UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Verificación automática del protocolo TLS 1.3 usando Maude-NPA.

Trabajo Fin de Máster

Máster Universitario en
Ingenieria y Tecnología de Sistemas Software

Departamento de Sistemas Informáticos y Computación

Autor: Jose Lluch Palop

Tutor: Dr. Santiago Escobar Román

Valencia, septiembre de 2019

Curso 2018-2019

Resumen

La importancia de la confidencialidad y seguridad en las tecnologías de la información es un tema en el que poco a poco más entidades tanto públicas como privadas aúnan esfuerzos con el fin de conseguir entornos más seguros y robustos frente a los atacantes. El protocolo TLS tiene un papel fundamental en el establecimiento de conexiones seguras sobre medios no seguros como es Internet. Es ampliamente utilizado hoy en día en todos los navegadores y en informática en general por lo que la razón de este trabajo se fundamenta en el estudio de las propiedades de seguridad de este protocolo criptográfico mediante el uso de Maude-NPA, herramienta de verificación de protocolos criptográficos de comunicación utilizada para dicho análisis. A lo largo de este trabajo se profundizará en la herramienta Maude-NPA, el protocolo TLS, el estado del arte en el análisis de este protocolo con herramienta similares como Tamarin y una comparativa de resultados obtenidos con Maude-NPA frente a los ya extraídos con Tamarin.

Índice de Contenido

1. Introducción 9
2. Maude-NPA 10
2.1 Modelado de protocolos en Maude-NPA 10
2.1.1 Sintaxis del protocolo: Tipos, subtipos y operadores. 11
2.1.2 Propiedades criptográficas 13
2.1.3 Especificación de Strands 14
2.1.3.1 Strands de protocolos 14
2.1.3.2 Hablidades del intruso 16
2.1.3.3 Análisis del protocolo: Búsqueda de ataques. 17
3. Protocolo TLS 19
3.1 Objetivos y propiedades de seguridad 19
3.2 Versión 1.3 20
3.3 Diseño de los nuevos *handshakes* 22
3.3.1 *Handshake* inicial 22
3.3.2 0-RTT 23
3.3.3 Reanudación de sesión y PSK. 24
4. Análisis del protocolo TLS 1.3 en Tamarin 26
4.1 Modelado de protocolos en Tamarin. 27
4.2 Modelado del protocolo TLS 1.3 en Tamarin. 30
4.2.1 Diseño y codificación del modelo en Tamarin. 30
4.2.2 Análisis formal del protocolo 34
4.3 Conclusiones del análisis 37
5. Simplificación del modelo de TLS 1.3 en Maude-NPA 40
5.1 Símbolos del protocolo y propiedades algebraicas 40
5.2 Especificación de los *Strands*. 42
5.2.1 Habilidades del intruso. Reglas Dolev-Yao 42
5.2.2 Strands del protocolo 43
5.2.3 Gramáticas 44
5.3 Análisis del protocolo 45
5.3.0 Ejecución regular 46
5.3.1 Autenticación del Servidor 48
5.3.2 Autenticación del Cliente 51
5.3.3 Secreto del Servidor 54

5.3.4 Secreto del Cliente 56
5.4 Conclusiones del análisis 58
6. Conclusiones y trabajo futuro 59
7. Bibliografía 60

Índice de Imágenes

Ilustración 1. Diagrama de interacción de Alice y Bob. 15
Ilustración 2. Handshake inicial. 23
Ilustración 3. 0-RTT. 24
Ilustración 4. Reanudación de sesión y PSK. 25
Ilustración 5. Reglas codificadas para el Cliente. 32
Ilustración 6. Reglas codificadas para el Servidor. 33
Ilustración 7. Ataque Man-in-the-middle encontrado. 38
Ilustración 8. Expresión regular. Ejecución esperada del modelo. 47
Ilustración 9. Ataque 1 como diagrama de interacción. 50
Ilustración 10. Primera configuración del ataque 2 en un diagrama de interacción. 53
Ilustración 11. Segunda configuración del ataque 2 en un diagrama de interacción. 54

1. Introducción

Hoy día, la seguridad en las comunicaciones a través de internet es uno de los temas que más preocupan a los profesionales del sector. Esta seguridad debe estar presente de inicio a fin en cada una de las transmisiones que se realizan, sin excepción. Si cada vez hay más dispositivos personales conectados, incluso personas que tienen más de un smartphone u ordenador personal, con la paulatina llegada del internet de las cosas (IoT), el impacto que puede llegar a tener una vulnerabilidad fuerte en la seguridad de las comunicaciones puede provocar el descubrimiento desde información personal de particulares hasta datos confidenciales de empresas o entidades gubernamentales.

Es por lo que razón de esta tesis de máster analiza formalmente un protocolo de comunicaciones encargado de establecer conexiones seguras en un entorno, como puede ser Internet, en el que no sabemos quién puede estar interceptando todo lo que enviemos. El objetivo de este trabajo es modelar mediante la herramienta Maude-NPA el protocolo TLS versión 1.3 y así poder estudiar sus propiedades de seguridad. De esta manera verificar la seguridad de un protocolo presente en millones de dispositivos en el mundo.

Este trabajo se divide en cinco capítulos. El primero se centra en la herramienta Maude-NPA. El segundo acerca el protocolo TLS y su versión 1.3. El tercer capítulo presenta el modelado realizado en la herramienta Tamarin, herramienta que como Maude-NPA, forma parte del estado del arte en el modelado formal. El cuarto capítulo expone cómo ha sido el trabajo de modelado en Maude-NPA y el último acerca de las conclusiones que se pueden extraer de todo este trabajo, así como futuros posibles trabajos que surgen a partir de esta tesis de máster.

2. Maude-NPA

En esta sección se presenta la herramienta Maude-NPA (Escobar, Meadows, & Meseguer, 2017), herramienta basada en el lenguaje de programación Maude (Clavel, y otros, 2016), cuya filosofía es la verificación de protocolos criptográficos de comunicación y, en especial, las propiedades. Estas propiedades a verificar se definen como propiedades algebraicas dentro del proceso de modelado del protocolo. Esta herramienta está especializada en demostrar propiedades de seguridad, así como realizar búsquedas de ataques de seguridad. Este trabajo se centra en la segunda de las habilidades de la herramienta mediante la simplificación del protocolo TLS 1.3. Maude-NPA funciona a partir de una configuración final del protocolo buscando una configuración inicial hacia atrás.

2.1 Modelado de protocolos en Maude-NPA

Cualquier protocolo a especificar en esta herramienta se codifica en un solo fichero con extensión maude. Esta especificación parte de una plantilla, la cual consta de tres módulos. En el primero se indica la sintaxis con la que se definirá el protocolo, los tipos, subtipos y operadores. En el segundo, las propiedades criptográficas de los operadores definidos. En último lugar, se definirá el comportamiento del protocolo mediante los denominados Strands.

2.1.1 Sintaxis del protocolo: Tipos, subtipos y operadores.

Maude-NPA importa tres tipos de datos básicos a partir de los cuales se construyen los datos complejos que pueda especificar el usuario:

- **Msg**: Cualquier tipo definido por el usuario debe ser subtipo de Msg, y ningún tipo puede ser supertipo (que Msg sea subtipo) de éste. Por definición, este tipo no puede ser vacío. Es decir, debe de haber al menos un término sin variables.

- **Fresh**: Utilizado para indicar a la herramienta términos cuyo valor debe de ser único a lo largo de toda la ejecución del programa. No puede haber subtipos de este tipo. Generalmente se utiliza como argumento de datos con valores únicos como claves o nonces generados por algunos de los actores del sistema, por ejemplo, n(A, r:Fresh). Al permitir solo variables de tipo Fresh, no es necesario definir símbolos que permitan este tipo de dato.

- **Public**: Utilizado para definir términos públicamente disponibles y conocidos por todos los actores, en especial el intruso. Al igual que el tipo Msg, no pueden haber supertipos de Public y debe de haber al menos, una instancia sin variables. Es decir, no puede estar vacío.

A continuación, se puede observar un ejemplo de tipos y subtipos:

```
--- Sort Information
sorts Name Nonce NeNonceSet Gen Exp Key GenvExp Secret .
subsort Gen Exp < GenvExp .
subsort Name NeNonceSet GenvExp Secret Key < Msg .
```

Una vez determinados los tipos y subtipos, es momento de indicar las posibles operaciones a realizar. Maude-NPA presenta una manera flexible de declarar los operadores del protocolo. Tenemos dos posibilidades, los prefijos, aquellos operadores situados antes del término; y los infijos, los situados entre dos términos.

Además, en la especificación de los operadores se pueden indicar distintas propiedades que pueden satisfacer los símbolos. Estos son:

- asociatividad indicada como *assoc*
- conmutatividad indicada como *comm*
- identidad indicada *id: keyword*
- asociatividad explícita por la derecha o izquierda indicado como *gather (e E)* o *gather (E e)*, respectivamente.

Un claro ejemplo, donde aparece todo lo comentado previamente, sería el siguiente código. En él, existen tanto operadores prefijos como infijos, así como algunos de los algoritmos de unificación.

```
--- Encoding operators for public/private encryption
op pk : Key Msg -> Msg [frozen] .
op sk : Key Msg -> Msg [frozen] .

--- Nonce operator
op n : Name Fresh -> Nonce .

--- NeNonceSet
subsort Nonce < NeNonceSet .
op _*_ : NeNonceSet NeNonceSet -> NeNonceSet [assoc comm] .

--- Concatenation
op _;_ : Msg Msg -> Msg [gather (e E)] .
```

2.1.2 Propiedades criptográficas

Esta herramienta ofrece la posibilidad de especificar propiedades criptográficas propias del protocolo. Estas propiedades se fundamentan en las reglas de la reescritura de términos, en las que dada una parte izquierda de una regla se permite la reescritura automática por la parte derecha de la misma regla y se codifican mediante las reglas ecuaciones propias del lenguaje Maude.

```
*** Encryption/Decryption Cancellation
eq pk(Ke:Key,sk(Ke:Key,Z:Msg)) = Z [variant] .
eq sk(Ke:Key,pk(Ke:Key,Z:Msg)) = Z [variant] .
```

En el ejemplo anterior se aprecia cómo se expresa la propiedad de cancelación, expresada mediante reglas complementarias, presente en la criptografía de clave pública y privada. La primera propiedad se basa en el resultado obtenido de cifrar un mensaje con una clave pública que ha sido previamente descifrado con clave privada, es el propio mensaje en texto plano. La segunda propiedad, complementaria a la primera y más intuitiva de entender, se basa

en el resultado obtenido de descifrar con una clave privada un mensaje previamente cifrado con una clave pública, es el propio mensaje en texto plano.

2.1.3 Especificación de Strands

En este apartado, él último módulo a codificar del fichero comentado previamente, se codifica el comportamiento en sí del protocolo mediante tres partes. Exactamente, se codifica el comportamiento de los participantes en el protocolo, las habilidades del atacante y, por último, los patrones de ataques.

2.1.3.1 Strands de protocolos

Por comportamiento del protocolo se entiende el intercambio de mensajes entre todos los participantes. A continuación, se puede observar un ejemplo de la especificación de un protocolo con la sintaxis propia de los strands:

```
eq STRANDS-PROTOCOL
   = :: r ::
[ nil |
      +(pk(B,A ; n(A,r))), -(pk(A,n(A,r) ; NB)), +(pk(B, NB)), nil ]

 &
     :: r' ::
[ nil |
      -(pk(B,A ; NA)), +(pk(A, NA ; n(B,r'))), -(pk(B,n(B,r'))), nil ]
  [nonexec] .
```

Como se aprecia, no deja de ser una ecuación con una sintaxis determinada. Tras el símbolo de igual puede haber una secuencia de términos entre los símbolos :: y :: . La razón de ser del tipo de datos Fresh, comentada previamente, es para ser puesta en entre los símbolos de puntos y así informar a la herramienta sobre los valores únicos de dichas variables.

Con motivo de entender mejor la esencia de estos Strands, en las Ilustración 1 se muestra el flujo de mensajes pero con una notación Alice y Bob mediante el uso de diagramas de interacción y cómo visualiza la recepción de los mensajes cada uno de los participantes. Dicha interacción consta del envío de dos mensajes y la recepción de uno. En el primero, Alice envía cifrado con la clave pública de Bob la identidad de Alice y un nonce generado por ella misma. En el segundo mensaje Alice recibe de Bob con la clave pública de Alice el nonce generado por ella y un nonce, secreto para Alice, generado por Bob. Por último, Alice envía a Bob el nonce generado por él mismo.

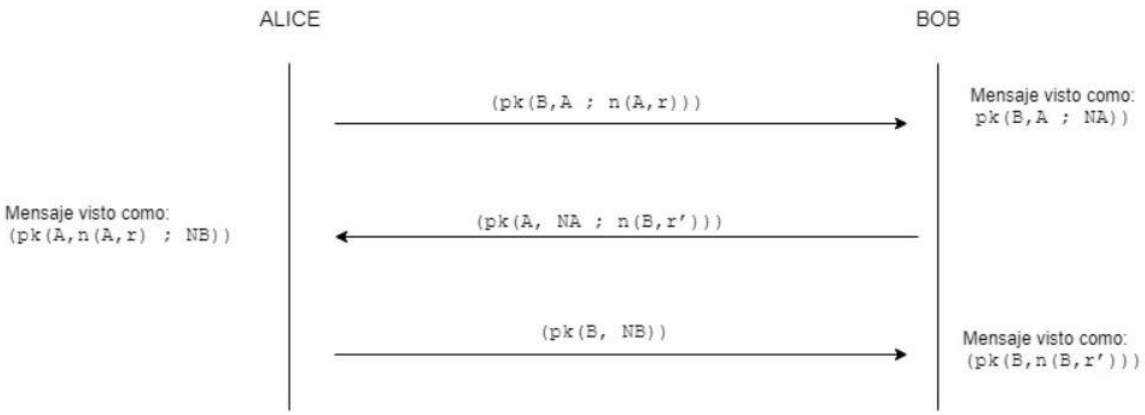

Ilustración 1. Diagrama de interacción de Alice y Bob.

2.1.3.2 Hablidades del intruso

Para especificar las habilidades, en base al protocolo, que puede tener el intruso se utilizan las reglas Dolev-Yao (D.Dolev & A.C.Yao, 1981) cuya especificación es mediante Strands de manera individual. Conceptualmente dentro de la herramienta, los participantes legítimos en un protocolo comparten un canal, como puede ser una red, y el atacante se encuentra dentro de este canal interceptando todos los mensajes. Es por lo que la sintaxis de los Strands siempre hay una recepción de mensaje previo al envío de este. Cada uno de estos Strands hace referencia a las habilidades que tiene el intruso respecto a las operaciones definidas y son propias del protocolo. Por tanto, Maude-NPA asume como únicas acciones posibles del atacante las indicadas en este apartado y no se pueden inferir nuevas.

El tipo de operaciones soportadas son muy variadas como puede ser la descomposición de mensaje en términos o la operación complementaria, agregar términos y crear y enviar un mensaje. El atacante también es capaz de generar valores frescos porque al utilizarse Strands la sintaxis permite indicar e informar variables de tipo Fresh.

A continuación, se puede apreciar un ejemplo de habilidades del intruso:

```
eq STRANDS-DOLEVYAO
 = :: nil :: [ nil | -(X), -(Y), +(X ; Y), nil ] &
   :: nil :: [ nil | -(X ; Y), +(X), nil ] &
   :: nil :: [ nil | -(X ; Y), +(Y), nil ] &
   :: nil :: [ nil | -(X), +(sk(i,X)), nil ] &
   :: nil :: [ nil | -(X), +(pk(Ke,X)), nil ] &
   :: nil :: [ nil | +(A), nil ]
[nonexec] .
```

2.1.3.3 Análisis del protocolo: Búsqueda de ataques.

En esta última parte del tercer módulo, se especifican los estados finales los cuales la herramienta comienza a buscar. Como se ha comentado previamente, esta herramienta realiza búsquedas de estados iniciales del protocolo desde el estado final indicado.

La especificación de un ataque no deja de ser una ecuación con un nombre especial, ATTACK-STATE con un número natural a modo de identificador. La sintaxis de estas ecuaciones es tal que así:

eq ATTACK-STATE(id) =

Strands || *Conocimiento del Intruso* || *Secuencia de mensajes* || *Información auxiliar* .

Cabe comentar la diferencia entre estados finales y estados iniciales. Se considera un estado inicial como aquel inmediatamente anterior a la ejecución de un protocolo donde aún no se han intercambiado mensajes y cada participante solamente conoce su información privada y la pública del resto de participantes. Por ejemplo, al inicio de un protocolo de intercambio de claves públicas y privadas, el cliente conoce la clave pública del servidor, pero en cambio no conoce su clave privada. Es por ello por lo que, a la hora de buscar ataques, se especifican patrones de estados finales deseados y así poder encontrar en caso de ser posible estados iniciales en los que poder ejecutar el protocolo de nuevo por parte del intruso y así demostrar alguna brecha de seguridad.

A continuación, se expone un ejemplo de patrón de estado de ataque:

```
eq ATTACK-STATE(0)
   = :: r ::
[ nil,
-(pk(b,a ; N)), +(pk(a, N ; n(b,r))), -(pk(b,n(b,r))) | nil ]
     || n(b,r) inI, empty
     || nil
     || nil
     || nil
  [nonexec] .
```

Este ataque extraído de los protocolos de ejemplo facilitados por la herramienta, exactamente del protocolo Needham-Schroeder sobre claves públicas y privadas, pretende buscar una ejecución en el que el cliente haya terminado su ejecución y el atacante haya aprendido el nonce generador por Bob y así poder suplantar su identidad. El ejemplo anterior es un patrón porque la variable N es una incógnita que la herramienta deberá instanciar con el valor que considere necesario.

3. Protocolo TLS

El protocolo TLS (Rescorla & Mozilla, 2018), de su nombre en inglés *Transport Layer Security*, es un protocolo criptográfico el cual permite crear comunicaciones seguras entre dos partes a través de capas no seguras como pueden ser HTTP o SMTP. Se busca con este protocolo acordar un escenario criptográfico y así garantizar confidencialidad e integridad en la transmisión de datos en la capa de aplicación. A ser exactos, este protocolo es el responsable de llevar a cabo de una manera segura el denominado *handshake*. Este saludo inicial da lugar a intercambio de mensajes entre dos entidades quienes no habían mantenido una comunicación previa y, por tanto, no tienen un secreto común como pueda ser una clave de sesión o certificados, por ejemplo.

Este *handshake* o saludo inicial se fundamente en el cifrado asimétrico para llegar a un acuerdo criptográfico y así poder llegar a compartir una clave secreta y simétrica a utilizar desde el momento en que ambos son conocedores de esta.

3.1 Objetivos y propiedades de seguridad

Desde el lanzamiento de este protocolo por parte de Netscape en 1995 este protocolo ha sido objeto de estudio e investigación por su extendido uso en los dispositivos de todo el mundo. Pero no es hasta el 2011 (Duong & Rizzo, 2011) donde se encuentra una fuerte vulnerabilidad en la que el atacante tiene la capacidad de descifrar mensajes enviados por un servidor seguro. Este ataque consta de obtener un mensaje enviado sin cifrar por parte del servidor, y más

adelante, el intruso vuelve a interceptar el mismo mensaje, pero cifrado.

Es por ello por lo que en la versión 1.3 del protocolo se toman decisiones con el fin de incrementar notablemente la seguridad de este. Medidas como obligar a autenticar al servidor y opcionalmente al cliente, así como establecer secretos que sólo conozcan ambas partes y no los atacantes pasivos y activos. De esta manera, se confirma que el *handshake* es resistente a la integridad de los mensajes sin que ningunos de los participantes sea consciente de esto. Por tanto, las propiedades de seguridad serán:

- Autenticación por parte del servidor (obligada).
- Autenticación mutua (opcional).
- Confidencialidad y propagación segura de la clave de sesión
- Integridad de los mensajes en el saludo inicial.

3.2 Versión 1.3

La última versión de este protocolo es la tercera. En ella se introducen mejoras tanto de seguridad como de eficiencia respecto a la versión anterior. Esta versión se lanza oficialmente en Octubre del 2015 y sus mejoras se pueden resumir en los siguientes puntos:

- **Cifrado del *handshake***: Esta mejora nace de reducir la cantidad de información visible para los atacantes. En la segunda versión del protocolo sólo se protege la información

una vez se ha finalizado el saludo inicial. Ahora se propone cifrar todo lo posible este saludo inicial también.

- **Contenido del *handshake***: Se propone una reestructuración del contenido de los mensajes del protocolo por motivos de eficiencia.

- **Latencia del *handshake***: En la segunda versión, se requería de dos intercambios de mensajes previa transmisión de comunicación cifrada de manera simétrica. Ahora, en la nueva versión se propone un solo intercambio ofreciendo incluso la posibilidad de enviar datos de la aplicación en el primer envío del mensaje. Supone una clara ventaja en recursos. También se ha ampliado el mecanismo original de la clave previamente compartida, denominada en la literatura como PSK. Esta extensión ofrece la posibilidad de reanudar una sesión creada previamente y de esta manera no tener que volver a intercambiar todos los mensajes.

- **Mecanismos de protección de registros**: En este apartado se propone un pequeño pero interesante detalle. En versiones anteriores se usa el conocido *MAC-then-encrypt* el cual se basa en ofrecer un código de autenticación a una de las partes (cliente, por ejemplo) y después comenzar a cifrar el contenido. En cambio, ahora se propone *encrypt-then-MAC* fundamentado en cifrar también esa transmisión de identificador.

3.3 Diseño de los nuevos *handshakes*

3.3.1 *Handshake* inicial

En la Ilustración 2 se puede encontrar el flujo de mensajes de un handshake inicial completo. Dentro del flujo de mensajes (Cremers, Horvat, Scott, & Merwe, 2016), aparecen mensajes dentro y fuera de símbolos como pueden ser llaves *{}* o corchetes [] así como algunos seguidos de asterisco *. Aquellos mensajes sin ningún tipo de símbolo se encuentran en texto plano, enviados sin cifrar por el canal de comunicación. Los mensajes dentro de las llaves se envían cifrados por claves del *handshake*, lo previamente comentados en estas páginas como claves asimétricas. Por otro lado, los mensajes enviados dentro de los corchetes son cifrados por la clave de sesión acordada entre ambas partes, es decir, simétrica. Los mensajes seguidos por el asterisco indican la no obligatoriedad de ser enviados.

Este *handshake*, respecto a la versión anterior, introduce la posibilidad por parte del servidor de rechazar la petición del cliente.

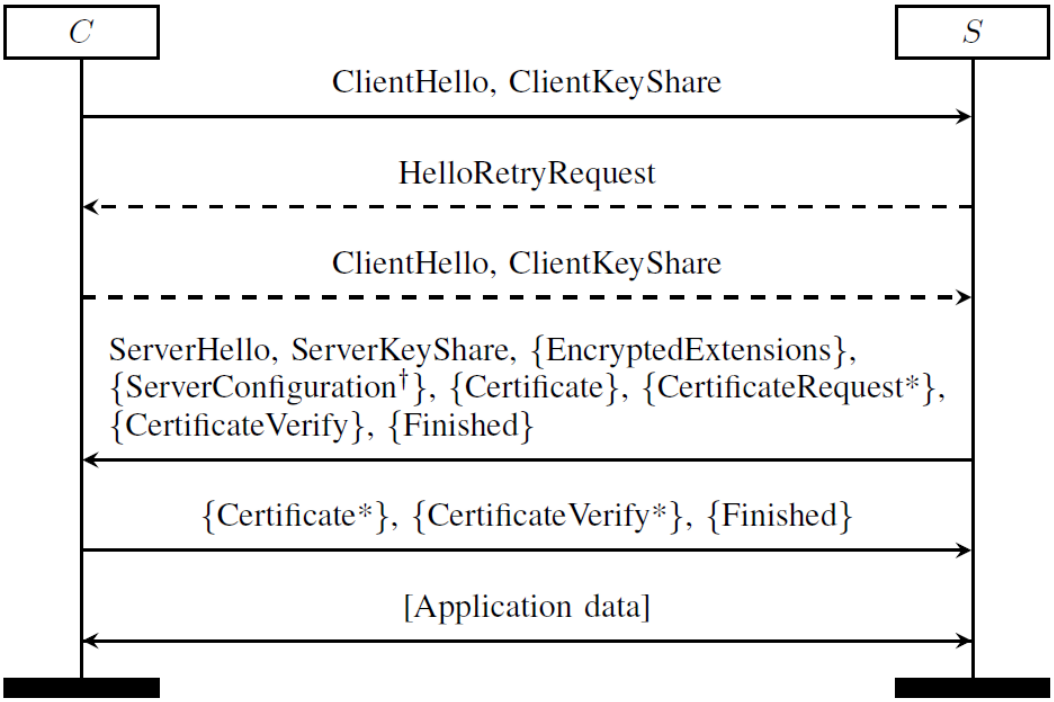

Ilustración 2. Handshake inicial.

3.3.2 0-RTT

Como se visualiza en la Ilustración 3, existe otra posibilidad de *handshake* soportada por la última versión del protocolo se centra en poder enviar datos ajenos al protocolo los cuales el cliente, en caso de ejecutar el handshake anterior, enviaría una vez acordada la clave de sesión. Se aprecian mensajes dentro de paréntesis *()*, indican mensajes cifrados con claves del tráfico del handshake. Estas son calculadas a partir de la clave pública semiestática del servidor y una clave pública temporal del cliente. Por lo demás, el flujo de mensajes es el mismo que el mismo que en el apartado 3.3.1.

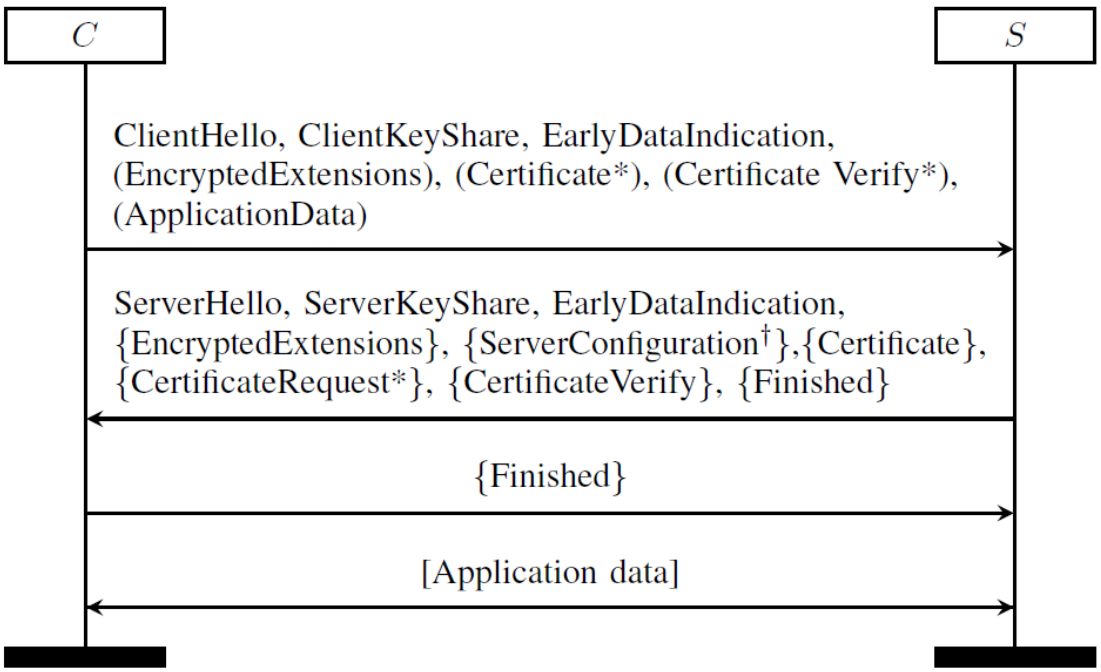

Ilustración 3. 0-RTT.

3.3.3 Reanudación de sesión y PSK.

Anteriormente, la reanudación de la sesión y el uso de una clave previamente compartida no se encuentran bajo el mismo flujo de mensajes. En cambio, ahora sí.

El flujo consta del *handshake* explicado en el punto 3.3.1 solo que previa transmisión de datos ajenos al protocolo, el servidor envía un secreto o *ticket* a ser usado en el futuro para la reanudación de la sesión. Así, si el cliente desea volver a establecer una conexión segura con el servidor existe la posibilidad de no realizar el saludo inicial completo pues una vez hecho, el servidor ya ha verificado todo lo pertinente y sólo debe hacer referencia al secreto que el servidor le ofreció en su momento. En la Ilustración 4 se puede visualizar el flujo de mensaje asociado a este *handshake.*

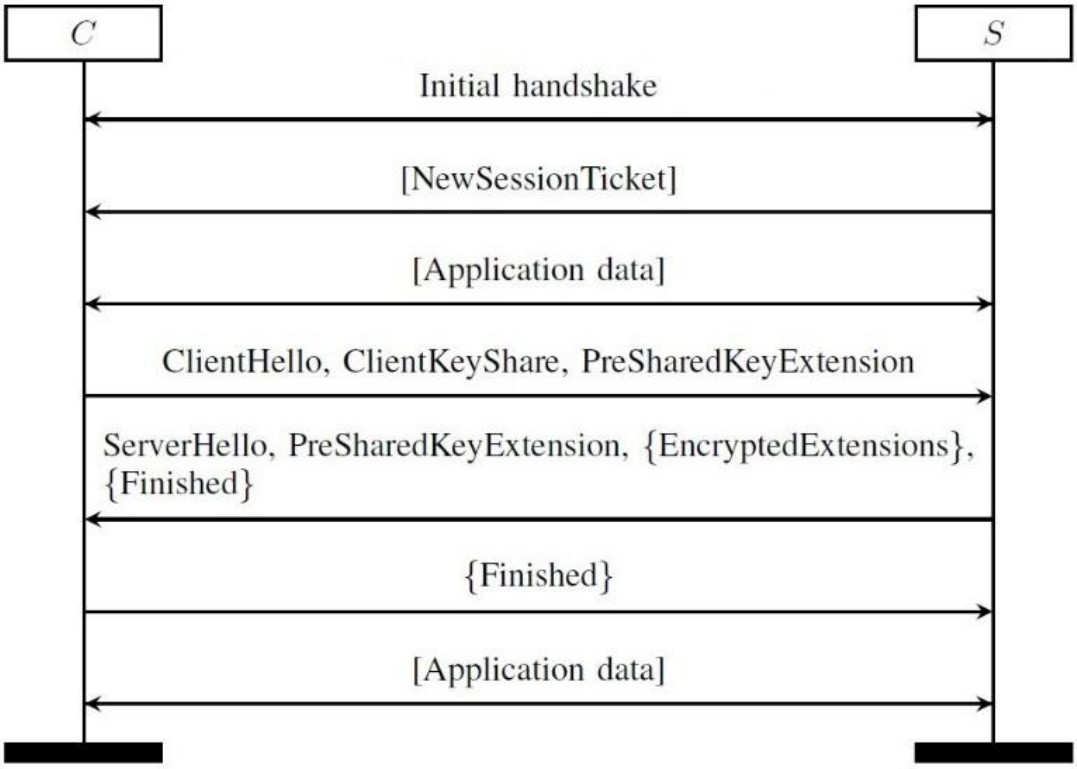

Ilustración 4. Reanudación de sesión y PSK.

4. Análisis del protocolo TLS 1.3 en Tamarin

Tamarin es una herramienta de modelado formal orientado a la verificación de protocolos de criptografía muy interesante dentro del campo del modelado simbólico desarrollada en *ETH* (Universidad Politécnica Federal) de Zurich. Exactamente es un demostrador de teoremas, *theorem prover* en inglés. Ofrece al usuario la posibilidad de especificar complejas propiedades de seguridad, modelar primitivas criptográficas por medio de teorías ecuacionales y también modelar protocolos con la peculiaridad de necesitar almacenar información de estado local. El tipo de teorías ecuacionales permitidas por la herramienta son del tipo de subtérminos convergentes. Además, existen teorías incorporadas por la herramienta como puede ser la exponenciación *Diffie-Hellman*, multiconjuntos y teorías ecuacionales con la propiedad de variante finita desarrollada para el lenguaje Maude, el cual se invoca de manera rutinaria para diversas tareas.

Esta herramienta toma como entrada un modelo de un protocolo criptográfico usando una notación de reescrita de multiconjutos de predicados el cual indica las acciones de los participantes, una especificación de las acciones del adversario y una especificación de las propiedades deseadas a cumplir por parte del protocolo. Una de las ventajas ofrecidas por Tamarin es la prueba automática de las propiedades teniendo en cuenta muchas sesiones conviviendo al mismo tiempo. Pues es conocido que muchas brechas de seguridad aparecen fruto de vulnerabilidades de protocolos que no tienen en cuenta sesiones paralelas.

4.1 Modelado de protocolos en Tamarin.

La especificación en Tamarin (Cremers, Dreier, & Sasse, Manual de Tamarin Prover, 2019) se fundamenta en tres niveles: Reglas, Hechos y Términos. Esta sección se centrará en las dos primeras.

En lo referido a las reglas, son reglas con parte izquierda y derecha típicas de la reescritura de términos. Y, como se ha comentado previamente, esta herramienta trabaja con reglas de multiconjuntos. Estas reglas se utilizan para avanzar en la ejecución del protocolo sólo que el hecho de "avanzar" en la ejecución se da cuando las reglas necesarias están presentes dentro del multiconjunto. Por tanto, un ejemplo de regla puede ser:

rule ExampleRule:

[F(a)]

--[F(b)]->

[F(c)]

En la que las funciones entre corchetes se refieren, son hechos. Se interpreta la regla como el consumo del hecho F(a) y reescrito a F(c). Los corchetes situados dentro de la propia flecha indicando qué parte izquierda se consume por qué parte izquierda, se encuentran los denominados *action facts*. Estos hechos, ofrecen la posibilidad de crear, consumir o realizar llamadas a funciones en el momento se realiza la reescritura.

En lo referido a los hechos, se puede encontrar diferentes tipos de hechos usados para modelar los diferentes tipos de comunicaciones. Los más usados son:

- ***In()*** = Función utilizada para indicar la recepción de mensajes por el canal. Cabe comentar que el atacante tiene acceso al canal y, si un mensaje pasa por el canal, será interceptado por el atacante.

- ***Out()*** = Función complementaria a la anterior, indica envío de mensajes por el canal.

- ***Fr()*** = Función utilizada para generar valores frescos (únicos) a lo largo de toda la ejecución de la herramienta para un protocolo determinado. Similar a los datos de tipo Fresh de Maude-NPA vistos en la sección 2.1.1.

- ***Name(a1, … , an)*** Hechos normales, producidos y consumidos por una regla. Una vez aparecen en una parte izquierda, se consumen y no pueden ser usados de nuevo.

- ***!Name(a1,…,an)*** = Misma filosofía que el hecho anterior, solo que este se guarda en memoria local del proceso y pueden ser usados de nuevo. Son denominados hechos de persistencia.

- ***- [actionFacts] ->*** = Hechos de acción, utilizados para indicar que el se supone que el mensaje es secreto cuando se realiza una reescritura.

Una vez comentada cómo se realiza ese avance a través de la regla, es preciso indicar también cómo se proporcionan las propiedades de seguridad asociadas al protocolo y para determinar si se cumplen o no, se utilizan los lemmas los cuales se prueban conforme avanza la ejecución. La sintaxis utilizada para declarar los lemmas es:

- ***Ex*** = Cuantificador existencial

- ***All*** = Cuantificador universal

- **#** = Variables temporales

- **&** = Conjunción lógica

- **|** = Disyunción lógica

- **==>** = Implicación lógica

- ***not*** = Negación lógica

- *f* **@** *i* = símbolo utilizado para modelar el tiempo, exactamente utilizado para indicar que el hecho *f* tiene lugar en el momento de tiempo del hecho *i*.

Por tanto, un ejemplo de lemma puede ser:

```
lemma Client_session_key_secrecy:
"not(Ex S k #i #j.
SessKeyC(S, k) @ #i & K(k) @ #j & not(Ex #r. LtkReveal(S) @ r))"
```

Por ultimo, cabe comentar que este lenguaje permite definir variables locales mediante la sintaxis *let* ... *in* seguido de las reglas comentadas previamente.

4.2 Modelado del protocolo TLS 1.3 en Tamarin.

El primer paso en el proceso de modelado (Cremers, Horvat, Scott, & Merwe, 2016) es construir una abstracción del *handshake* con un nivel de detalle lo suficiente como para poder realizar un análisis exhaustivo de las propiedades del protocolo. Por ello, en el modelo realizado en Tamarin se aprecia un balance entre un modelo potencialmente complejo y preciso, pero centrándose sólo en los mensajes y algoritmos criptográficos más importantes del protocolo.

4.2.1 Diseño y codificación del modelo en Tamarin.

De entre estos aspectos más importante de la codificación de TLS 1.3 en Tamarin se puede recalcar:

- **Criptografía perfecta**: Se parte de una base muy sólida en lo referido a primitivas criptográficas de la herramienta,

conocidas como *builtins*. Por ejemplo, se asume la confidencialidad absoluta de los mensajes cifrados, firmas de mensajes y MACs inolvidables, funciones hash con ninguna probabilidad de colisionar (dos funciones generen el mismo hash) y que todas las partes generan valores verdaderamente aleatorios cuando es necesario.

- **Parámetros de configuración**: Estos parámetros dentro de la ejecución real del protocolo sirven para acordar la *suite* criptográfica entre cliente y servidor. Esta *suite* hace referencia, por ejemplo, al algoritmo de encriptación a usar durante la sesión. En el modelo se propone obviar estos parámetros y usar siempre los mismos pues a la hora del análisis, este aspecto no es el más importante.

- **Mensajes de alerta**: Este aspecto es otro el cual se decide no modelar por motivos de exploración de la búsqueda de ataques una vez el modelo está hecho. El protocolo real implementa y maneja mensajes de error para terminar de manera inmediata la ejecución y el intercambio de mensajes. Dentro del modelo, se considera el hecho de lanzar un mensaje de error, es decir, un escenario no esperado dentro del intercambio de mensajes como el hecho de darse una tesitura dentro del multiconjunto de reglas que no permiten avanzar en la ejecución del protocolo. Por tanto, a la hora de la exploración en el campo de búsqueda, será un camino que no lleva a ninguna parte y desechado en el análisis.

- **Sobre-aproximaciones**: Si en algunos momentos se decide no modelar ciertos aspectos, en otros se realizan asunciones

sobre la ejecución del modelo. Un ejemplo puede ser en el caso de 0-RTT en la que siempre el servidor envía el mensaje *CertificateRequest* cuando en la especificación del protocolo es un mensaje opcional. Sin embargo, el cliente puede responder o no a esa petición. Pese a estas, se afirma haber visto comportamientos equivalentes y de esta manera, se disminuye la complejidad en cierta medida, en la codificación.

En lo referido a la codificación de dicho modelo (Archive with TLS 1.3 Rev 10 models and property specificacions, 2015), se ha codificado cualquier posible ejecución de este protocolo pues como se pueden ver en las Ilustraciones 1, 2 y 3; hay mensajes opcionales y obligatorios y esto pueden suponer interacciones diferentes en función de ellos. Es por lo que en las Ilustraciones 5 y 6 se puede observar cada una de las reglas codificadas tanto para cliente como para el servidor.

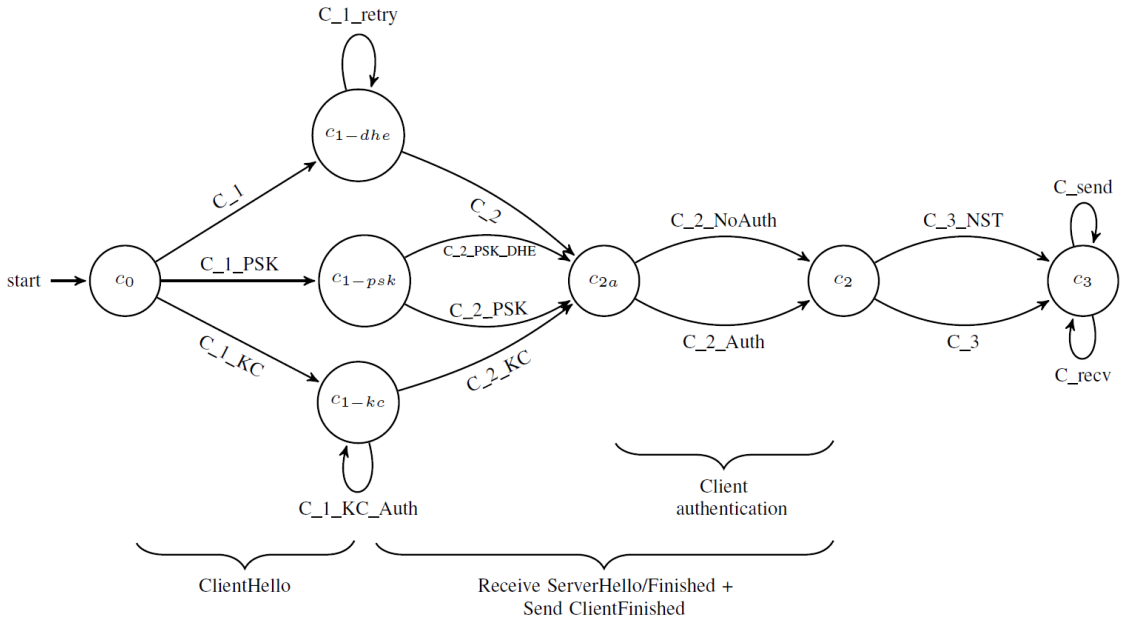

Ilustración 5. Reglas codificadas para el Cliente.

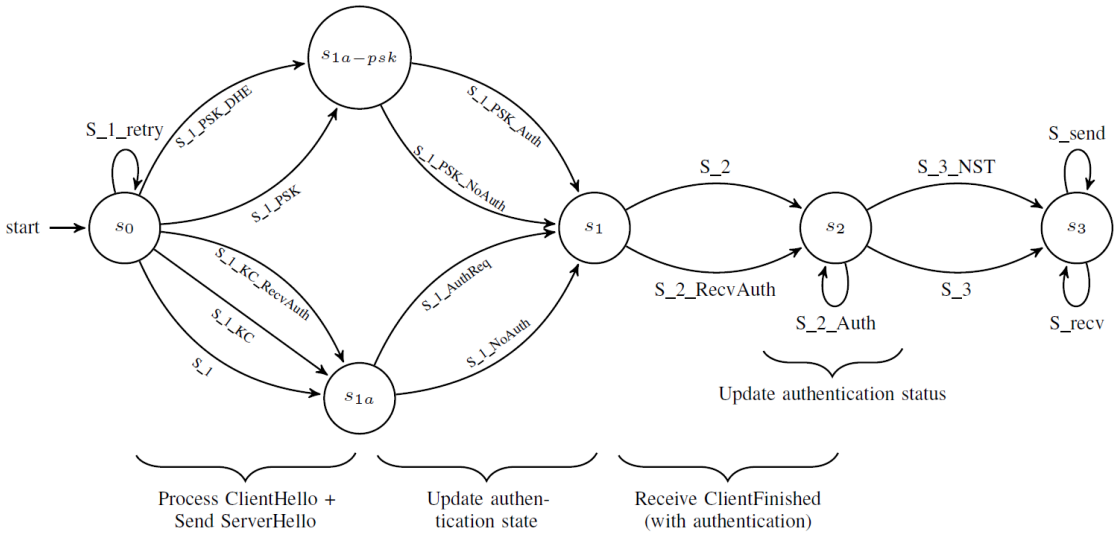

Ilustración 6. Reglas codificadas para el Servidor.

Por poner en situación, se procede a explicar la Ilustración 5, siendo la Ilustración 6 complementaria a ésta. El cliente puede iniciar tres tipos de *handshake*: un saludo inicial (EC)DHE (C_1), un saludo inicial del tipo PSK (C_1_PSK) y otro saludo inicial 0-RTT (C_1_KC). En el modo (EC)DHE, el servidor puede rechazar al cliente por diferentes motivos como puede ser que se encuentre dentro de una lista de Cliente con acceso denegado o por utilizar una configuración criptográfica incorrecta para el Servidor (C_1_retry). Además, el cliente puede no autenticarse si propone un saludo inicial del tipo 0-RTT (C_1_KC_Auth).

Como se observa en las Ilustraciones 5 y 6, el saludo inicial PSK tiene dos modos diferentes: PSK simple (C_2_PSK) y PSK con DHE (C_2_PSK_DHE). Se modela el servidor para que siempre solicite la autenticación del cliente, pese a ser opcional por parte del cliente. Si el cliente decide autenticarse, envía los mensajes de autenticación junto con el mensaje *Finished* del cliente (C_2_Auth). De lo contrario, solo se transmite el mensaje *Finished* (C_2_NoAuth). El *handshake* concluye cuando el cliente recibe un nuevo ticket de sesión (C_3_NST), que se puede utilizar para

reanudar la conexión mediante PSK, o, simplemente, no hacer nada (C_3). El cliente puede entonces enviar (C_Send) y recibir (C_Recv) cualquier número finito de mensajes de datos de la aplicación.

Cabe comentar la gran cantidad de reglas y consecuentes posibles caminos existentes en este modelo propuesto en Tamarin. Esto nace de la inexistencia de estructuras condicionales en base a datos de variables o detección de ejecuciones concretas. Esto provoca la generación de más reglas y así poder abarcar todas las posibles ejecuciones.

4.2.2 Análisis formal del protocolo

Tras codificar el modelo, se realiza el análisis. Este se centra en probar las propiedades existentes tanto de secreto como de confidencialidad. Se parte de un ecosistema que consta de una red donde los participantes, en este caso cliente y servidor, realizan el intercambio de mensajes y hay un atacante el cual es capaz de comprometer las claves a largo plazo de los participantes. La manera pues de plasmar estas propiedades es a través de los lemmas, comentados con anterioridad en este documento. La característica del lemma es que debe ser cumplido siempre a lo largo de la ejecución. Pues si en algún momento no se cumple un lemma, es decir que una propiedad no se cumple en un determinado momento, se encuentra una vulnerabilidad.

Así pues, en lo referido a las reglas de secreto se plasman dos propiedades. La primera hace referencia a la perfecta transmisión de un secreto/clave en presencia de un atacante. Se codifica de la siguiente manera:

```
lemma secret_session_keys:
(1) "All actor peer role k #i.
(2) SessionKey(actor, peer, role, <k, 'authenticated'>)@i
(3) & not ((Ex #r. RevLtk(peer)@r & #r < #i)
|(Ex #r. RevLtk(actor)@r & #r < #i))
(4) ==> not Ex #j. KU(k)@j"
```

Esta propiedad hace referencia a todas las posibles ejecuciones(*All*). (1): si una clave de sesión autenticada k es aceptado (codificado por la aparición de *SessionKey*) (2), y el adversario no ha revelado el largo plazo claves privadas del actor o del compañero antes de que se acepte la clave de sesión (3), entonces el adversario no puede derivar la clave k (4).

La segunda propiedad hace referencia a que las claves de datos temprana del cliente son seguras siempre que no se revele la clave privada a largo plazo del servidor. Se plasman de la siguiente manera:

```
lemma secret_early_data_keys:
(1) "All actor peer k #i.
(2) EarlyDataKey(actor, peer, 'client', k)@i
(3) & not (Ex #r. RevLtk(peer)@r)
(4) ==> not Ex #j. KU(k)@j"
```

Se interpreta como que cada vez (1) que un cliente registra que ha producido una clave de datos temprana (2) y las claves privadas a largo plazo de los pares no se ven comprometidas (3), entonces el adversario no conoce la clave de datos temprana (4).

Por otro lado, en lo referido a las claves de autenticación se proponen un total de cuatro. La autenticación en este tipo de escenarios se entiende como el acuerdo de ciertos valores tales como identidades de agente y nonces. Por tanto, la primera de las propiedades descansa en esta definición, cuando un cliente comparte nonces con otro participante:

```
lemma entity_authentication:
(1) "All actor peer nonces #i.
(2) CommitNonces(actor, peer, 'client', nonces)@i
(3) & not (Ex #r. RevLtk(peer)@r)
(4) ==> (Ex #j peer2.
(5) RunningNonces(peer, peer2, 'server', nonces)@j
(6) & #j < #i)"
```

Que quiere decir que cuando el cliente registra que ha observado ciertos nonces al final de su hilo y piensa que se está comunicando con un participante específico (1,2), y la clave privada a largo plazo de este par no ha sido comprometida (3), entonces hay un hilo (4) de ese participante en el rol del servidor que coincide con el nonce (5) anterior (6).

La segunda propiedad codifica que el actor y el compañero no solo están de acuerdo sobre los nonces y quién es el servidor, sino que también están de acuerdo con la transcripción completa:

```
lemma transcript_agreement:
"All actor peer transcript #i.
CommitTranscript(actor, peer, 'client', transcript)@i
& not (Ex #r. RevLtk(peer)@r)
==> (Ex #j peer2.
RunningTranscript(peer, peer2, 'server', transcript)@j
& #j < #i)"
```

La tercera y cuarta propiedad solo ofrecen garantías en los escenarios en los que sólo el servidor se autentifica. Por tanto, la tercera propiedad representa la garantía de autenticación para el servidor, que se puede obtener si el servidor solicita autenticación mutua mutuamente o sólo solicita la autenticación del cliente:

```
lemma mutual_entity_authentication:
"All actor peer nonces #i.
CommitNonces(actor, peer, 'server', nonces)@i
& not ((Ex #r. RevLtk(peer)@r)
|(Ex #r. RevLtk(actor)@r))
==> (Ex #j.
RunningNonces(peer, actor, 'client', nonces)@j
& #j < #i)"
```

La cuarta propiedad es análoga a la segunda y garantiza que el servidor obtenga una garantía sobre el acuerdo de la transcripción con el cliente, después de que se haya autenticado:

```
lemma mutual_transcript_agreement:
"All actor peer transcript #i.
CommitTranscript(actor, peer, ’server’, transcript)@i
& not ((Ex #r. RevLtk(peer)@r)
|(Ex #r. RevLtk(actor)@r))
==> (Ex #j.
RunningTranscript(peer, actor, ’client’, transcript)@j
& #j < #i)"
```

4.3 Conclusiones del análisis

Este modelo cubre muchas posibles interacciones complejas entre los diversos modos, para un número ilimitado de sesiones. Cuando se combina con las propiedades de seguridad en esta sección, esto genera problemas de verificación muy complejos. Sin embargo, se logra probar con éxito las principales propiedades. Los resultados implican la ausencia de una gran clase de ataques, muchos de los cuales no están cubiertos por otros métodos de análisis de otros tipos. Este es un resultado muy alentador, ya que muestra que el diseño es sólido.

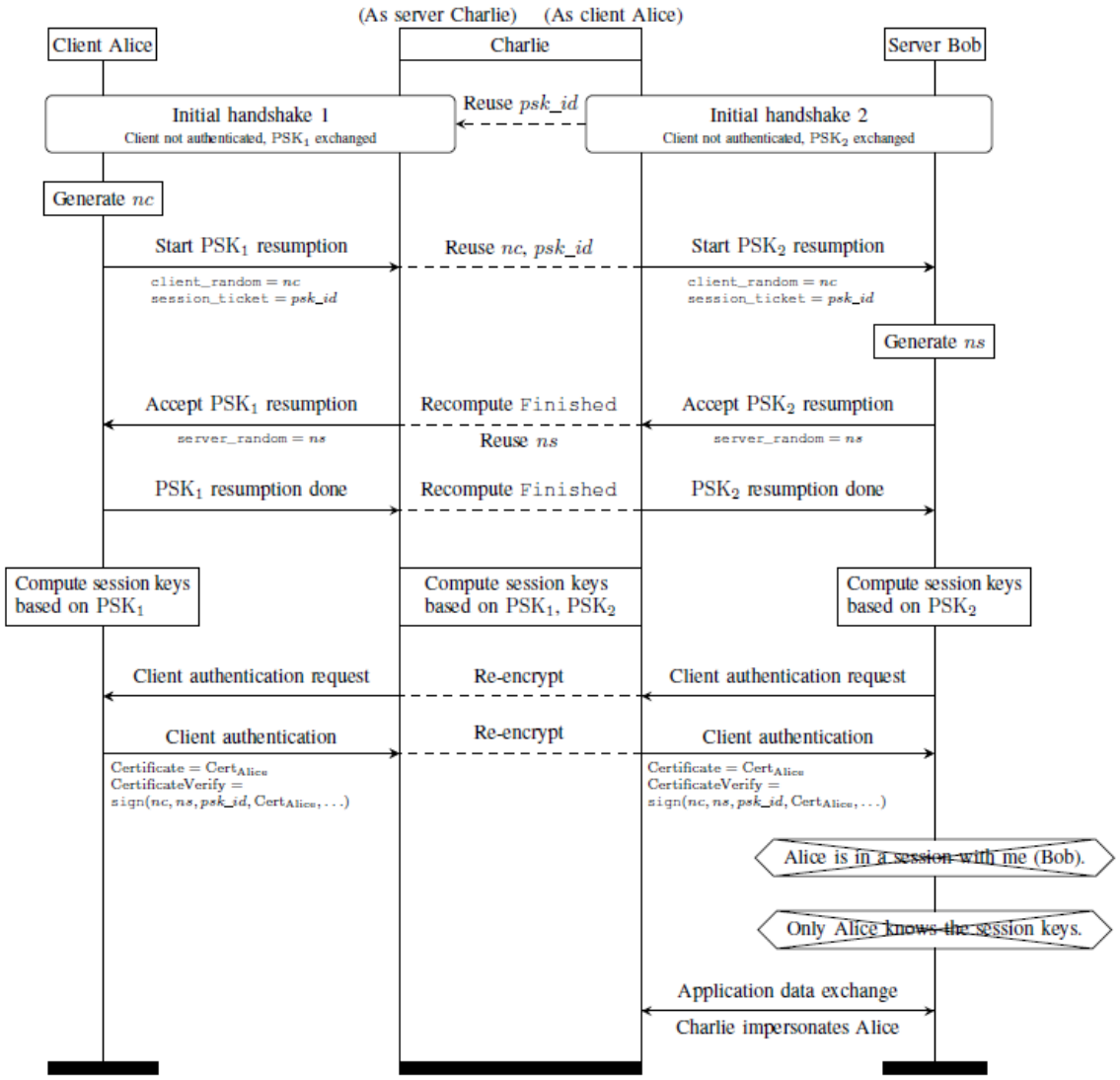

Ilustración 7. Ataque Man-in-the-middle encontrado.

Pese a no haber encontrado ataques en el modelo, si se realiza una extensión de este y se permite autenticación mediante certificados (o firmas digitales) en reanudaciones de sesiones mediante PSK que corresponde a la Ilustración 4, entonces la herramienta encuentra un ataque. Este ataque, expuesto en la Ilustración 7, se trata de un *Man-in-the-middle* en el que las propiedades de autenticación dejan de cumplirse.

El ataque, por tanto, consta de tres partes. En la primera Alice inicia una conexión con Charlie, y Charlie inicia una conexión con Bob. En ambas conexiones, se establece un PSK. En este punto, ambas conexiones son lícitas. Alice comparte una PSK llamada PSK1 con Charlie, y Charlie comparte una PSK llamada PSK2 con Bob. Cabe comentar que Charlie se asegura de que el ticket de sesión (*psk_id*) sea el mismo en ambas conexiones al reproducir el valor obtenido de Bob.

En la segunda parte, se realiza una reanudación de la conexión mediante PSK. Al realizarlo, Alice y Charlie comparten una clave simétrica y Charlie y Bob, comparten otra clave simétrica. Ambas claves derivadas de PSK1 y PSK2 respectivamente.

En la tercer y última parte, se requiere la autenticación del cliente por parte de Bob. Es por lo que Charlie al estar en medio de ambos, solicita también autenticación a Alice. Esta envía su certificado a Charlie y este, envía el certificado de Alice a Bob. Bob, asume que es Alice quien envía el certificado. Y es así como Charlie, suplante la identidad de Alice consiguiendo realizar un ataque *Man-in-the-middle*.

5. Simplificación del modelo de TLS 1.3 en Maude-NPA.

El objetivo de este trabajo es poder modelar el protocolo TLS en su tercera versión en la herramienta Maude-NPA y así poder analizar las propiedades de seguridad tal y como se realiza con la herramienta Tamarin. En este capítulo se expone cómo se ha llegado al modelo presentado y el posterior análisis de las propiedades de seguridad junto a las conclusiones.

La primera de las decisiones tomadas a la hora de codificar modelo ha sido el nivel de detalle y en qué aspectos centrar el esfuerzo. Es por lo que se decide codificar a alto nivel el protocolo con el flujo mínimo de interacción entre los participantes y así poder plantear las propiedades tanto de secreto como de autenticación. Respecto a la *suite* criptográfica se decide un entorno *Diffie-Hellman* y el flujo de mensajes consta del intercambio de nonces para establecer la clave de sesión y los secreto transmitidos por el servidor (*psk_id*) y el cliente (su certificado).

5.1 Símbolos del protocolo y propiedades algebraicas

En primer lugar, se encuentran los tipos, subtipos y operadores. Estos son los necesarios para poder generar los nonces y los secretos de los respectivos participantes, la capacidad de exponenciación y de cifrado y descifrado de mensajes.

```
--- Sort Information
 sorts Name Nonce NeNonceSet Gen Exp Key GenvExp Secret .
 subsort Gen Exp < GenvExp .
 subsort Name NeNonceSet GenvExp Key Secret < Msg .
 subsort Exp < Key .
 subsort Name < Public .
 subsort Gen < Public .

 --- Secret
 op sec : Name Fresh -> Secret [frozen] .

 --- Nonce operator
 op n : Name Fresh -> Nonce [frozen] .

 --- Intruder
 ops a b i : -> Name .

 --- Encryption
 op e : Key Msg -> Msg [frozen] .
 op d : Key Msg -> Msg [frozen] .

 --- Exp
 op exp : GenvExp NeNonceSet -> Exp [frozen] .

 --- Gen
 op g : -> Gen .

 --- NeNonceSet
 subsort Nonce < NeNonceSet .
 op _*_ : NeNonceSet NeNonceSet -> NeNonceSet [frozen assoc comm] .

 --- Concatenation
 op _;_ : Msg Msg -> Msg [frozen gather (e E)] .

 --- Special messages
 ops serverFin clientFin : -> Msg .
```

En lo que concierne a las propiedades algebraicas, encontramos dos tipos. El primero hace referencia a la criptografía asimétrica de *Diffie-Hellman* en la que se aprecia la propiedad matemática de los exponentes propia de este tipo de cifrado. Por otro lado, se aprecia cómo es la anulación del cifrado de un mensaje con el descifrado de este, así como la propiedad complementaria a esta.

```
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] .
```

5.2 Especificación de los *Strands*.

Una vez definidos los elementos mínimos del protocolo, tipos y operadores, es momento de especificar cómo será este comportamiento, tanto para los roles legítimos como para el atacante.

5.2.1 Habilidades del intruso. Reglas Dolev-Yao

Estas habilidades del atacante se definen bajo el modelo Dolev-Yao (D.Dolev & A.C.Yao, 1981) y como se puede ver más abajo el nivel de detalle es clave para poder indicar qué acciones, y solo ellas, es capaz de hacer dicho atacante. Se observan acciones de descomposición de mensajes o agrupación y también acciones de cifrado, descifrado y generación tanto de nuevos nonces como de secretos. A continuación, se puede observar cómo sería una especificación de habilidades del intruso mediante reglas Dolev-Yao:

```
eq STRANDS-DOLEVYAO =
   :: nil :: [ nil | -(M1 ; M2), +(M1), nil ] &
   :: nil :: [ nil | -(M1 ; M2), +(M2), nil ] &
   :: nil :: [ nil | -(M1), -(M2), +(M1 ; M2), nil ] &
   :: nil :: [ nil | -(Ke), -(M), +(e(Ke,M)), nil ] &
   :: nil :: [ nil | -(Ke), -(M), +(d(Ke,M)), nil ] &
   :: nil :: [ nil | -(NS1), -(NS2), +(NS1 * NS2), nil ] &
   :: nil :: [ nil | -(GE), -(NS), +(exp(GE,NS)), nil ] &
   :: r ::   [ nil | +(n(i,r)), nil ] &
   :: r ::   [ nil | +(sec(i,r)), nil ] &
   :: nil :: [ nil | +(g), nil ] &
   :: nil :: [ nil | +(A), nil ]
[nonexec] .
```

5.2.2 Strands del protocolo

Lo referido al comportamiento del protocolo aquí se expone lo comentado al principio de este capítulo. Se ha simplificado el protocolo para realizar el intercambio de nonces y secretos. La razón de la simplificación es el hecho de poder realizar un análisis posterior donde se encuentra el ataque especificado en el trabajo realizado en la herramienta Tamarin del capítulo tercero y las respectivas propiedades de seguridad.

El modelo responde a una ejecución concreta del protocolo en la que se acuerda una sesión, el servidor envía el ticket (un secreto sólo conocido por él) y acto seguido el cliente envía su certificado (un secreto sólo conocido por él) para así autenticarse.

```
eq STRANDS-PROTOCOL =
  :: ra, CA:: --- Cliente
  [nil |
  +(A ; exp(g, n(A, ra))),
  -(B ; e(BE, serverFin)),
  +(A ; e(exp(g, n(A, ra)), clientFin)),
  -(B ; e(exp(BE, n(A, ra)), psk_id)),
  +(A ; e(exp(BE, n(A, ra)), psk_id ; sec(A, CA))), nil] &

  :: rb, pskId :: --- Servidor
  [nil |
  -(A ; AE),
  +(B ; e(exp(g, n(B, rb)), serverFin)),
  -(A ; e(AE, clientFin)),
  +(B ; e(exp(AE, n(B, rb)), sec(B, pskId))),
  -(A ; e(exp(AE, n(B, rb)), sec(B, pskId); cert_Alice)),nil]
[nonexec] .
```

5.2.3 Gramáticas.

Por motivos de eficiencia en la exploración del campo de búsqueda, se han añadido gramáticas extra además de las existentes internamente en Maude-NPA, (Escobar, Meadows, & Meseguer, 2017) tal y como se puede observar más adelante. El motivo de las gramáticas es limitar el espacio de búsqueda mediante la eliminación de caminos que se puede garantizar que nunca alcanzarán un estado inicial. Esta gramática se conforma de dos semillas. La primera semilla trata acerca de los nonces *n(a,r)* y *n(b,r)* nunca podrán ser aprendidos independientemente si se encuentran o no junto a otros nonces. La segunda semilla informa sobre la imposibilidad de aprender el mensaje *e(AE, sec(b, psk_id))* si previamente no se ha aprendido el nonce *AE*. En el momento de la búsqueda, si la herramienta tiene esta información, supone una fuerte orientación el mero hecho de saber que puede eliminar espacios de búsqueda donde no va a encontrar ninguna solución.

```
eq EXTRA-GRAMMARS
 = (grl empty => (NS * n(a,r)) inL . ;
    grl empty => n(a,r) inL . ;
    grl empty => (NS * n(b,r)) inL . ;
    grl empty => n(b,r) inL .
    ! S2 )
   |
   (grl AE notInI => e(AE, sec(b, psk_id)) inL .
    ! S1 )
[nonexec] .
```

5.3 Análisis del protocolo.

En esta sección del capítulo se pretende exponer cómo ha sido el análisis del modelo en búsqueda de demostrar si son seguras o no las propiedades de secreto y autenticación planteadas, tal y como se realiza en el trabajo hecho en Tamarin.

Se plantean cinco ejecuciones a realizar por Maude-NPA. La primera es una ejecución regular explicada más adelante. Las propiedades de los apartados 5.3.1 y 5.3.2 son de propiedades de autenticación tanto por parte del servidor como del cliente. Las propiedades de los otros dos apartados, 5.3.3 y 5.3.4, tratan sobre propiedades de secreto para ambos participantes, cliente y servidor. Cabe comentar que estos últimos dos ataques no han sido descritos en el estudio realizado en Tamarin (Cremers, Horvat, Scott, & Merwe, 2016) y, por tanto, se desconoce si dicha herramienta es capaz de encontrarlos.

5.3.0 Ejecución regular

La razón de esta ejecución es comprobar si el modelo codificado en la sección 5.2.2 tiene el comportamiento deseado. Se trata de una ejecución forzada en la que como configuración final se indica la configuración final deseada y, en caso de encontrar una configuración inicial se demuestra que el comportamiento es el esperado. A continuación, se expone esta ejecución regular:

```
eq ATTACK-STATE(0) = --- Regular execution
 :: ra, CA :: --- Cliente
 [nil,
 +(a ; exp(g, n(a, ra))),
 -(b ; e(exp(g, n(b, rb)), serverFin)),
 +(a ; e(exp(g, n(a, ra)), clientFin)),
 -(b ; e(exp(exp(g, n(b, rb)), n(a, ra)), sec(b, pskId))),
 +(a ; e(exp(exp(g, n(b, rb)), n(a, ra)), sec(b, pskId) ; sec(a, CA)))
 |  nil ] &

 :: rb, pskId :: --- Servidor
 [nil,
 -(a ; exp(g, n(a, ra))),
 +(b ; e(exp(g, n(b, rb)), serverFin)),
 -(a ; e(exp(g, n(a, ra)), clientFin)),
 +(b ; e(exp(exp(g, n(a, ra)), n(b, rb)), sec(b, pskId))),
 -(a ; e(exp(exp(g, n(a, ra)), n(b, rb)), sec(b, pskId); sec(a, CA)))
|  nil ]
     || empty
     || nil
     || nil
     || nil
  [nonexec] .
```

Tras realizar una búsqueda de cinco niveles de profundidad, la herramienta encuentra una configuración inicial, tal y como se esperaba:

```
Maude> red summary(0,0) .
result Summary: States>> 1 Solutions>> 0
Maude> red summary(0,1) .
result Summary: States>> 2 Solutions>> 0
Maude> red summary(0,2) .
result Summary: States>> 4 Solutions>> 0
Maude> red summary(0,3) .
result Summary: States>> 9 Solutions>> 0
Maude> red summary(0,4) .
result Summary: States>> 19 Solutions>> 0
Maude> red summary(0,5) .
```

```
result Summary: States>> 38 Solutions>> 1
```

A continuación se muestra la ejecución que encuentra Maude-NPA:

```
+(a ; exp(g, n(a, #0:Fresh))),
-(a ; exp(g, n(a, #0:Fresh))),
+(b ; e(exp(g, n(b, #2:Fresh)), serverFin)),
-(b ; e(exp(g, n(b, #2:Fresh)), serverFin)),
+(a ; e(exp(g, n(a, #0:Fresh)), clientFin)),
-(a ; e(exp(g, n(a, #0:Fresh)), clientFin)),
+(b ; e(exp(g, n(a, #0:Fresh) * n(b, #2:Fresh)), sec(b,
#3:Fresh))),
-(b ; e(exp(g, n(a, #0:Fresh) * n(b, #2:Fresh)), sec(b,
#3:Fresh))),
+(a ; e(exp(g, n(a, #0:Fresh) * n(b, #2:Fresh)), sec(b,
#3:Fresh) ; sec(a, #1:Fresh))),
-(a ; e(exp(g, n(a, #0:Fresh) * n(b, #2:Fresh)), sec(b,
#3:Fresh) ; sec(a, #1:Fresh)))
```

Con motivo de facilitar la comprensión al lector, se propone en la Ilustración 8 la misma salida, pero plasmada en un diagrama de interacción con los dos participantes.

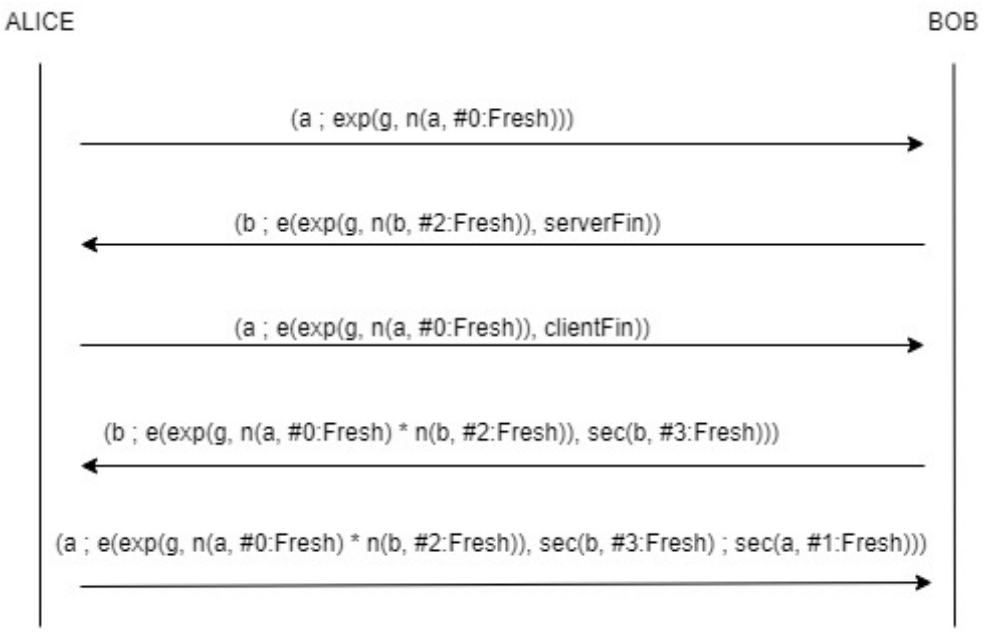

Ilustración 8. Expresión regular. Ejecución esperada del modelo.

5.3.1 Autenticación del Servidor

En esta primera propiedad de autenticación se busca encontrar una ejecución en la que el servidor haya finalizado todo su flujo de mensajes y el intruso haya conseguido suplantar la identidad del Cliente. Es decir, conseguir un ataque *Man-in-the-middle* suplantando el Cliente frente al Servidor. A continuación, se puede apreciar cómo se plasma el análisis en forma de ataque en Maude-NPA:

```
eq ATTACK-STATE(1) = --- MITM attack
    :: rb, pskId :: --- Servidor
    [nil,
           -(a ; AE),
           +(b ; e(exp(g, n(b, rb)), serverFin)),
           -(a ; e(AE, clientFin)),
           +(b ; e(exp(AE, n(b, rb)), sec(b, pskId))),
           -(a ; e(exp(AE, n(b, rb)), sec(b, pskId); cert_Alice))
    |   nil ]
    || empty
    || nil
    || nil
    || never
    *** Authentication never pattern for Client
    (:: R:FreshSet :: --- Cliente
    [nil |
          +(a ; AE),
          -(b ; e(exp(g, n(b, rb)), serverFin)),
          +(a ; e(AE, clientFin)),
          -(b ; e(exp(AE, n(b, rb)), sec(b, pskId))),
          +(a ; e(exp(AE, n(b, rb)), sec(b, pskId); cert_Alice)),
       nil ]
     & S:StrandSet || K:IntruderKnowledge)
 [nonexec] .
```

Cabe aclarar una funcionalidad que ofrece la especificación de ataques en Maude-NPA. Se puede observar un Strand bajo el comentario *never pattern*. La finalidad del Strand es acotar el espacio de búsqueda restringiendo a no buscar caminos en los que dicho Strand haya tenido lugar. Utilizando las fórmulas LTL del *model checking* y el operador *Until* se puede expresar cómo afecta

el *never pattern* a la búsqueda. Siendo *A* el Strand a buscar en el ataque y siendo *B* el Strand del *never pattern*, la fórmula LTL sería:

∃ ¬B U A

Cuyo significado es que existe un camino en el que B nunca ocurre hasta que A sea cierto.

Tras realizar una búsqueda de nueve niveles de profundidad, la herramienta encuentra una configuración inicial:

```
Maude> red summary(1,0) .
result Summary: States>> 1 Solutions>> 0
Maude> red summary(1,1) .
result Summary: States>> 4 Solutions>> 0
Maude> red summary(1,2) .
result Summary: States>> 9 Solutions>> 0
Maude> red summary(1,3) .
result Summary: States>> 17 Solutions>> 0
Maude> red summary(1,4) .
result Summary: States>> 28 Solutions>> 0
Maude> red summary(1,5) .
result Summary: States>> 44 Solutions>> 0
Maude> red summary(1,6) .
result Summary: States>> 62 Solutions>> 0
Maude> red summary(1,7) .
result Summary: States>> 75 Solutions>> 0
Maude> red summary(1,8) .
result Summary: States>> 80 Solutions>> 0
Maude> red summary(1,9) .
result Summary: States>> 72 Solutions>> 1
```

A continuación se muestra la ejecución que encuentra Maude-NPA:

```
+(a ; exp(g, n(a, #2:Fresh))),
-(a ; exp(g, n(a, #2:Fresh))),
+(b ; e(exp(g, n(b, #1:Fresh)), serverFin)),
-(b ; e(exp(g, n(b, #1:Fresh)), serverFin)),
+(e(exp(g, n(b, #1:Fresh)), serverFin)),
-(#0:Name),
-(e(exp(g, n(b, #1:Fresh)), serverFin)),
+(#0:Name ; e(exp(g, n(b, #1:Fresh)), serverFin)),
-(#0:Name ; e(exp(g, n(b, #1:Fresh)), serverFin)),
+(a ; e(exp(g, n(a, #2:Fresh)), clientFin)),
```

```
-(a ; e(exp(g, n(a, #2:Fresh)), clientFin)),
+(b ; e(exp(g, n(a, #2:Fresh) * n(b, #1:Fresh)), sec(b, #3:Fresh))),
-(b ; e(exp(g, n(a, #2:Fresh) * n(b, #1:Fresh)), sec(b, #3:Fresh))),
+(e(exp(g, n(a, #2:Fresh) * n(b, #1:Fresh)), sec(b, #3:Fresh))),
-(#0:Name),
-(e(exp(g, n(a, #2:Fresh) * n(b, #1:Fresh)), sec(b, #3:Fresh))),
+(#0:Name ; e(exp(g, n(a, #2:Fresh) * n(b, #1:Fresh)), sec(b,
#3:Fresh))),
-(#0:Name ; e(exp(g, n(a, #2:Fresh) * n(b, #1:Fresh)), sec(b,
#3:Fresh))),
+(a ; e(exp(g, n(a, #2:Fresh) * n(b, #1:Fresh)), sec(b, #3:Fresh) ;
sec(a, #4:Fresh))),
-(a ; e(exp(g, n(a, #2:Fresh) * n(b, #1:Fresh)), sec(b, #3:Fresh) ;
sec(a, #4:Fresh)))
```

Con motivo de facilitar la comprensión al lector, se propone en la Ilustración 9 la misma salida, pero plasmada en un diagrama de interacción con los tres participantes. Este ataque se corresponde con el ataque encontrado en la Ilustración 7, aunque más simplificado. En el flujo de mensajes del ataque, el término #0:Name representa una variable lógica asociada al nombre de un tercer participante, es decir, Charlie el intruso. Es indiferente el valor que tenga, sólo que represente al intruso.

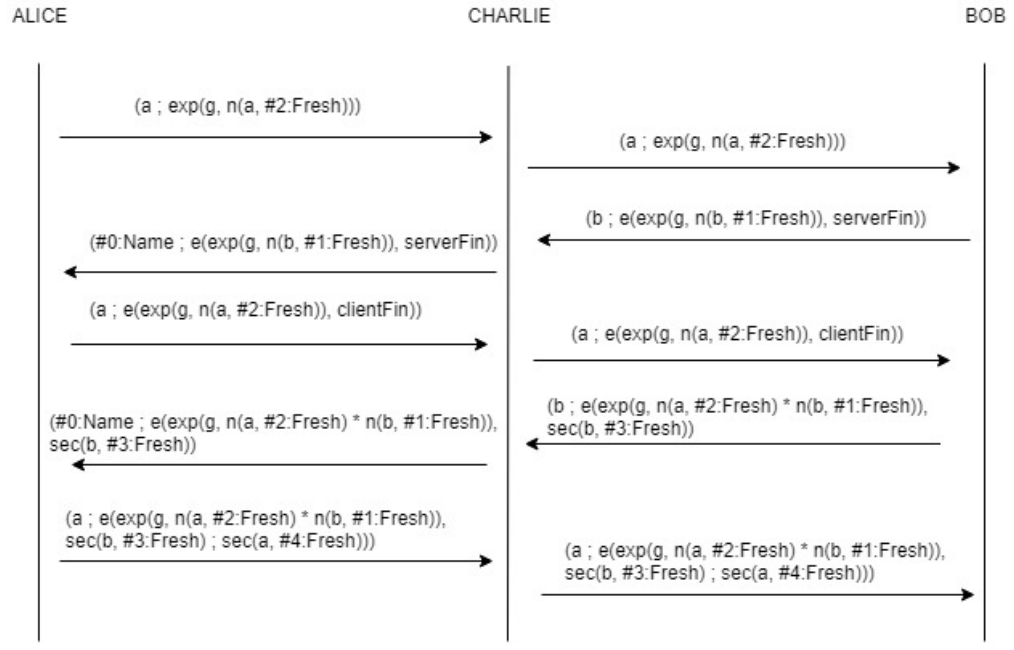

Ilustración 9. Ataque 1 como diagrama de interacción.

5.3.2 Autenticación del Cliente

Esta segunda propiedad es complementaria a la primera. El objetivo es encontrar una ejecución en la que el cliente haya terminado su ejecución y el intruso haya conseguido suplantar la identidad del Servidor frente al Cliente. Se pretende conseguir otro *Man-in-the-middle.* A continuación, se puede apreciar cómo se plasma el análisis en forma de ataque en Maude-NPA:

```
eq ATTACK-STATE(2) = --- MITM attack
    :: ra, CA :: --- Cliente
    [nil,
    +(a ; exp(g, n(a, ra))),
    -(b ; e(BE, serverFin)),
    +(a ; e(exp(g, n(a, ra)), clientFin)),
    -(b ; e(exp(BE, n(a, ra)), psk_id)),
    +(a ; e(exp(BE, n(a, ra)), psk_id ; sec(a, CA)))
     |  nil]
    || empty
    || nil
    || nil
    || never
    *** Authentication never pattern for Server
    (:: R:FreshSet :: --- Servidor
    [nil | -(a ; exp(g, n(a, ra))),
           +(b ; e(BE, serverFin)),
           -(a ; e(exp(g, n(a, ra)), clientFin)),
           +(b ; e(exp(BE, n(a, ra)), psk_id)),
      nil]
     & S:StrandSet || K:IntruderKnowledge)
 [nonexec] .
```

Al igual que en la propiedad anterior, tras una búsqueda en profundidad de nueve niveles de profundidad se encuentran dos configuraciones iniciales:

```
Maude> red summary(2,0) .
result Summary: States>> 1 Solutions>> 0
Maude> red summary(2,1) .
result Summary: States>> 2 Solutions>> 0
Maude> red summary(2,2) .
result Summary: States>> 6 Solutions>> 0
Maude> red summary(2,3) .
result Summary: States>> 12 Solutions>> 0
Maude> red summary(2,4) .
result Summary: States>> 22 Solutions>> 0
Maude> red summary(2,5) .
result Summary: States>> 35 Solutions>> 0
```

```
Maude> red summary(2,6) .
result Summary: States>> 53 Solutions>> 0
Maude> red summary(2,7) .
result Summary: States>> 77 Solutions>> 0
Maude> red summary(2,8) .
result Summary: States>> 128 Solutions>> 0
Maude> red summary(2,9) .
result Summary: States>> 198 Solutions>> 2
```

En esta ocasión la herramienta alcanza dos configuraciones iniciales. A continuación se muestra la primera de de las ejecuciones junto al mismo flujo de mensajes en un diagrama de interacción en la Ilustración 10:

```
+(a ; exp(g, n(a, #2:Fresh))),
-(a ; exp(g, n(a, #2:Fresh))),
+(#0:Name ; e(exp(g, n(#0:Name, #1:Fresh)), serverFin)),
-(#0:Name ; e(exp(g, n(#0:Name, #1:Fresh)), serverFin)),
+(e(exp(g, n(#0:Name, #1:Fresh)), serverFin)),
-(b),
-(e(exp(g, n(#0:Name, #1:Fresh)), serverFin)),
+(b ; e(exp(g, n(#0:Name, #1:Fresh)), serverFin)),
-(b ; e(exp(g, n(#0:Name, #1:Fresh)), serverFin)),
+(a ; e(exp(g, n(a, #2:Fresh)), clientFin)),
-(a ; e(exp(g, n(a, #2:Fresh)), clientFin)),
+(#0:Name ; e(exp(g, n(a, #2:Fresh) * n(#0:Name, #1:Fresh)),
sec(#0:Name, #3:Fresh))),
-(#0:Name ; e(exp(g, n(a, #2:Fresh) * n(#0:Name, #1:Fresh)),
sec(#0:Name, #3:Fresh))),
+(e(exp(g, n(a, #2:Fresh) * n(#0:Name, #1:Fresh)), sec(#0:Name,
#3:Fresh))),
-(b),
-(e(exp(g, n(a, #2:Fresh) * n(#0:Name, #1:Fresh)), sec(#0:Name,
#3:Fresh))),
+(b ; e(exp(g, n(a, #2:Fresh) * n(#0:Name, #1:Fresh)), sec(#0:Name,
#3:Fresh))),
-(b ; e(exp(g, n(a, #2:Fresh) * n(#0:Name, #1:Fresh)), sec(#0:Name,
#3:Fresh))),
+(a ; e(exp(g, n(a, #2:Fresh) * n(#0:Name, #1:Fresh)), sec(#0:Name,
#3:Fresh) ; sec(a, #4:Fresh)))
```

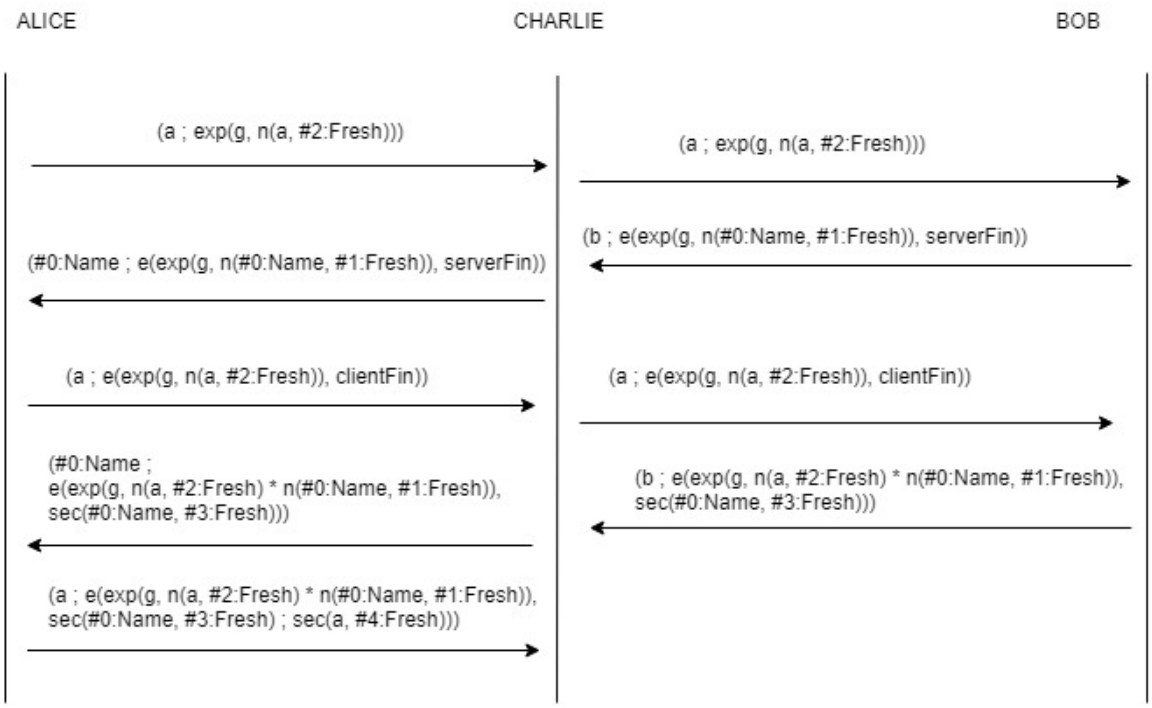

Ilustración 10. Primera configuración del ataque 2 en un diagrama de interacción.

A continuación, se expone la segunda de las ejecuciones para poder suplantar al Servidor junto al mismo flujo de mensajes, pero en un diagrama de interacción en la Ilustración 11:

```
+(a ; exp(g, n(a, #1:Fresh))),

-(a ; exp(g, n(a, #1:Fresh))),
+(exp(g, n(a, #1:Fresh))),
-(#0:Name),
-(exp(g, n(a, #1:Fresh))),
+(#0:Name ; exp(g, n(a, #1:Fresh))),
-(#0:Name ; exp(g, n(a, #1:Fresh))),
+(b ; e(exp(g, n(b, #3:Fresh)), serverFin)),
-(b ; e(exp(g, n(b, #3:Fresh)), serverFin)),
+(a ; e(exp(g, n(a, #1:Fresh)), clientFin)),
-(a ; e(exp(g, n(a, #1:Fresh)), clientFin)),
+(e(exp(g, n(a, #1:Fresh)), clientFin)),
-(#0:Name),
-(e(exp(g, n(a, #1:Fresh)), clientFin)),
+(#0:Name ; e(exp(g, n(a, #1:Fresh)), clientFin)),
-(#0:Name ; e(exp(g, n(a, #1:Fresh)), clientFin)),
+(b ; e(exp(g, n(a, #1:Fresh) * n(b, #3:Fresh)), sec(b, #4:Fresh))),
-(b ; e(exp(g, n(a, #1:Fresh) * n(b, #3:Fresh)), sec(b, #4:Fresh))),
+(a ; e(exp(g, n(a, #1:Fresh) * n(b, #3:Fresh)), sec(b, #4:Fresh) ;
sec(a, #2:Fresh)))
```

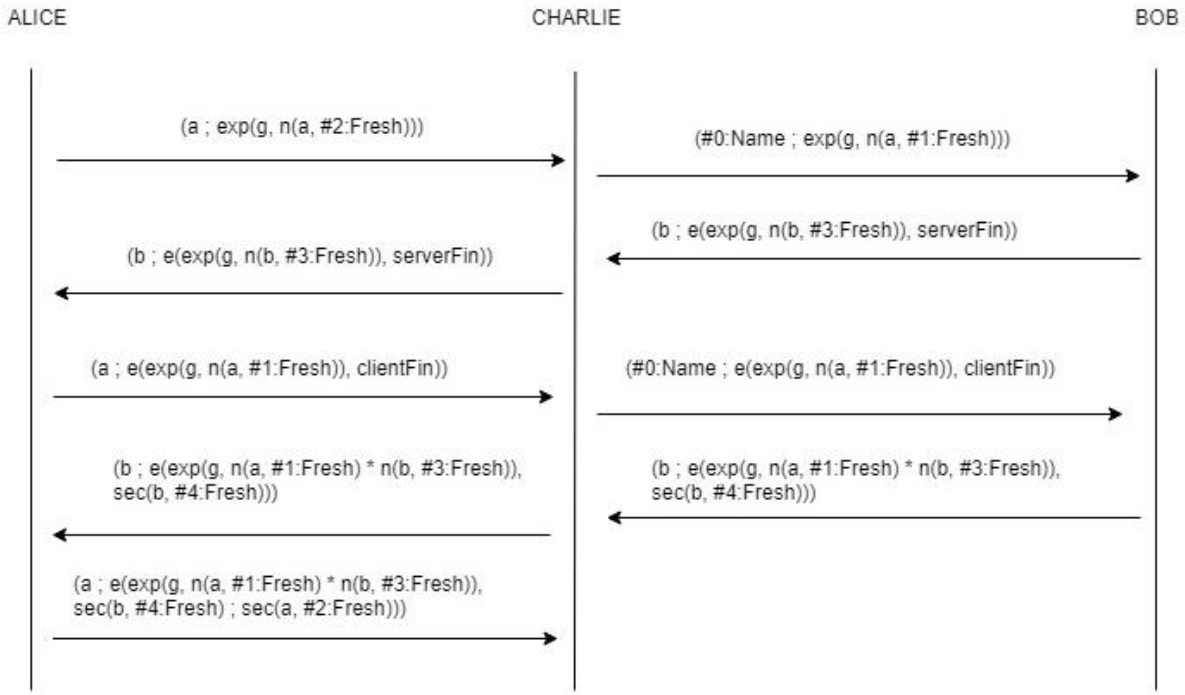

Ilustración 11. Segunda configuración del ataque 2 en un diagrama de interacción.

5.3.3 Secreto del Servidor

La tercera propiedad, acerca de secreto, busca encontrar una ejecución en la que el Servidor haya finalizado todo su flujo de mensajes y el intruso haya conseguido aprender el secreto generado por Servidor para el cliente. El objetivo de esta propiedad es demostrar que la transmisión de secretos, bajo el amparo de la criptografía, es secreta entre los participantes. A continuación, se puede apreciar cómo se plasma el análisis en forma de ataque en Maude-NPA:

```
eq ATTACK-STATE(3) =
    :: rb, pskId :: --- Servidor
    [nil,
           -(a ; AE),
           +(b ; e(exp(g, n(b, rb)), serverFin)),
           -(a ; e(AE, clientFin)),
           +(b ; e(exp(AE, n(b, rb)), sec(b, pskId)))
    |   nil ]
    || sec(b, psk_id) inI
```

```
    || nil
    || nil
    || nil
  [nonexec] .
```

Cabe comentar un pequeño detalle sobre el planteamiento de este ataque. Si se comparan los Strands a ejecutar por parte del Servidor con los Strands del protocolo completo del apartado 5.2.2 se puede observar la ausencia, en el ataque, de la última recepción del mensaje enviado por Alice. Se ha omitido esta última recepción debido a que el ataque puede tener lugar sin que el Servidor haya acabado su ejecución para poder encontrar la ejecución deseada. Además, en el momento de realizar la exploración por parte de la herramienta, el haber suprimido una recepción de mensaje esto supone una disminución muy grande del campo de búsqueda y por lo tanto una exploración más eficiente.

Tras una búsqueda en profundidad de 18 niveles, se cierra el espacio de búsqueda como síntoma de no poder encontrar una ejecución en la que se descubra el secreto generado por el Servidor:

```
Maude> red summary(3,0) .
result Summary: States>> 1 Solutions>> 0
Maude> red summary(3,1) .
result Summary: States>> 6 Solutions>> 0
Maude> red summary(3,2) .
result Summary: States>> 10 Solutions>> 0
Maude> red summary(3,3) .
result Summary: States>> 6 Solutions>> 0
Maude> red summary(3,4) .
result Summary: States>> 6 Solutions>> 0
Maude> red summary(3,5) .
result Summary: States>> 4 Solutions>> 0
Maude> red summary(3,6) .
result Summary: States>> 6 Solutions>> 0
Maude> red summary(3,7) .
result Summary: States>> 12 Solutions>> 0
Maude> red summary(3,8) .
result Summary: States>> 14 Solutions>> 0
Maude> red summary(3,9) .
result Summary: States>> 19 Solutions>> 0
Maude> red summary(3,10) .
result Summary: States>> 19 Solutions>> 0
Maude> red summary(3,11) .
result Summary: States>> 23 Solutions>> 0
```

```
Maude> red summary(3,12) .
result Summary: States>> 26 Solutions>> 0
Maude> red summary(3,13) .
result Summary: States>> 34 Solutions>> 0
Maude> red summary(3,14) .
result Summary: States>> 43 Solutions>> 0
Maude> red summary(3,15) .
result Summary: States>> 28 Solutions>> 0
Maude> red summary(3,16) .
result Summary: States>> 15 Solutions>> 0
Maude> red summary(3,17) .
result Summary: States>> 1 Solutions>> 0
Maude> red summary(3,18) .
result Summary: States>> 0 Solutions>> 0
```

5.3.4 Secreto del Cliente

La cuarta y última propiedad, es complementaria a la tercera. El objetivo de esta es buscar una ejecución en la que el intruso descubre el secreto generado por el Cliente para el Servidor. A continuación, se puede apreciar cómo se plasma el análisis en forma de ataque en Maude-NPA:

```
eq ATTACK-STATE(4) =
   :: ra, CA :: --- Cliente
   [nil,  +(a ; exp(g, n(a, ra))),
          -(b ; e(BE, serverFin)),
          +(a ; e(exp(g, n(a, ra)), clientFin)),
          -(b ; e(exp(BE, n(a, ra)), psk_id)),
          +(a ; e(exp(BE, n(a, ra)), psk_id ; sec(a, CA)))
    |  nil]
   || sec(a, certByAlice) inI
   || nil
   || nil
   || nil
[nonexec] .
```

Tras una búsqueda en profundidad de 15 niveles, se cierra el espacio de búsqueda como síntoma de no poder encontrar una ejecución en la que se descubra el secreto generado por el Cliente:

```
Maude> red summary(4,0) .
result Summary: States>> 1 Solutions>> 0
Maude> red summary(4,1) .
result Summary: States>> 6 Solutions>> 0
Maude> red summary(4,2) .
result Summary: States>> 10 Solutions>> 0
Maude> red summary(4,3) .
result Summary: States>> 4 Solutions>> 0
Maude> red summary(4,4) .
result Summary: States>> 6 Solutions>> 0
Maude> red summary(4,5) .
result Summary: States>> 17 Solutions>> 0
Maude> red summary(4,6) .
result Summary: States>> 19 Solutions>> 0
Maude> red summary(4,7) .
result Summary: States>> 16 Solutions>> 0
Maude> red summary(4,8) .
result Summary: States>> 15 Solutions>> 0
Maude> red summary(4,9) .
result Summary: States>> 18 Solutions>> 0
Maude> red summary(4,10) .
result Summary: States>> 20 Solutions>> 0
Maude> red summary(4,11) .
result Summary: States>> 20 Solutions>> 0
Maude> red summary(4,12) .
result Summary: States>> 14 Solutions>> 0
Maude> red summary(4,13) .
result Summary: States>> 7 Solutions>> 0
Maude> red summary(4,14) .
result Summary: States>> 2 Solutions>> 0
Maude> red summary(4,15) .
result Summary: States>> 0 Solutions>> 0
```

5.4 Conclusiones del análisis

Analizando los resultados devueltos por Maude-NPA se puede demostrar que, bajo el modelo codificado, se han verificado la seguridad propiedades de secreto, pero no las propiedades de autenticación pues se han encontrado ataques.

En lo referido a las propiedades de secreto, que la herramienta haya explorado todo el espacio de búsqueda significa que, de todas las posibles configuraciones inferidas desde los estados finales planteados, ninguna acaba en una configuración inicial. Es de esta manera cómo Maude-NPA informa al usuario que esa configuración no se puede dar porque, al igual que en el *model checking* estándar, puede haber un espacio de búsqueda de infinitos estados y es sobre este espacio en el que Maude-NPA asegura no haber encontrado ninguna configuración.

En lo referido a las propiedades de autenticación, Maude-NPA sí ha encontrado configuraciones iniciales desde las finales propuestas, tanto para Cliente como para Servidor. Esto implica una configuración en la que el intruso consigue alcanzar una configuración inicial y poder suplantar la identidad del Cliente y del Servidor.

Para finalizar, realizando una comparativa con el trabajo realizado en Tamarin (Cremers, Horvat, Scott, & Merwe, 2016) se encuentran las mismas conclusiones. Pues mediante el modelo en Tamarin se consiguen probar las propiedades de secreto y de autenticación, pero al incorporar la reanudación de sesión junto a PSK se revela un ataque *Man-in-the-middle.* Este trabajo, al incorporar al modelo la reanudación de sesión también ha podido encontrar el ataque.

6. Conclusiones y trabajo futuro

De este trabajo se puede extraer la conclusión de que es posible modelar el protocolo TLS 1.3, a alto nivel, en Maude-NPA. Ha sido complejo el hecho de entender el protocolo, estudiar el estado del arte en modelado formal de la versión del protocolo y en última instancia realizar en análisis en esta herramienta. Se considera un trabajo satisfactorio pues se han podido llegar a las mismas conclusiones que las extraídas en el trabajo realizado en la herramienta Tamarin.

Con respecto al trabajo a futuro, existen diversos caminos, aunque uno realmente interesante es poder modelar con mayor detalle la misma versión del protocolo y así poder estudiar si las conclusiones son las mismas con un modelo más complejo. Otra posibilidad interesante dentro del entorno de la ciberseguridad es poder modelar el protocolo WPA en su versión 3 el cual tiene, a fecha de 2019, una vulnerabilidad relacionada con la retrocompatibilidad con la versión anterior.

7. Bibliografía

Archive with TLS 1.3 Rev 10 models and property specificacions. (2015). Obtenido de http://tls13tamarin.github.io/.

Clavel, M., Durán, F., Eker, S., Escobar, S., Lincoln, P., Martí-Oliet, N., . . . Talcott, C. (2016). *Maude Manual (Version 2.7.1).*

Cremers, C., Dreier, J., & Sasse, R. (2016). *Manual de Tamarin Prover*. Obtenido de Tamarin Manual: https://tamarin-prover.github.io/manual/index.html

Cremers, C., Horvat, M., Scott, S., & Merwe, T. v. (2016). Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication. San Jose, CA, USA: IEEE.

D.Dolev, & A.C.Yao. (1981). On the security of public key protocols. *SFCS '81 Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, 350-357.

Duong, T., & Rizzo, J. (2011). *Here Comes the XOR ninjas.* Obtenido de https://nerdoholic.org/uploads/dergln/beast_part2/ssl_jun21.pdf

Escobar, S., Meadows, C., & Meseguer, J. (2017). *Maude-NPA, Version 3.1.*

Rescorla, E., & Mozilla. (Agosto de 2018). *Especificación TLS 1.3*. Obtenido de IETF-related tools: https://tools.ietf.org/html/rfc8446