = ANAME, BNAME, SNAME .
In(B) = ANAME, BNAME, SNAME .
In(S) = ANAME, BNAME, SNAME .
```

Protocolo con sintaxis Alice-Bob

```
1 . A -> B :  c  ; ANAME ; BNAME ; e(skA, ra ; c ;
                                    ANAME ; BNAME)
          |- CB ; ANAME ; BNAME ; M1 .


2 . B -> S :  CB ; ANAME ; BNAME ;
              M1 ;
              e(skB, rb ; CB ; ANAME ; BNAME)
          |- CS ; ANAME ; BNAME ;
              e(skA, RA ; CS ; ANAME ; BNAME) ;
              e(skB, RB ; CS ; ANAME ; BNAME) .


3 . S -> B :  CS ; e(skA, RA ; kc) ;
              e(skB, RB ; kc)
          |- CB ; MA ;
              e(skB, rb ; KCB) .


4 . B -> A :  CB ; MA
          |- c  ; e(skA, ra ; KCA) .
```

Resultado de la transformación a Maude NPA

```
eq STRANDS-PROTOCOL =
:: r':Fresh ::
[ nil |
   -(CB:Nonce ; ANAME:UName ; BNAME:UName ; M1:Msg),
   +(CB:Nonce ; ANAME:UName ; BNAME:UName ; M1:Msg ;
   e(mkey(BNAME:UName, SNAME:SName), n(BNAME:UName, r':Fresh) ;
   CB:Nonce ; ANAME:UName ; BNAME:UName)),
   -(CB:Nonce ; MA:Msg ; e(mkey(BNAME:UName, SNAME:SName),
   n(BNAME:UName, r':Fresh) ; KCB:Sessionkey)),
   +(CB:Nonce ; MA:Msg), nil]  &
:: r'':Fresh ::
[ nil |
   -(CS:Nonce ; ANAME:UName ; BNAME:UName ; e(mkey(ANAME:UName, SNAME:SName),
    RA:Nonce ; CS:Nonce ; ANAME:UName ; BNAME:UName) ; e(mkey(BNAME:UName, SNAME:SName),
     RB:Nonce ; CS:Nonce ; ANAME:UName ; BNAME:UName)),
   +(CS:Nonce ; e(mkey(ANAME:UName, SNAME:SName), RA:Nonce ;
   seskey(ANAME:UName, BNAME:UName, n(SNAME:SName, r'':Fresh))) ;
    e(mkey(BNAME:UName, SNAME:SName), RB:Nonce ; seskey(ANAME:UName,
     BNAME:UName, n(SNAME:SName, r'':Fresh)))), nil]  &
:: r:Fresh,rM:Fresh ::
[ nil |
   +(n(ANAME:UName, rM:Fresh) ; ANAME:UName ; BNAME:UName ;
   e(mkey(ANAME:UName, SNAME:SName), n(ANAME:UName, r:Fresh) ;
   n(ANAME:UName, rM:Fresh) ; ANAME:UName ; BNAME:UName)),
   -(n(ANAME:UName, rM:Fresh) ; e(mkey(ANAME:UName, SNAME:SName),
    n(ANAME:UName, r:Fresh) ; KCA:Sessionkey)), nil] [nonexec].
var LIST : SMsgList-R . var K : IntruderKnowledge . var S : StrandSet .
```

4.7.3. Intruder

Valores conocidos por el intruso.

```
vars P : UName .
var S : SName .
vars K : Key .
vars N M : Msg .

=> s, P, mkey(i, s) .
M, N <=> M ; N .
K, M => d(K, M), e(K, M) .
P => mkey(P, i), mkey(i, P) .
```

Transformación a Maude NPA, una de las ventajas de PSL es su syntax sobre DOLEVYAO que usa Maude NPA.

```
eq STRANDS-DOLEVYAO =
:: nil ::
[ nil |
   +(s), nil]  &
:: nil ::
[ nil |
   +(P:UName), nil]  &
:: nil ::
[ nil |
   +(mkey(i, s)), nil]  &
:: nil ::
[ nil |
   -(M:Msg),
   -(N:Msg),
   +(M:Msg ; N:Msg), nil]  &
:: nil ::
[ nil |
   -(P:UName),
   +(mkey(i, P:UName)), nil]  &
:: nil ::
```

```
[ nil |
   -(P:UName),
   +(mkey(P:UName, i)), nil]  &
:: nil ::
[ nil |
   -(K:Key),
   -(M:Msg),
   +(e(K:Key, M:Msg)), nil]  &
:: nil ::
[ nil |
   -(K:Key),
   -(M:Msg),
   +(d(K:Key, M:Msg)), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; N:Msg),
   +(N:Msg), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; N:Msg),
   +(M:Msg), nil] [nonexec].
```

4.7.4. Attacks

Ataque 0 en Maude PSL

```
0 .
     A executes protocol .
     Subst(A) = ANAME |-> a, BNAME |-> b, SNAME |-> s .
```

Ataque 0 en Maude NPA

```
eq ATTACK-STATE(0)=
:: r:Fresh,rM:Fresh ::
[ nil,
   +(n(a, rM:Fresh) ; a ; b ; e(mkey(a, s),
   n(a, r:Fresh) ; n(a, rM:Fresh) ; a ; b)),
   -(n(a, rM:Fresh) ; e(mkey(a, s), n(a, r:Fresh) ;
    KCA:Sessionkey)) | nil]
|| empty
||
nil
||
nil
||
nil[nonexec].
```

4.7.5. Resultados

Ejecución (Otway-Rees) del Maude NPA generado por PSL

```
reduce in MAUDE-NPA : summary(2) .
rewrites: 14109081 in 14024ms cpu (14025ms real) (1006003 rewrites/second)
result Summary: States>> 6 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(3) .
rewrites: 40476564 in 41122ms cpu (41121ms real) (984290 rewrites/second)
result Summary: States>> 15 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(4) .
rewrites: 97856933 in 93953ms cpu (93944ms real) (1041542 rewrites/second)
result Summary: States>> 45 Solutions>> 1
==========================================
```

Ejecución (Otway-Rees) del Maude NPA original

```
reduce in MAUDE-NPA : summary(2) .
rewrites: 13098224 in 12272ms cpu (12281ms real) (1067259 rewrites/second)
result Summary: States>> 6 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(3) .
rewrites: 36806362 in 34810ms cpu (34808ms real) (1057344 rewrites/second)
result Summary: States>> 15 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(4) .
rewrites: 88814747 in 78864ms cpu (78860ms real) (1126162 rewrites/second)
result Summary: States>> 45 Solutions>> 1
==========================================
```

4.8. Wide Mouthed Frog

Especificación del protocolo en notación Alice-Bob:

A -> S : A,E(Kas:B,Kab)

S -> B : E(Kbs:A,Kab)

A -> B : A,E(Kab:M)

4.8.1. Theory

Types y subtypes en PSL.

```
types  UName SName Name Key Nonce  Masterkey Sessionkey .
subtype Masterkey  Sessionkey < Key .
subtype SName UName < Name .
subtype Name < Public . // This is quite relevant and necessary
```

Ilustración 12 Ilustraciones protocolo Wide Mouthed Frog

Resultado types/subtypes.

```
sorts UName  SName  Name  Key  Nonce  Masterkey  Sessionkey .
subsorts Masterkey  Sessionkey < Key .
subsorts SName  UName < Name .
subsorts Name < Public .
```

Operadores en PSL.

```
op n  : Name Fresh -> Nonce  .
ops a b i : -> UName .
op s : ->  SName .
op mkey : Name Name -> Masterkey  .
op seskey : Name Name Nonce -> Sessionkey  .
op e : Key Msg -> Msg  .
op d : Key Msg -> Msg  .
op p : Msg -> Msg  .
op _;_ : Msg Msg -> Msg [gather (e E)] .
```

Resultado transformación operadores.

```
op n : Name  Fresh -> Nonce [ frozen ] .
ops a  b  i : -> UName .
op s : -> SName .
op mkey : Name  Name -> Masterkey [ frozen ] .
op seskey : Name  Name  Nonce -> Sessionkey [ frozen ] .
op e : Key  Msg -> Msg [ frozen ] .
op d : Key  Msg -> Msg [ frozen ] .
op p : Msg -> Msg [ frozen ] .
op _;_ : Msg  Msg -> Msg [ gather  (  e  E  )  frozen ] .
```

Variables y ecuaciones algebraicas PSL.

```
var K : Key .
var Z : Msg .
eq d(K, e(K, Z)) = Z .
eq e(K, d(K, Z)) = Z .
```

Transformación ecuaciones en Maude NPA.

```
eq d ( K:Key , e ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
eq e ( K:Key , d ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
```

4.8.2. Protocol

Creación de roles, inputs y Definitions en PSL.

```
vars ANAME BNAME SNAME : Name .
vars KAS KBS KAB KBA : Key .
vars r1 r2 r3 : Fresh .
vars MA MB : Msg .

roles A B S .

Def(A) = keyAS := mkey(ANAME, SNAME),
         keyAB := seskey(ANAME, BNAME, n(ANAME, r2)) .
Def(B) = keyBS := mkey(BNAME, SNAME) .
Def(S) = keyAS := mkey(ANAME, SNAME),
         keyBS := mkey(BNAME, SNAME) .

In(A) = ANAME, BNAME, SNAME, MA .
In(B) = ANAME, BNAME, SNAME .
In(S) = ANAME, BNAME, SNAME .
```

Protocolo con notación Alice-Bob

```
1 . A -> S :  ANAME ; e(keyAS, BNAME ; keyAB)
          |- ANAME ; e(keyAS, BNAME ; KAB) .

2 . S -> B :  e(keyBS, ANAME ; KAB)
          |- e(keyBS, ANAME ; KBA) .

3 . A -> B :  ANAME ; e(keyAB, MA)
          |- ANAME ; e(KBA,   MB) .

Out(A) = keyAB .
Out(S) = KAB .
Out(B) = KBA .
```

Resultado de la transformación del protocolo en Maude NPA

```
:: nil ::
[ nil |
   -(e(mkey(BNAME:UName, s), ANAME:UName ; KBA:Sessionkey)),
   -(ANAME:UName ; e(KBA:Sessionkey, MA:Nonce)), nil]  &
:: nil ::
[ nil |
   -(ANAME:UName ; e(mkey(ANAME:UName, s), BNAME:UName ; KAB:Sessionkey)),
   +(e(mkey(BNAME:UName, s), ANAME:UName ; KAB:Sessionkey)), nil]  &
:: r2:Fresh ::
[ nil |
   +(ANAME:UName ; e(mkey(ANAME:UName, s), BNAME:UName ;
   seskey(ANAME:UName, BNAME:UName, n(ANAME:UName, r2:Fresh)))),
   +(ANAME:UName ; e(seskey(ANAME:UName, BNAME:UName,
   n(ANAME:UName, r2:Fresh)), MA:Nonce)), nil] [nonexec].
var LIST : SMsgList-R . var K : IntruderKnowledge . var S : StrandSet .
```

4.8.3. Intruder

```
var r : Fresh .
vars X Y : Msg .
var K : Key .
var N : Name .
var S : SName .
      => N, seskey(i, N, n(i, r)), mkey(i, S) .
X, Y <=> X ; Y .
K, X  => e(K, X), d(K, X) .
```

Transformación a Maude NPA, una de las ventajas de PSL es su syntax sobre DOLEVYAO que usa Maude NPA.

```
eq STRANDS-DOLEVYAO =
:: nil ::
[ nil |
   +(s), nil]  &
:: nil ::
[ nil |
   +(A:UName), nil]  &
:: nil ::
[ nil |
   +(mkey(i, s)), nil]  &
:: nil ::
[ nil |
   +(mkey(i, A:UName)), nil]  &
:: nil ::
[ nil |
   +(mkey(A:UName, i)), nil]  &
:: nil ::
[ nil |
   -(Y:Msg),
   -(X:Msg),
   +(X:Msg ; Y:Msg), nil]  &
:: nil ::
```

```
[ nil |
   -(K:Key),
   -(X:Msg),
   +(e(K:Key, X:Msg)), nil]  &
:: nil ::
[ nil |
   -(K:Key),
   -(X:Msg),
   +(d(K:Key, X:Msg)), nil]  &
:: nil ::
[ nil |
   -(X:Msg ; Y:Msg),
   +(X:Msg), nil]  &
:: nil ::
[ nil |
   -(X:Msg ; Y:Msg),
   +(Y:Msg), nil] [nonexec].
```

4.8.4. Attacks

Ataque 0 PSL.

```
0 .
    A executes protocol .
    Subst(A) = ANAME |-> a, BNAME |-> b, SNAME |-> s .
```

Ataque 0 en Maude NPA

```
 eq ATTACK-STATE(0)= (
:: nil ::
[ nil,
   -(e(mkey(b, s), a ; seskey(a, b, n(a, r2:Fresh)))),
   -(a ; e(seskey(a, b, n(a, r2:Fresh)), MA:Nonce)) | nil]  &
:: r2:Fresh ::
[ nil,
   +(a ; e(mkey(a, s), b ; seskey(a, b, n(a, r2:Fresh)))),
   +(a ; e(seskey(a, b, n(a, r2:Fresh)), MA:Nonce)) | nil] )
|| empty
||
nil
||
nil
||
nil[nonexec].
```

4.8.5. Resultado

Ejecución (Wide Mouthed Frog) del Maude NPA generado por PSL

```
reduce in MAUDE-NPA : summary(1) .
rewrites: 1613115 in 2384ms cpu (2382ms real) (676599 rewrites/second)
result Summary: States>> 5 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(2) .
rewrites: 5921619 in 6300ms cpu (6302ms real) (939880 rewrites/second)
result Summary: States>> 13 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(3) .
rewrites: 27953751 in 19913ms cpu (19909ms real) (1403776 rewrites/second)
result Summary: States>> 27 Solutions>> 1
==========================================
```

Ejecución (Wide Mouthed Frog) del Maude NPA original.

```
reduce in MAUDE-NPA : summary(1) .
rewrites: 1601967 in 2064ms cpu (2060ms real) (776098 rewrites/second)
result Summary: States>> 5 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(2) .
rewrites: 5894895 in 5612ms cpu (5614ms real) (1050343 rewrites/second)
result Summary: States>> 13 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(3) .
rewrites: 27889420 in 18369ms cpu (18366ms real) (1518275 rewrites/second)
result Summary: States>> 27 Solutions>> 1
==========================================
```

4.9. Kao Chow

Protocolo de autenticación que no es susceptible a los ataques de Neuman Stubblebine.

Anotación Alice-Bob:

(1) A -> S : A , B , Na

(2) S -> B : E(Kas:A,B,Na,Kab) , E(Kbs:A,B,Na,Kab)

(3) B -> A : E(Kas:A,B,Na,Kab) , E(Kab:Na) , Nb

(4) A -> B : E(Kab:Nb)

4.9.1. Theory

Parte Types/subtypes

```
types  UName SName Name Key Nonce Masterkey Sessionkey .
subtype Masterkey Sessionkey < Key .
subtype SName UName < Name .
subtype Name < Public .
```

Ilustración 13 Ilustraciones protocolo Kao Chow

En la transformación a Maude NPA la notación types/subtypes será sorts/subsorts.

```
sorts UName  SName  Name  Key  Nonce  Masterkey  Sessionkey .
subsorts Masterkey  Sessionkey < Key .
subsorts SName  UName < Name .
subsorts Name < Public .
```

También se puede utilizar en Maude NPA los tipos types y subtypes.

Parte propiedades operaciones

```
op n : Name Fresh -> Nonce  .
op t : Name Fresh -> Nonce  .
ops a b i : -> UName [ctor] .
op s : ->  SName [ctor] .
op mkey : Name Name -> Masterkey .
op seskey : Name Name Nonce -> Sessionkey .
op e : Key Msg -> Msg .
op d : Key Msg -> Msg .
op _;_ : Msg Msg -> Msg [gather (e E)] .
```

Transformación a Maude NPA

```
op n : Name  Fresh -> Nonce [ frozen ] .
op t : Name  Fresh -> Nonce [ frozen ] .
ops a  b  i : -> UName [ ctor ] .
op s : -> SName [ ctor ] .
op mkey : Name  Name -> Masterkey [ frozen ] .
op seskey : Name  Name  Nonce -> Sessionkey [ frozen ] .
op e : Key  Msg -> Msg [ frozen ] .
op d : Key  Msg -> Msg [ frozen ] .
op _;_ : Msg  Msg -> Msg [ gather  (  e  E  )  frozen ] .
subsorts UName SName Name  Key  Nonce  Masterkey  Sessionkey < Msg .
op _$;_ : Msg Msg -> Msg [ctor gather(e E) frozen].
```

Ecuaciones y variables PSL

```
var K : Key .
var Z : Msg .
eq d(K, e(K, Z)) = Z .
eq e(K, d(K, Z)) = Z .
```

Ecuaciones algebraicas transformadas a Maude NPA

```
eq d ( K:Key , e ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
eq e ( K:Key , d ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
```

4.9.2. Protocol

Creación de roles, inputs y Definitions.

```
vars ANAME BNAME : UName .
var SNAME : SName .
vars MA M N : Msg .
vars r r1 r2 : Fresh .
vars SKA SKB : Sessionkey .
var NAS NBS NBA NAB : Nonce .

roles A B S .

In(A) = ANAME, BNAME, SNAME .
In(B) = ANAME, BNAME, SNAME .
In(S) = ANAME, BNAME, SNAME .

Def(A) =  na := n(ANAME, r),    kas := mkey(ANAME, s) .
Def(B) =  nb := n(ANAME, r1),   kbs := mkey(BNAME, s) .
Def(S) = kas := mkey(ANAME, s), kbs := mkey(BNAME, s),
         kab := seskey(ANAME,BNAME,n(s,r2)) .
```

Especificación del protocolo usando la syntax Alice-Bob

```
1 . A -> S :  ANAME ; BNAME ; na
          |- ANAME ; BNAME ; NAS  .

2 . S -> B :  e(kas, ANAME ; BNAME ; NAS ; kab) ; e(kbs, ANAME ; BNAME ; NAS ; kab)
          |- MA                                 ; e(kbs, ANAME ; BNAME ; NAB ; SKB) .

3 . B -> A :  MA                          ; e(SKB, NAB)       ; nb
          |- e(kas, ANAME ; BNAME ; na ; SKA) ; e(SKA, na) ; NBA .

4 . A -> B :  e(SKA, NBA)
          |- e(SKB, nb) .
```

Transformación a Maude-NPA, los tipos de las variables, las asigna al lado de las variables.

```
eq STRANDS-PROTOCOL =
:: r:Fresh ::
[ nil |
   +(ANAME:UName ; BNAME:UName ; n(ANAME:UName, r:Fresh)),
   -(e(mkey(ANAME:UName, s), ANAME:UName ; BNAME:UName ; n(ANAME:UName, r:Fresh) ;
    SKA:Sessionkey) ; e(SKA:Sessionkey, n(ANAME:UName, r:Fresh)) ; NBA:Nonce),
   +(e(SKA:Sessionkey, NBA:Nonce)), nil]  &
:: r1:Fresh ::
[ nil |
   -(MA:Msg ; e(mkey(BNAME:UName, s), ANAME:UName ; BNAME:UName ; NAB:Nonce ;
    SKB:Sessionkey)),
   +(MA:Msg ; e(SKB:Sessionkey, NAB:Nonce) ; n(ANAME:UName, r1:Fresh)),
   -(e(SKB:Sessionkey, n(ANAME:UName, r1:Fresh))), nil]  &
:: r2:Fresh ::
[ nil |
   -(ANAME:UName ; BNAME:UName ; NAS:Nonce),
   +(e(mkey(ANAME:UName, s), ANAME:UName ; BNAME:UName ; NAS:Nonce ;
   seskey(ANAME:UName, BNAME:UName, n(s, r2:Fresh))) ; e(mkey(BNAME:UName, s),
   ANAME:UName ; BNAME:UName ; NAS:Nonce ; seskey(ANAME:UName, BNAME:UName,
    n(s, r2:Fresh)))), nil] [nonexec].
var LIST : SMsgList-R . var K : IntruderKnowledge . var S : StrandSet .
```

Output, el conocimiento final de los roles, como las claves de sesión, etc.

```
Out(A) = na, NBA, SKA .
Out(B) = NAB, nb, SKB .
Out(S) = NAS, NBS, kab .
```

4.9.3. Intruder

```
var D : Name .
var r : Fresh .
var K : Key .
vars M N : Msg .

        => D, n(i, r), mkey(i, D), mkey(D, i) .
K, M    => d(K, M), e(K, M) .
M ; N <=> M, N .
```

Transformación a Maude NPA, una de las ventajas de PSL es su syntax sobre DOLEVYAO que usa Maude NPA.

```
eq STRANDS-DOLEVYAO =
:: nil ::
[ nil |
   +(D:Name), nil]  &
:: nil ::
[ nil |
   +(mkey(i, D:Name)), nil]  &
:: nil ::
[ nil |
   +(mkey(D:Name, i)), nil]  &
:: nil ::
[ nil |
   -(M:Msg),
   -(N:Msg),
   +(M:Msg ; N:Msg), nil]  &
:: nil ::
[ nil |
   -(K:Key),
   -(M:Msg),
   +(e(K:Key, M:Msg)), nil]  &
:: nil ::
[ nil |
   -(K:Key),
   -(M:Msg),
   +(d(K:Key, M:Msg)), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; N:Msg),
   +(N:Msg), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; N:Msg),
   +(M:Msg), nil]  &
:: r:Fresh ::
[ nil |
   +(n(i, r:Fresh)), nil] [nonexec].
```

4.9.4. Attacks

Nos centramos en el Attack-0 que es el necesario para saber si el protocolo funciona correctamente.

```
Attacks
    0 .
        B executes protocol .
        Subst(B) = ANAME |-> a, BNAME |-> b, SNAME |-> s .
```

El ataque generado a Maude NPA

```
eq ATTACK-STATE(0)=
:: r1:Fresh ::
[ nil,
   -(MA:Msg ; e(mkey(b, s), a ; b ; NAB:Nonce ; SKB:Sessionkey)),
   +(MA:Msg ; e(SKB:Sessionkey, NAB:Nonce) ; n(a, r1:Fresh)),
   -(e(SKB:Sessionkey, n(a, r1:Fresh))) | nil]
|| empty
||
nil
||
nil
||
nil[nonexec].
```

4.9.5. Resultados

Ejecución (Kao Chow) del Maude NPA generado por PSL

```
==========================================
reduce in MAUDE-NPA : summary(3) .
rewrites: 22335379 in 25821ms cpu (25822ms real) (864987 rewrites/second)
result Summary: States>> 16 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(4) .
rewrites: 67435227 in 66112ms cpu (66105ms real) (1020012 rewrites/second)
result Summary: States>> 30 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(5) .
rewrites: 203661921 in 166790ms cpu (166786ms real) (1221064 rewrites/second)
result Summary: States>> 52 Solutions>> 1
==========================================
```

Ejecución (Kao Chow) del Maude NPA original

```
==========================================
reduce in MAUDE-NPA : summary(3) .
rewrites: 21596475 in 23597ms cpu (23594ms real) (915202 rewrites/second)
result Summary: States>> 16 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(4) .
rewrites: 65131023 in 60035ms cpu (60032ms real) (1084870 rewrites/second)
result Summary: States>> 30 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(5) .
rewrites: 200417040 in 153309ms cpu (153299ms real) (1307270 rewrites/second)
result Summary: States>> 52 Solutions>> 1
==========================================
```

4.10. Yahalom

Definición protocolo en anotación Alice-Bob:

A -> B : A,Na

B -> S : B,E(Kbs:A,Na,Nb)

S -> A : E(Kas:B,Kab,Na,Nb),E(Kbs:A,Kab)

A -> B : E(Kbs:A,Kab),E(Kab:Nb)

4.10.1. Theory

Types y subtypes PSL

```
types  Uname Sname Name Key Nonce Masterkey Sessionkey .
subtype Masterkey Sessionkey < Key .
subtype Sname Uname < Name .
subtype Name < Public .
```

Ilustración 14 Ilustraciones protocolo Yahalom

Resultado en Maude NPA.

```
sorts Uname  Sname  Name  Key  Nonce  Masterkey  Sessionkey .
subsorts Masterkey  Sessionkey < Key .
subsorts Sname  Uname < Name .
subsorts Name < Public .
```

Operaciones PSL.

```
op n : Name Fresh -> Nonce .
ops a b i : -> Uname [ctor] .
op s : ->  Sname [ctor] .
op mkey : Name Name -> Masterkey .
op seskey : Name Name Nonce -> Sessionkey .
op e : Key Msg -> Msg .
op d : Key Msg -> Msg .
op _;_ : Msg Msg -> Msg [gather (e E)] .
```

Operadores Maude NPA.

```
op n : Name  Fresh -> Nonce [ frozen ] .
ops a  b  i : -> Uname [ ctor ] .
op s : -> Sname [ ctor ] .
op mkey : Name  Name -> Masterkey [ frozen ] .
op seskey : Name  Name  Nonce -> Sessionkey [ frozen ] .
op e : Key  Msg -> Msg [ frozen ] .
op d : Key  Msg -> Msg [ frozen ] .
op _;_ : Msg  Msg -> Msg [ gather  (  e  E  )  frozen ] .
```

Ecuaciones PSL.

```
eq d(K:Key, e(K:Key, Z:Msg )) = Z:Msg .
eq e(K:Key, d(K:Key, Z:Msg )) = Z:Msg .
```

Ecuaciones en Maude NPA

```
eq d ( K:Key , e ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
eq e ( K:Key , d ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
```

4.10.2. Protocol

```
vars ANAME BNAME : Uname .
var SNAME : Sname .
vars NA NBA NAS NBS : Nonce .
var  r : Fresh .
var M N MB : Msg .
var SKA SKB : Sessionkey .
var D : Name .
var K : Key .

roles A B S .

In(A) = ANAME, BNAME, SNAME .
In(B) = ANAME, BNAME, SNAME .
In(S) = ANAME, BNAME, SNAME .

Def(A) = na := n(ANAME, r), kas := mkey(ANAME,s) .

Def(B) = nb := n(BNAME, r), kbs := mkey(BNAME,s) .

Def(S) = kas := mkey(ANAME, s), kbs := mkey(BNAME, s),
    kab := seskey(ANAME , BNAME , n(s,r)) .
```

Declaración del protocolo en Alice-Bob.

```
1 . A -> B :  ANAME ; na
            |- ANAME ; NA .

2 . B -> S :  BNAME ; e(kbs, ANAME ; NA  ; nb)
            |- BNAME ; e(kbs, ANAME ; NAS ; NBS) .

3 . S -> A :  e(kas, BNAME ; kab  ; NAS ; NBS) ; e(kbs, ANAME ; kab)
            |- e(kas, BNAME ; SKA  ; na  ; NBA) ; MB .

4 . A -> B :  MB                  ; e(SKA, NBA)
            |- e(kbs, ANAME ; SKB) ; e(SKB, nb) .

Out(A) = na, NBA, SKA .
Out(B) = NA, nb, SKB .
Out(S) = NA, NBS, kab .
```

Resultado de la transformación del protocolo en Maude NPA

```
eq STRANDS-PROTOCOL =
:: r:Fresh ::
[ nil |
   +(ANAME:Uname ; n(ANAME:Uname, r:Fresh)),
   -(e(mkey(ANAME:Uname, s), BNAME:Uname ; SKA:Sessionkey ;
    n(ANAME:Uname, r:Fresh) ; NBA:Nonce) ; MB:Msg),
   +(MB:Msg ; e(SKA:Sessionkey, NBA:Nonce)), nil]  &
:: r:Fresh ::
[ nil |
   -(ANAME:Uname ; NA:Nonce),
   +(BNAME:Uname ; e(mkey(BNAME:Uname, s), ANAME:Uname ;
   NA:Nonce ; n(BNAME:Uname, r:Fresh))),
   -(e(mkey(BNAME:Uname, s), ANAME:Uname ; SKB:Sessionkey) ;
    e(SKB:Sessionkey, n(BNAME:Uname, r:Fresh))), nil]  &
:: r:Fresh ::
[ nil |
   -(BNAME:Uname ; e(mkey(BNAME:Uname, s), ANAME:Uname ;
    NAS:Nonce ; NBS:Nonce)),
   +(e(mkey(ANAME:Uname, s), BNAME:Uname ;
   seskey(ANAME:Uname, BNAME:Uname, n(s, r:Fresh)) ;
    NAS:Nonce ; NBS:Nonce) ; e(mkey(BNAME:Uname, s),
     ANAME:Uname ; seskey(ANAME:Uname, BNAME:Uname,
     n(s, r:Fresh)))), nil] [nonexec].
var LIST : SMsgList-R . var K : IntruderKnowledge .
var S : StrandSet .
```

4.10.3. Intruder

```
 vars C D : Name .
 var r : Fresh .
 var K : Key .
 vars M N : Msg .
        => D, n(i, r), mkey(i, C), mkey(C, i), mkey(i, s) .
K, M   => d(K, M), e(K, M) .
M ; N <=> M, N .
```

Transformación a Maude NPA, una de las ventajas de PSL es su syntax sobre DOLEVYAO que usa Maude NPA.

```
eq STRANDS-DOLEVYAO =
:: nil ::
[ nil |
   +(D:Name), nil]  &
:: nil ::
[ nil |
   +(mkey(i, s)), nil]  &
:: nil ::
[ nil |
   +(mkey(i, C:Name)), nil]  &
:: nil ::
[ nil |
   +(mkey(C:Name, i)), nil]  &
:: nil ::
[ nil |
   -(M:Msg),
   -(N:Msg),
   +(M:Msg ; N:Msg), nil]  &
:: nil ::
```

```
[ nil |
   -(K:Key),
   -(M:Msg),
   +(e(K:Key, M:Msg)), nil]  &
:: nil ::
[ nil |
   -(K:Key),
   -(M:Msg),
   +(d(K:Key, M:Msg)), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; N:Msg),
   +(N:Msg), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; N:Msg),
   +(M:Msg), nil]  &
:: r:Fresh ::
[ nil |
   +(n(i, r:Fresh)), nil] [nonexec].
```

4.10.4. Attacks

Ataque 0 PSL.

```
0 .
      B executes protocol .
      Subst(B) = ANAME |-> a , BNAME |-> b, SNAME |-> s .
```

Ataque 0 Maude NPA.

```
eq ATTACK-STATE(0)=
:: r:Fresh ::
[ nil,
   -(a ; NA:Nonce),
   +(b ; e(mkey(b, s), a ; NA:Nonce ; n(b, r:Fresh))),
   -(e(mkey(b, s), a ; SKB:Sessionkey) ;
   e(SKB:Sessionkey, n(b, r:Fresh))) | nil]
|| empty
||
nil
||
nil
||
nil[nonexec].
```

4.10.5. Resultados

Ejecución (Yahalom) del Maude NPA generado.

```
reduce in MAUDE-NPA : summary(1) .
rewrites: 5514310 in 4956ms cpu (4955ms real) (1112583 rewrites/second)
result Summary: States>> 2 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(2) .
rewrites: 7637308 in 8296ms cpu (8295ms real) (920543 rewrites/second)
result Summary: States>> 8 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(3) .
rewrites: 20945956 in 23101ms cpu (23099ms real) (906694 rewrites/second)
result Summary: States>> 19 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(4) .
rewrites: 54142598 in 56571ms cpu (56568ms real) (957064 rewrites/second)
result Summary: States>> 31 Solutions>> 1
```

Ejecución (Yahalom) del Maude NPA original.

```
reduce in MAUDE-NPA : summary(1) .
rewrites: 5514310 in 4948ms cpu (4950ms real) (1114382 rewrites/second)
result Summary: States>> 2 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(2) .
rewrites: 7637308 in 8312ms cpu (8312ms real) (918771 rewrites/second)
result Summary: States>> 8 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(3) .
rewrites: 20945956 in 23093ms cpu (23092ms real) (907008 rewrites/second)
result Summary: States>> 19 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(4) .
rewrites: 54142598 in 56879ms cpu (56874ms real) (951881 rewrites/second)
result Summary: States>> 31 Solutions>> 1
```

4.11. Woo and Lam

Notación Alice-Bob para la especificación.

A -> B : A

B -> A : Nb

A -> B : E(Kas:Nb)

B -> S : E(Kbs:A,E(Kas:Nb))

S -> B : E(Kbs:Nb)

4.11.1. Theory

Types y subtypes.

```
types  UName SName Name Key Nonce  Masterkey Sessionkey .
subtype Masterkey  Sessionkey < Key .
subtype SName UName < Name .
subtype Name < Public .
```

Ilustración 15 Ilustraciones protocolo Woo and Lam

Transformación a Maude NPA

```
sorts UName  SName  Name  Key  Nonce  Masterkey  Sessionkey .
subsorts Masterkey  Sessionkey < Key .
subsorts SName  UName < Name .
subsorts Name < Public .
```

Operadores PSL

```
op n : Name Fresh -> Nonce .
ops a b i : -> UName [ctor] .
op s : ->  SName  [ctor] .
op mkey : Name Name -> Masterkey .
op seskey : Name Name Nonce -> Sessionkey .
op e  : Key Msg -> Msg .
op d : Key Msg -> Msg  .
op _;_ : Msg Msg -> Msg [gather (e E)] .
```

Operadores en Maude NPA

```
op n : Name  Fresh -> Nonce [ frozen ] .
ops a  b  i : -> UName [ ctor ] .
op s : -> SName [ ctor ] .
op mkey : Name  Name -> Masterkey [ frozen ] .
op seskey : Name  Name  Nonce -> Sessionkey [ frozen ] .
op e : Key  Msg -> Msg [ frozen ] .
op d : Key  Msg -> Msg [ frozen ] .
op _;_ : Msg  Msg -> Msg [ gather  (  e  E  )  frozen ] .
```

Variables y ecuaciones

```
var K : Key .
var Z : Msg .
eq d(K, e(K, Z)) = Z .
eq e(K, d(K, Z)) = Z .
```

Ecuaciones en Maude NPA

```
eq d ( K:Key , e ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
eq e ( K:Key , d ( K:Key , Z:Msg ) ) = Z:Msg [ variant ] .
```

4.11.2. Protocol

Creacion de variables, roles, input y Definiciones.

```
vars ANAME BNAME : UName .
var SNAME : SName .
vars NB NBS : Nonce .
vars MA M N : Msg .
var r : Fresh .
var K : Key .
var D : Name .

roles A B S .

In(A) = ANAME, BNAME, SNAME .
In(B) = ANAME, BNAME, SNAME .
In(S) = ANAME, BNAME, SNAME .

Def(A) = kas := mkey(ANAME, s) .
Def(B) = nb  := n(BNAME,r),    kbs := mkey(BNAME, s) .
Def(S) = kas := mkey(ANAME,s), kbs := mkey(BNAME,s) .
```

Protocolo en especificación Alice-Bob

```
1 . A -> B : ANAME |- ANAME   .

2 . B -> A : nb |- NB .

3 . A -> B : e(kas, NB) |- MA .

4 . B -> S :  e(kbs, ANAME ; MA)
              |- e(kbs, ANAME ; e(kas, NBS)) .

5 . S -> B :  e(kbs, NBS)
              |- e(kbs, nb) .

Out(A) = NB .
Out(B) = nb .
Out(S) = NB .
```

Resultado de la transformación del protocolo

```
:: nil ::
[ nil |
   +(ANAME:UName),
   -(NB:Nonce),
   +(e(mkey(ANAME:UName, s), NB:Nonce)), nil]  &
:: nil ::
[ nil |
   -(e(mkey(BNAME:UName, s), ANAME:UName ;
   e(mkey(ANAME:UName, s), NBS:Nonce))),
   +(e(mkey(BNAME:UName, s), NBS:Nonce)), nil]  &
:: r:Fresh ::
[ nil |
   -(ANAME:UName),
   +(n(BNAME:UName, r:Fresh)),
   -(MA:Msg),
   +(e(mkey(BNAME:UName, s), ANAME:UName ; MA:Msg)),
   -(e(mkey(BNAME:UName, s), n(BNAME:UName, r:Fresh))),
    nil] [nonexec].
var LIST : SMsgList-R . var K : IntruderKnowledge .
var S : StrandSet .
```

4.11.3. Intruder

Declaración de variables y especificación donde el intruso tiene el conocimiento de las mismas.

```
var D    : Name .
var r    : Fresh .
var K    : Key .
vars M N : Msg .
       => D, n(i, r), mkey(D, i), mkey(i, D) .
K, M  => d(K, M), e(K, M) .
M, N <=> M ; N .
```

Transformación a Maude NPA, una de las ventajas de PSL es su syntax sobre DOLEVYAO que usa Maude NPA.

```
:: nil ::
[ nil |
   +(D:Name), nil]  &
:: nil ::
[ nil |
   +(mkey(i, D:Name)), nil]  &
:: nil ::
[ nil |
   +(mkey(D:Name, i)), nil]  &
:: nil ::
[ nil |
   -(M:Msg),
   -(N:Msg),
   +(M:Msg ; N:Msg), nil]  &
:: nil ::
[ nil |
   -(K:Key),
   -(M:Msg),
   +(e(K:Key, M:Msg)), nil]  &
:: nil ::
```

```
:: nil ::
[ nil |
   -(K:Key),
   -(M:Msg),
   +(d(K:Key, M:Msg)), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; N:Msg),
   +(N:Msg), nil]  &
:: nil ::
[ nil |
   -(M:Msg ; N:Msg),
   +(M:Msg), nil]  &
:: r:Fresh ::
[ nil |
   +(n(i, r:Fresh)), nil] [nonexec].
```

4.11.4. Attacks

Ataque 0 de PSL

```
0 .
   B executes protocol .
   Subst(B) = ANAME |-> a , BNAME |-> b, SNAME |-> s .
```

Ataque 0 del Maude NPA generado

```
eq ATTACK-STATE(0)=
:: r:Fresh ::
[ nil,
   -(a),
   +(n(b, r:Fresh)),
   -(MA:Msg),
   +(e(mkey(b, s), a ; MA:Msg)),
   -(e(mkey(b, s), n(b, r:Fresh))) | nil]
|| empty
||
nil
||
nil
||
nil[nonexec].
```

4.11.5. Resultados

Ejecución (Woo and Lam) del Maude NPA generado por PSL.

```
reduce in MAUDE-NPA : summary(6) .
rewrites: 6691003 in 11140ms cpu (11222ms real) (600591 rewrites/second)
result Summary: States>> 7 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(7) .
rewrites: 7312552 in 10180ms cpu (10183ms real) (718280 rewrites/second)
result Summary: States>> 5 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(8) .
rewrites: 3078403 in 3880ms cpu (3880ms real) (793353 rewrites/second)
result Summary: States>> 2 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(9) .
rewrites: 474194 in 540ms cpu (538ms real) (878081 rewrites/second)
result Summary: States>> 1 Solutions>> 1
==========================================
```

Ejecución (Woo and Lam) del Maude NPA original.

```
reduce in MAUDE-NPA : summary(6) .
rewrites: 7194773 in 6028ms cpu (6028ms real) (1193484 rewrites/second)
result Summary: States>> 7 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(7) .
rewrites: 7817884 in 6112ms cpu (6111ms real) (1279024 rewrites/second)
result Summary: States>> 5 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(8) .
rewrites: 3314965 in 2400ms cpu (2401ms real) (1381149 rewrites/second)
result Summary: States>> 2 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(9) .
rewrites: 508990 in 288ms cpu (285ms real) (1767215 rewrites/second)
result Summary: States>> 1 Solutions>> 1
==========================================
```

5. Protocolos con homomorfismo

Las propiedades algebraicas que son usadas en estos protocolos son las homomórficas sobre el cifrado. La propiedad a seguir es {[x, y]}z= [{x}z, {y}z].

5.1. Needham Schroeder Lowe ECB

Especificación del protocolo en notación Alice-Bob.

A -> B: Kb:(A,Na)

B -> A: Ka:(B,Nb,Na)

A -> B: Kb:Nb

5.1.1. Theory

Types y subtypes.

```
types   SName Name Key Nonce Masterkey Sessionkey .
subtypes Masterkey  Sessionkey < Key .
subtype SName < Name .
subtype Name < Public .
```

Ilustración 16 Ilustraciones protocolo Needham Schroeder Lowe ECB

Transformación a Maude NPA.

```
sorts Name  Key  Nonce .
subsorts Name < Key .
subsorts Name  Nonce  Key < Msg .
subsorts Name < Public .
```

Operadores PSL.

```
op pk : Msg Key -> Msg .
op n : Name Fresh -> Nonce .
op a : -> Name . // Alice
op b : -> Name . // Bob
op i : -> Name . // Intruder
op _;_ : Msg Msg -> Msg [gather (e E)] .
```

Operadores en Maude NPA.

```
op pk : Msg  Key -> Msg [ frozen ] .
op n : Name  Fresh -> Nonce [ frozen ] .
op a : -> Name .
op b : -> Name .
op i : -> Name .
op _;_ : Msg  Msg -> Msg [ gather  (  e  E  )  frozen ] .
```

Ecuaciones en PSL con homomorfismo.

```
eq pk(X:Msg ; Y:Msg, K:Key) = pk(X:Msg, K:Key) ; pk(Y:Msg, K:Key)
[nonexec label homomorphism metadata "builtin-unify"] .
```

Ecuaciones en Maude NPA con homomorfismo.

```
eq pk ( X:Msg ; Y:Msg , K:Key ) = pk ( X:Msg , K:Key ) ; pk ( Y:Msg , K:Key )
[ nonexec label homomorphism metadata "builtin-unify" ] .
```

5.1.2. Protocol

Creacion de variables, roles input y Definiciones.

```
vars ANAME BNAME A1NAME B1NAME : Name .
var r r1 : Fresh .
vars M1 M2 : Msg .
vars KA KB : Key .
vars NA NB NB1 : Nonce .

roles A B .

  In(A) = ANAME, BNAME .
  In(B) = BNAME, ANAME .
```

Protocolo en especificación Alice-Bob

```
1 . A -> B : pk(ANAME ; n(ANAME, r), BNAME)
             |- pk(ANAME ; NA, BNAME) .

2 . B -> A : pk(NA ; n(BNAME, r1) ; BNAME, ANAME)
             |- pk(n(ANAME, r) ; NB ; BNAME, ANAME) .

3 . A -> B : pk(NB, BNAME)
             |- pk(n(BNAME, r1), BNAME) .

Out(A) = n(ANAME, r) .
```

Transformación del protocolo a Maude NPA, la lectura es difícil y las variables necesarias las declara en el momento con la declaración

de tipos que ofrece Maude (variable: tipo), también dificulta la lectura el hecho de que el código haya sido generado automáticamente.

```
eq STRANDS-PROTOCOL =
:: r:Fresh ::
[ nil |
   +(pk(ANAME:Name ; n(ANAME:Name, r:Fresh), BNAME:Name)),
   -(pk(n(ANAME:Name, r:Fresh) ; NB:Nonce ; BNAME:Name, ANAME:Name)),
   +(pk(NB:Nonce, BNAME:Name)), nil]  &
:: r1:Fresh ::
[ nil |
   -(pk(ANAME:Name ; NA:Nonce, BNAME:Name)),
   +(pk(NA:Nonce ; n(BNAME:Name, r1:Fresh) ; BNAME:Name, ANAME:Name)),
   -(pk(n(BNAME:Name, r1:Fresh), BNAME:Name)), nil] [nonexec].
var LIST : SMsgList-R . var K : IntruderKnowledge . var S : StrandSet .
```

5.1.3. Intruder

Declaración de variables y especificación, donde el intruso tiene el conocimiento de las mismas.

```
var C   : Name .
var r2   : Fresh .
vars X Y : Msg .
var K    : Key .

       => C .
X, Y <=> X ; Y .
X => pk(X, K) .
pk(X, i) => X .
```

Transformación a Maude NPA, una de las ventajas de PSL es su syntax sobre DOLEVYAO que usa Maude NPA.

```
eq STRANDS-DOLEVYAO =
:: nil ::
[ nil |
   +(C:Name), nil]  &
:: nil ::
[ nil |
   -(X:Msg),
   +(pk(X:Msg, K:Key)), nil]  &
:: nil ::
[ nil |
   -(Y:Msg),
   -(X:Msg),
   +(X:Msg ; Y:Msg), nil]  &
:: nil ::
```

```
[ nil |
   -(pk(X:Msg, i)),
   +(X:Msg), nil]  &
:: nil ::
[ nil |
   -(X:Msg ; Y:Msg),
   +(X:Msg), nil]  &
:: nil ::
[ nil |
   -(X:Msg ; Y:Msg),
   +(Y:Msg), nil] [nonexec].
```

5.1.4. Attack

Ataque 0, ejección regular del protocolo

```
0 .
    B executes protocol .
    Subst(B) = BNAME |-> b, ANAME |-> a .
    Intruder learns n(BNAME, r1) .
```

Ataque 0 del Maude NPA generado

```
 eq ATTACK-STATE(0)=
:: r1:Fresh ::
[ nil,
   -(pk(a ; NA:Nonce, b)),
   +(pk(NA:Nonce ; n(b, r1:Fresh) ; b, a)),
   -(pk(n(b, r1:Fresh), b)) | nil]
||
n(b, r1:Fresh) inI
||
nil
||
nil
||
nil[nonexec].
```

5.1.5. Resultados

Ejecución (Needham-Schroeder-Lowe Modified Protocol with ECB) del Maude NPA generado por PSL.

```
reduce in MAUDE-NPA : summary(9) .
rewrites: 110935623 in 124003ms cpu (123994ms real) (894615 rewrites/second)
result Summary: States>> 23 Solutions>> 3
==========================================
reduce in MAUDE-NPA : summary(10) .
rewrites: 72768097 in 62847ms cpu (62843ms real) (1157844 rewrites/second)
result Summary: States>> 19 Solutions>> 4
==========================================
reduce in MAUDE-NPA : summary(11) .
rewrites: 73871559 in 54183ms cpu (54179ms real) (1363361 rewrites/second)
result Summary: States>> 25 Solutions>> 5
==========================================
reduce in MAUDE-NPA : summary(12) .
rewrites: 144518034 in 104322ms cpu (104315ms real) (1385300 rewrites/second)
result Summary: States>> 26 Solutions>> 5
==========================================
reduce in MAUDE-NPA : summary(13) .
rewrites: 195064584 in 160658ms cpu (160649ms real) (1214160 rewrites/second)
result Summary: States>> 26 Solutions>> 6
==========================================
```

Ejecución (Needham-Schroeder-Lowe Modified Protocol with ECB) del Maude NPA original.

```
reduce in MAUDE-NPA : summary(9) .
rewrites: 109235788 in 109542ms cpu (109536ms real) (997196 rewrites/second)
result Summary: States>> 23 Solutions>> 3
==========================================
reduce in MAUDE-NPA : summary(10) .
rewrites: 70943799 in 52371ms cpu (52368ms real) (1354631 rewrites/second)
result Summary: States>> 19 Solutions>> 4
==========================================
reduce in MAUDE-NPA : summary(11) .
rewrites: 72827601 in 46810ms cpu (46804ms real) (1555782 rewrites/second)
result Summary: States>> 25 Solutions>> 5
==========================================
reduce in MAUDE-NPA : summary(12) .
rewrites: 143045797 in 91053ms cpu (91048ms real) (1571004 rewrites/second)
result Summary: States>> 26 Solutions>> 5
==========================================
reduce in MAUDE-NPA : summary(13) .
rewrites: 196857704 in 143236ms cpu (143229ms real) (1374349 rewrites/second)
result Summary: States>> 26 Solutions>> 6
==========================================
```

5.2. Homomorfismo HCP

Protocolo desarrollado por Catherine Meadows, en la que Alice y Bob tienen unos mensaje de datos DA y DB. Tenemos una función f(DA,DB) en la que ellos quieren que sea computada, pero…

1. Ellos mismos no pueden computarla.
2. Y aunque ellos pudieran, ellos no quieren compartir sus datos eluno con el otro.

Para ello tenemos el Server que puede computar. Pero Alice y Bob no quieren que el server vea DA y DB. Asumimos que el server es "honesto pero curioso"; con esto, ellos aceptan seguir las reglas del protocolo para tener así los datos computados, por otro lado el Server intentará averiguar durante el proceso todo lo que pueda acerca de DA y DB.

Notación Alice-Bob

A -> B: sign(B ; NA ; pke(hpke (DA, k(A,B)), s),A)

B -> A: sign(NA ; NB ; pke(hpke (DB, k(A,B)), s),B)

A -> S: sign(A;B;NA;NB;pke(hpke(DA,k(A,B)),s);pke(hpke(DB,k(A,B)), s),A)

S-> A,B : sign(A ; B ; NA ; NB ; f(hpke(DA,k(A,B)) , hpke(DB,k(A,B))), s)

5.2.1. Theory

Types y subtypes en PSL

```
types Name Nonce Pkey Data Enc Sign HEnc .
subtypes Nonce Pkey Data Name Enc Sign HEnc < Msg  .
subtype Name < Public .
```

Ilustración 17 ilustraciones Homomorfismo HCP

Transformación a Maude NPA.

```
sorts Name  Nonce  Pkey  Data  Enc  Sign  HEnc .
subsorts Nonce  Pkey  Data  Name  Enc  Sign  HEnc < Msg .
subsorts Name < Public .
```

Operadores PSL.

```
op n : Name Fresh -> Nonce .
op data : Name Fresh -> Data .
op a : -> Name . // Alice
op b : -> Name . // Bob
op s : -> Name . // Server
op i : -> Name . // Intruder
op _;_ : Msg Msg -> Msg [gather (e E)] .
op pkey : Name Name -> Pkey .
op f : Msg Msg -> HEnc .
op pke : Msg Name -> Enc .
op hpke : Msg Pkey -> HEnc .
op sign : Msg Name -> Sign .
```

Tras la transformación a Maude NPA de los operadores.

```
op n : Name  Fresh -> Nonce [ frozen ] .
op data : Name  Fresh -> Data [ frozen ] .
op a : -> Name .
op b : -> Name .
op s : -> Name .
op i : -> Name .
op _;_ : Msg  Msg -> Msg [ gather  (  e  E  )  frozen ] .
op pkey : Name  Name -> Pkey [ frozen ] .
op f : Msg  Msg -> HEnc [ frozen ] .
op pke : Msg  Name -> Enc [ frozen ] .
op hpke : Msg  Pkey -> HEnc [ frozen ] .
op sign : Msg  Name -> Sign [ frozen ] .
```

Ecuación en PSL con homomorfismo.

```
eq hpke(f(X:Msg,Y:Msg) , K:Pkey) = f(hpke(X:Msg , K:Pkey) , hpke(Y:Msg , K:Pkey))
[nonexec label homomorphism metadata "builtin-unify"] .
```

Resultado de la ecuación homomórfica en Maude NPA.

```
eq hpke ( f ( X:Msg , Y:Msg ) , K:Pkey ) = f ( hpke ( X:Msg , K:Pkey ) , hpke ( Y:Msg , K:Pkey )
[ nonexec label homomorphism metadata "builtin-unify" ] .
```

5.2.2. Protocol

Declaración de las variables, los roles Alice, Bob, Server y los Input correspondientes a cada rol.

```
vars X1 X2 X3 X4 : HEnc .
vars Y1 Y2 : Enc .
vars Z1 Z2 : Sign .
vars X' Y' Z' V' W' : Msg .
vars r r' r'' r1 r2 r3 r4 : Fresh .
vars N N1 N2 : Nonce .
vars V P A B S : Name .

roles A B S .

   In(A) = A, B, N, S .
   In(B) = B, A .
   In(S) = A, B, N1 .
```

Notación Alice-Bob del protocolo.

```
1 . A -> B : sign(B ; n(A, r) ; pke(hpke(data(A,r'), pkey(A,B)),s),A)
             |- sign( B ; N1  ; Y2 , A) .

2 . B -> A : sign( N1 ; n(B,r1) ; pke(hpke(data(B,r2),pkey(A,B)),s), B)
             |- sign(n(A,r) ; N ; Y1 , B) .

3 . A -> S : sign( A ; B ; n(A,r) ; N ; pke(hpke(data(A,r'),pkey(A,B)),s) ;
             Y1  , A)
             |- sign( A ; B ; N1 ; N2 ; pke(X3 , s) ; pke(X4 , s) , A) .

4 . S -> A : sign( A ; B ; N1 ; N2 ; f(X3,X4) , s)
             |- sign( A ; B ; n(A,r) ; N ; X1 , s ) .

             //|- sign( A ; B ; N1 ; n(B,r1) ; X2, s ) .

5 . S -> B : sign( A ; B ; N1 ; N2 ; f(X3,X4) , s)
              |- sign( A ; B ; N1 ; n(B,r1) ; X2, s ) .

Out(A) = A, B, s .
Out(B) = B, A .
Out(S) = A, B .
```

Transformación del protocolo a Maude NPA, la lectura es difícil y las variables necesarias las declara en el momento con la declaración de tipos que ofrece Maude (variable: tipo), también dificulta la lectura el hecho de que el código haya sido generado automáticamente.

```
eq STRANDS-PROTOCOL =
:: nil ::
[ nil |
   -(sign(A:Name ; B:Name ; N1:Nonce ; N2:Nonce ;
    pke(X3:HEnc, s) ; pke(X4:HEnc, s), A:Name)),
   +(sign(A:Name ; B:Name ; N1:Nonce ; N2:Nonce ;
   f(X3:HEnc, X4:HEnc), s)),
   +(sign(A:Name ; B:Name ; N1:Nonce ; N2:Nonce ;
   f(X3:HEnc, X4:HEnc), s)), nil]  &
:: r:Fresh,r':Fresh ::
[ nil |
   +(sign(B:Name ; n(A:Name, r:Fresh) ;
   pke(hpke(data(A:Name, r':Fresh), pkey(A:Name, B:Name)), s), A:Name)),
   -(sign(n(A:Name, r:Fresh) ; N:Nonce ; Y1:Enc, B:Name)),
   +(sign(A:Name ; B:Name ; n(A:Name, r:Fresh) ; N:Nonce ;
    pke(hpke(data(A:Name, r':Fresh), pkey(A:Name, B:Name)),
     s) ; Y1:Enc, A:Name)),
   -(sign(A:Name ; B:Name ; n(A:Name, r:Fresh) ; N:Nonce ;
    X1:HEnc, s)), nil]  &
:: r1:Fresh,r2:Fresh ::
[ nil |
   -(sign(B:Name ; N1:Nonce ; Y2:Enc, A:Name)),
   +(sign(N1:Nonce ; n(B:Name, r1:Fresh) ;
   pke(hpke(data(B:Name, r2:Fresh), pkey(A:Name, B:Name)), s), B:Name)),
   -(sign(A:Name ; B:Name ; N1:Nonce ; n(B:Name, r1:Fresh) ; X2:HEnc, s)),
    nil] [nonexec].
var LIST : SMsgList-R . var K : IntruderKnowledge . var S : StrandSet .
```

5.2.3. Intruder

Declaración de variables y especificación, donde el intruso tiene el conocimiento de las mismas.

```
vars CA CB : Name .
vars X Y Z W  : Msg .

X, Y               <=> X ; Y .
X, CA               => pke(X, CA) .
pke(X,i)            => X .
X                  <=> sign(X,i) .
X, CA, CB           => hpke(X,pkey(CA,CB)) .
hpke(X,pkey(CA,i)) => X .
hpke(X,pkey(i,CA)) => X .
                    => CA .
```

Transformación a Maude NPA, una de las ventajas de PSL es su syntax sobre DOLEVYAO que usa Maude NPA.

```
eq STRANDS-DOLEVYAO =
:: nil ::
[ nil |
   +(CA:Name), nil]  &
:: nil ::
[ nil |
   -(X:Msg),
   +(sign(X:Msg, i)), nil]  &
:: nil ::
[ nil |
   -(Y:Msg),
   -(X:Msg),
   +(X:Msg ; Y:Msg), nil]  &
:: nil ::
[ nil |
   -(CA:Name),
   -(X:Msg),
   +(pke(X:Msg, CA:Name)), nil]  &
:: nil ::
[ nil |
   -(CB:Name),
   -(CA:Name),
   -(X:Msg),
   +(hpke(X:Msg, pkey(CA:Name, CB:Name))), nil]  &
:: nil ::
[ nil |
   -(X:Msg ; Y:Msg),
   +(X:Msg), nil]  &
:: nil ::
[ nil |
   -(X:Msg ; Y:Msg),
   +(Y:Msg), nil]  &
:: nil ::
[ nil |
   -(pke(X:Msg, i)),
   +(X:Msg), nil]  &
:: nil ::
[ nil |
   -(hpke(X:Msg, pkey(i, CA:Name))),
   +(X:Msg), nil]  &
:: nil ::
[ nil |
   -(hpke(X:Msg, pkey(CA:Name, i))),
   +(X:Msg), nil]  &
:: nil ::
[ nil |
   -(sign(X:Msg, i)),
   +(X:Msg), nil] [nonexec].
```

5.2.4. Attacks

Ataque en PSL.

```
0 .
  A executes protocol .
  Subst(A) = A |-> a, B |-> b, S |-> s .
  without:
    B executes protocol .
    Subst(B) = A |-> a, B |-> b, Y2 |-> pke(hpke(data(A,r'),
    pkey(A,B)),s), N1 |-> n(a,r) .
```

Traducción del ataque tras la transformación a Maude NPA.

```
eq ATTACK-STATE(0)=
:: r:Fresh,r':Fresh ::
[ nil,
   +(sign(b ; n(a, r:Fresh) ; pke(hpke(data(a, r':Fresh),
    pkey(a, b)), s), a)),
   -(sign(n(a, r:Fresh) ; N:Nonce ; Y1:Enc, b)),
   +(sign(a ; b ; n(a, r:Fresh) ; N:Nonce ; pke(hpke(data(a, r':Fresh),
    pkey(a, b)), s) ; Y1:Enc, a)),
   -(sign(a ; b ; n(a, r:Fresh) ; N:Nonce ; X1:HEnc, s)) | nil]
|| empty
||
nil
||
nil
|| never((S &
:: r1:Fresh,r2:Fresh ::
[ nil,
   -(sign(b ; n(a, r:Fresh) ; pke(hpke(data(a, r':Fresh), pkey(a, b)),
   s), a)),
   +(sign(n(a, r:Fresh) ; n(b, r1:Fresh) ; pke(hpke(data(b, r2:Fresh),
   pkey(a, b)), s), b)),
   -(sign(a ; b ; n(a, r:Fresh) ; n(b, r1:Fresh) ; X2:HEnc, s)) | nil] )
   || K)[nonexec].
```

5.2.5. Resultados

Ejecución (Homomorphism HPC) del Maude NPA generado por PSL.

```
reduce in MAUDE-NPA : summary(9) .
rewrites: 12170330 in 89761ms cpu (89755ms real) (135585 rewrites/second)
result Summary: States>> 3 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(10) .
rewrites: 18752096 in 154305ms cpu (154297ms real) (121525 rewrites/second)
result Summary: States>> 2 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(11) .
rewrites: 13724403 in 113407ms cpu (113399ms real) (121018 rewrites/second)
result Summary: States>> 1 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(12) .
rewrites: 8416331 in 73812ms cpu (73809ms real) (114022 rewrites/second)
result Summary: States>> 1 Solutions>> 0
==========================================
```

Ejecución (Homomorphism HPC) del Maude NPA original.

```
reduce in MAUDE-NPA : summary(9) .
rewrites: 11778794 in 95893ms cpu (95889ms real) (122831 rewrites/second)
result Summary: States>> 2 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(10) .
rewrites: 13283493 in 109122ms cpu (109114ms real) (121729 rewrites/second)
result Summary: States>> 3 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(11) .
rewrites: 16258577 in 123823ms cpu (123818ms real) (131304 rewrites/second)
result Summary: States>> 2 Solutions>> 1
==========================================
reduce in MAUDE-NPA : summary(12) .
rewrites: 10750638 in 85025ms cpu (85016ms real) (126440 rewrites/second)
result Summary: States>> 1 Solutions>> 1
==========================================
```

Observar que en este protocolo no se consigue lo deseado, que es obtener el mismo resultado que un protocolo original para Maude NPA. Esto es debido a que PSL tiene sus limitaciones con la notación Alice-Bob, ya que no acepta el envío de un mensaje por un rol y la recepción de dicho mensaje por diferentes roles.

6. Futuro trabajo y conclusiones

6.1. Conclusiones

La especificación de protocolos criptográficos es compleja, por eso pienso que hay que facilitar el uso de notaciones como Alice-Bob, ya que estas notaciones son intuitivas, claras y expresivas. Maude-PSL es una herramienta que usa una especificación experimental que contiene mucho potencial y que si se desarrolla adecuadamente puede llegar al uso que actualmente tiene Maude-NPA.

Una de las conclusiones a las que he llegado en el desarrollo de mi trabajo final de grado, es en el proceso de documentación/aprendizaje sobre ciertos protocolos criptográficos, estos normalmente se encuentra en notación Alice and Bob, y a la hora de especificar los protocolos, si trabajas con una herramienta como Maude-PSL que utiliza la misma notación facilita el desarrollando de la especificación y no lo retrasa como en el caso de Maude-NPA

en la que le tienes que dedicar tiempo al estudio de la notación de strands.

Maude PSL abre una puerta a la compresión y el desarrollo de protocolos criptográficos. Con el tiempo necesario para su evolución y mejora puede llegar al nivel de Maude-NPA.

6.2. Futuro trabajo

Como hemos podido observar, Maude-PSL contempla parte de las características de Maude-NPA.

Maude-PSL expresa la mayoría de protocolos criptográficos que Maude-NPA es capaz de soportar. Mejorando el desarrollo de la transformación, gracias a Python, PSL podrá dar soporte a todo tipo de casos: como el de envío de un mensaje por un rol y recibido por diferentes roles (principal problema observado en el protocolo homomórfico HPC).

Otra de las tareas interesantes se centraría en modelar y observar protocolos con or-exclusivo para ver la capacidad de soporte que puede llegar a tener Maude-PSL.

7. Referencias

[1] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Mart ´ı-Oliet, J. Meseguer, and C. Talcott. Maude Manual (Version 2.6), 2011. Última consulta 11 marzo 2016.

[2] S. Escobar, C. Meadows, and J. Meseguer. Maude-NPA, (Version 2.0), November 26th, 2011. Última consulta 02 septiembre 2016.

[3] A. Russel Cholewa. Maude-PSL: A new input language for Maude-NPA, Thesis of master of Science in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2015. Última consulta 02 septiembre 2016.

[4] Sitio Web Maude NPA (Universitat Politecnica de Valencia, realizada por Antonio González Burgueño), Última consultad 03 agosto 2016. Disponible:http://users.dsic.upv.es/~sescobar/Maude-NPA_Protocols/index.html

[5] Sitio Web de almacenamiento archivos (Github, datos del usuario A. Cholewa).Actualizada Mayo 15, 2015, Última consulta 14 febrero 2016. Disponible :https://github.com/archolewa/Maude-PSL

[6] C. Meadows, and J. Meseguer. Maude-NPA: Cryptographic Protocol Analysis Modulo Equationals Properties, 2006. Última consulta 12 julio 2016