UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Verificación automática de protocolos criptográficos de seguridad

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Torres Moliner, David

Tutor/a: Escobar Román, Santiago

CURSO ACADÉMICO: 2021/2022

Resumen

Las tarjetas de crédito, el método de pago mayormente usado alrededor del mundo, hacen uso del protocolo EMV para poder comunicar la transacción bancaria de una forma cómoda y segura. La comunicación entre la tarjeta hacia la terminal y la terminal hasta el banco tiene que ser segura para que no se produzca fraude o ningún tipo de error. Actualmente, con la evolución de la tecnología nuevas formas de pago han surgido y entre ellas el modo sin contacto permitiendo hacer uso de las tarjetas para pagar con un pequeño "toque" a la terminal.

El estudio *The EMV Standard: Break, Fix, Verify* estudia y analiza EMV creando un modelo, haciendo uso de la herramienta Tamarin, para poder comprobar la verificación y seguridad del protocolo de una forma automática. El estudio demuestra que el protocolo no es seguro al encontrar que varias de sus configuraciones se le pueden llevar a cabo distintos ataques como saltarse el PIN de seguridad y la verificación del propietario, cuando se está pagando por una gran cantidad de dinero, o ataques que permiten obtener servicios y productos de una forma completamente gratuita sin ningún tipo de cobro.

En el trabajo actual se ha entendido y modelado haciendo uso de la herramienta Maude-NPA el protocolo EMV para poder reproducir y verificar de forma automática varios ataques. De esta forma se consigue modelar dos ataques para unas configuraciones del modelo *contactless* al tratarse de una tecnología más reciente y por tanto, vulnerable. El trabajo corrobora los ataques descubiertos en el estudio mencionado anteriormente y afirma que el protocolo EMV no es seguro para ciertas configuraciones del modelo *contactless* para las tarjetas VISA.

Palabras clave: EMV, contactless, Maude-NPA, tarjetas de crédito, verificar, VISA.

Abstract

Credit cards, the most widely used payment method around the world, use the EMV protocol in order to communicate the banking transaction in a convenient and secure way. The communication between the card to the terminal and the terminal to the bank must be secure in order to avoid fraud or any kind of error. Currently, with the evolution of technology new ways of payment have emerged, including contactless mode, allowing cards to be used to pay with a small "touch" to the terminal.

The study The EMV Standard: Break, Fix, Verify studies and analyzes EMV by creating a model, using the Tamarin tool, to verify the verification and security of the protocol automatically. The study shows that the protocol is not secure by finding that several of its configurations can be attacked in different ways such as bypassing the security PIN and owner verification, when paying for a large amount of money, or attacks that allow services and products to be obtained completely free of charge.

In the current work, the EMV protocol has been understood and modeled using the Maude-NPA tool in order to automatically reproduce and verify several attacks. In this way two attacks have been modeled for some configurations of the contactless model as it is a more recent technology and consequently, vulnerable. The work corroborates the attacks discovered in the aforementioned study and affirms that the EMV protocol is not secure for certain configurations of the contactless model for VISA cards.

Keywords: EMV, contactless, Maude-NPA, credit cards, verify, VISA.

Índice

Índice de Ilustraciones

1. Introducción

Actualmente, las tarjetas de crédito [1] son el sistema de pago mayormente utilizado y que prácticamente ha dejado en segundo plano al dinero en efectivo. Las tarjetas de crédito han sido y son utilizadas diariamente por la gran mayoría de personas para efectuar pagos por servicios o productos en miles de empresas alrededor del mundo, tanto en formato físico como a través de Internet.

Los bancos ofrecen a sus clientes este método de pago al tratarse de un método fiable, cómodo y seguro. Así, de este modo, tan solo se necesita asociar una cuenta bancaria o depositar fondos en la tarjeta para que el propietario no tenga que preocuparse por no llevar más dinero efectivo en la cartera. Además, con la evolución de la tecnología, las tarjetas de crédito añaden nuevas características como, por ejemplo, el modo de pago sin contacto (*contactless)*, haciendo uso de tecnología como Near Field Communications (NFC) [2].

El trabajo actual, trabajo de fin de grado (TFG), Verificación automática de protocolos criptográficos de seguridad, estudia el protocolo que efectúan las tarjetas de crédito para comunicarse entre los distintos agentes que participan. El nombre del protocolo que hacen uso las tarjetas más utilizadas se nombra Europay MasterCard VISA (EMV) [3].

Con el objetivo de analizar los distintos modelos y vulnerabilidades que existen en el protocolo EMV el alumno se ha basado en el análisis del estudio *The EMV Standard: Break, Fix, Verify* realizado por David Basin, Ralf Sasse, y Jorge Toro-Pozo [4]. A lo largo del estudio se analizaron los distintos fallos de seguridad que tiene el protocolo, incluyendo fallos que afectan a las tarjetas VISA.

El TFG hace uso de una herramienta automática similar a Tamarin, la herramienta usada por el estudio mencionado, para la verificación y análisis de la seguridad del protocolo EMV. De esta forma, la idea principal es poder corroborar los descubrimientos del estudio y confirmar la existencia de dichos fallos de seguridad. Maude-NPA [5] será la herramienta a utilizar, desarrollada por Santiago Escobar, Catherine Meadows y José Meseguer, permitiendo modelar el protocolo EMV con el fin de buscar patrones de ataque que permitan verificar la seguridad del protocolo.

A continuación se explicarán las distintas fuentes de motivación y objetivos del alumno, respecto al trabajo, finalizando con la estructura que se ha decidido seguir.

1.1. Motivación

El TFG supone un significativo avance en relación con el grado universitario del alumno al tratarse de su primer trabajo propio de investigación creando o desarrollando algo completamente nuevo y único.

La Universidad Politécnica de Valencia (UPV) propone cada año, mediante la oferta pública, un listado con diversos temas y materias a escoger, incluyendo temas de seguridad

informática. El alumno decidió escoger el trabajo actual ya que su principal búsqueda y motivación es hacer un proyecto relacionado con la ciberseguridad.

Maude-NPA [5] es la herramienta usada para analizar y comprobar la seguridad en protocolos criptográficos, el tema principal del TFG. Por lo que cuando se le asignó el TFG al alumno conjuntamente a su tutor, se buscó analizar y entender un protocolo que fuese interesante, vistoso, actual y que tuviera brechas de seguridad.

Al final se llegó a la conclusión de que con el uso de la herramienta Maude-NPA, una herramienta nunca utilizada por el alumno y lo que aumentaba su motivación e interés por aprenderla, podían replicar el estudio [4] sobre las tarjetas VISA y su protocolo EMV. La principal motivación por parte del alumno ha sido comprender un estudio de una alta reputación, permitiendo entender cada configuración que ofrece EMV y poder replicar lo que según ellos era posible de ejecutar, afirmando que las tarjetas VISA no eran seguras.

La motivación personal del alumno a la hora de escoger un tema para el TFG siempre había sido la ciberseguridad, pues se trata de un trabajo donde hay que ir un paso más allá de lo que se había pensado para un servicio, protocolo, o uso de una herramienta. Por esta razón, un protocolo como el de las tarjetas de crédito era un gran logro de entender cada una de sus fases, características y funcionamiento. Al mismo tiempo, comprender las vulnerabilidades ya descubiertas o encontrar nuevas siempre han sido grandes objetivos del alumno.

1.2. Objetivos

En este apartado se exponen los distintos objetivos que se pretenden conseguir por parte del alumno y los cuales han tenido que ser superados encontrando distintas soluciones y resolviendo varios problemas a través de determinados procesos. Los objetivos planteados han sido los siguientes:

1. Comprender el estudio realizado en [4] sobre EMV.
2. Estudiar y analizar el funcionamiento de los distintos modelos y configuraciones del protocolo EMV.
3. Entender la herramienta Maude-NPA [5] y cómo analiza protocolos de seguridad automáticamente.
4. Modelar el protocolo EMV haciendo uso de la herramienta Maude-NPA.
5. Investigar las distintas vulnerabilidades y ataques que existen en el protocolo EMV.
6. Recrear alguno de los ataques del *paper* [4] mediante Maude-NPA.
7. Extraer resultados y analizarlos con Maude-NPA.
8. Comparar los resultados obtenidos con Maude-NPA y los del trabajo [4].

Durante la preparación inicial, antes de empezar con el trabajo, el alumno tuvo que utilizar varias semanas para poder entender y comprender los objetivos uno y tres. Esto se debe a que Maude-NPA es una herramienta no usada antes por el alumno, teniendo que comprender su sintaxis y uso desde cero y además, el estudio de una importante reputación sobre el protocolo EMV que tampoco se había visto con anterioridad. Por esta razón, ambos objetivos se pueden clasificar con una dificultad alta.

Una vez obtenidos los objetivos uno y tres, el alumno buscó añadir conocimiento sobre el protocolo EMV con más información, y no tan solo con lo que el estudio [4] exponía. Para ello, se propuso el objetivo número dos y cinco, con una duración menor a la de los objetivos anteriores, pero con una inversión de tiempo esencial. Ambos objetivos son necesarios debido a que sin el conocimiento adicional extraído de la documentación del protocolo o la investigación independiente de las vulnerabilidades que existen, el trabajo no habría sido capaz de ser finalizado por el alumno. Por esta razón el grado de dificultad es medio.

Los principales objetivos propuestos por el alumno, número cuatro y seis, son los más extensos y complicados de todos ellos, debido a que es donde verdaderamente el trabajo realizado y conocimiento adquirido por el alumno se va a ver reflejado. La duración se remonta a meses, ya que continuamente se han estado aplicando cambios a la especificación del protocolo EMV con Maude-NPA para poder hacerlo lo más cercano a la realidad. También, los diferentes ataques extraídos de [4] que se ha llevado a cabo su modelado para su realización, necesitaban horas diariamente de ejecución para finalmente poder ver reflejado una traza de mensajes correcta con una solución válida. Por todas estas razones, el grado de dificultad es el más alto.

Finalmente, los últimos objetivos número siete y ocho, son los más rápidos de obtener necesitando tan solo un par de días. La razón por la que son los más sencillos es debido a que solo se tenía que analizar los resultados obtenidos de todos los objetivos anteriores para ver si coincidían con los resultados esperados del previo estudio realizado. El grado de dificultad de estos objetivos es el más bajo.

1.3. Estructura

En cuanto a la estructura del trabajo se han planteado organizar el TFG en ocho secciones:

Sección 1: expone el tema del trabajo de una forma clara y concisa con sus respectivas motivaciones y objetivos que el alumno espera conseguir.

Sección 2: una pequeña investigación bibliográfica sobre trabajos previos y actuales que tengan el mismo tema relacionado con el actual TFG.

Sección 3: un estudio realizado sobre la herramienta Maude-NPA que se va a usar explicando los distintos módulos, sintaxis y propiedades que se pueden definir a la hora de modelar un protocolo. Del mismo modo, los distintos comandos de ejecución de la herramienta.

Sección 4: la explicación y análisis del protocolo EMV con sus distintas fases, modelos y ataques, terminando con una conclusión del protocolo de los problemas de seguridad encontrados en [4].

Sección 5: después de la investigación de Maude-NPA y EMV en esta sección se han usado todos los conocimientos adquiridos para poder replicar el comportamiento del protocolo y verificar su seguridad. Asimismo, los distintos resultados se comparan con el estudio realizado en la cuarta sección.

Sección 6: finalizando el TFG se presentan las conclusiones del trabajo, cómo se han conseguido todos los objetivos marcados y si se han desarrollado satisfactoriamente. Además, se proponen en esta sección trabajos futuros para el perfeccionamiento del proyecto actual.

Sección 7 y Sección 8: se expone una lista con todos los recursos provenientes de distintas fuentes de información que se han usado a lo largo del trabajo para la adquisición del conocimiento necesario. Conjuntamente se sitúan los anexos, con información extra pero necesaria, por ejemplo, el código, resultados y los Objetivos de Desarrollo Sostenible (ODS).

2. Estado del Arte

En la actualidad son muchos los trabajos relacionados con la ciberseguridad, debido a que es un tema en constante evolución, y además de todas las distintas vulnerabilidades que surgen prácticamente a diario. Por esta razón, los sistemas informáticos necesitan ser actualizados con parches de seguridad continuamente. A pesar de esto, lamentablemente, siempre surgen nuevos fallos de seguridad que en el futuro tendrán que ser arreglados del mismo modo. Las aplicaciones, sistemas operativos e incluso protocolos criptográficos tienen dichos fallos de seguridad y, entre ellos, está incluido el protocolo de las tarjetas de crédito EMV.

El protocolo EMV se trata de un protocolo crítico al estar relacionado directamente con el dinero de las personas como método de pago. Por esta razón, existen una gran cantidad de trabajos previos al actual sobre el protocolo EMV y sus distintas vulnerabilidades. Principalmente, el trabajo previo con mayor relevancia se trata de *The EMV Standard: Break, Fix, Verify,* realizado por David Basin, Ralf Sasse, y Jorge Toro-Pozo [4] debido a que es el trabajo del cual el proyecto actual obtiene la inspiración y la base para poder comprobar y verificar los mismos fallos que ellos han obtenido. El trabajo es realizado a través de la herramienta para la verificación de protocolos de forma automática: Tamarin. El modelado corresponde tanto al modo *contact* como al *contactless*, incluyendo cada una de las posibles combinaciones de configuraciones que existen en el protocolo EMV. Los autores han conseguido demostrar, después de la ejecución de su modelo, que efectivamente el protocolo no es seguro, tanto para el modelo de contacto como el de sin. Finalmente, concluyen con nuevos ataques descubiertos del protocolo y confirmando, gracias a su modelado, otros que fueron demostrados previamente como *Chip and PIN is Broken* realizado por Steven J. Murdoch, Saar Drimer, Ross Anderson y Mike Bond [6], donde para una específica configuración se puede saltar el PIN de las tarjetas de crédito. Además, han creado una prueba de concepto haciendo uso de un dispositivo móvil Android para poder reproducirlo en la vida real usando una técnica llamada ataque *relay*, lo que les permite replicar la comunicación entre la tarjeta y la terminal.

Del mismo modo, otro trabajo publicado recientemente es *Practical EMV Relay Protection*, desarrollado por Andreea-Ina Radu, Tom Chothia, Christopher J.P. Newton, Ioana Boureanu y Liqun Chen [7] también haciendo uso de la herramienta Tamarin para poder demostrar una importante vulnerabilidad en el sistema de pago Apple Pay a través de un iPhone haciendo uso de la técnica de ataque *relay* con una tarjeta VISA en modo tránsito. Este ataque permite, aún teniendo el iPhone bloqueado con el FaceID, huella dactilar o número secreto, usar la tarjeta que se tenga guardada internamente en el teléfono sin necesidad de desbloquearlo. Asimismo, no existen restricciones de dinero pudiendo utilizar una cantidad de dinero ilimitada.

Por otro lado, aparte de los trabajos sobre algunas especificaciones para ciertos ataques sobre el protocolo, también existen trabajos teóricos como *EMV Cards Vulnerabilities Detection Using Deterministic Finite Automaton* realizado por Tarik Hajjia, Noura Ouerdib, Abdelmalek Azizib, Mostafa Azizic [8]. Donde buscan demostrar teóricamente y haciendo uso de autómatas finitos deterministas que EMV es vulnerable ante distintas posibles intrusiones durante el protocolo.

Por otra parte, la herramienta que se va a usar a lo largo de este trabajo es Maude-NPA, e igualmente existen estudios que hacen uso de esta herramienta para el modelado de protocolos

como *Verificación automática del protocolo DP-3T asociado a las aplicaciones COVID-19* de Daniel Martínez Martín [9] para modelar un protocolo relacionado con el COVID-19, en específico el DP-3T. También, existen trabajos que ponen a prueba distintos protocolos criptográficos usando Maude-NPA como en *Verificación automática de protocolos criptográficos de seguridad* de Nataniel Renzo Rondo Van [10]. Asimismo, como se ha podido observar muchos son los trabajos de EMV que hacen uso de Tamarin como herramienta de modelado, pero existen a su vez trabajos como *Verificación automática del protocolo TLS 1.3 usando Maude-NPA* de José Lluch Palop [11], que han sido capaces de transformar la especificación del protocolo TLS hecho en Tamarin a un modelo que funcione en la herramienta Maude-NPA y replicar alguno de los ataques.

3. Maude-NPA

Maude-NRL Protocol Analyzer (Maude-NPA) [5] es una herramienta desarrollada por Santiago Escobar, Catherine Meadows y José Meseguer en 2017, basada en su predecesora NRL Protocol Analyzer (NPA), la cual usaba la unificación y la búsqueda hacia atrás desde un estado final con el objetivo de determinar si dicho estado era alcanzable o no. Principalmente, lo que hace diferente a Maude-NPA es el análisis de protocolos criptográficos mediante el uso del lenguaje de programación Maude, más información en [12]. Además, consta de una diversa cantidad de nuevas propiedades algebraicas no vistas en otras herramientas permitiendo la verificación de diversos protocolos.

NPA tiene una gran base teórica basada en la reducción y la reescritura a nivel lógico, sin embargo, solo puede usarse para un cierto número limitado de reglas. No obstante, gracias a Maude-NPA se ha podido ampliar este rango con nuevas ecuaciones teóricas, puesto que poseen el soporte a teorías que hacen uso de símbolos permitiendo la asociatividad, la conmutatividad y axiomas de identidad. Por consiguiente, se puede obtener teorías enfocadas a la seguridad como la cancelación de cifrado y descifrado o el XOR.

En el trabajo que se va a describir a continuación se usa la herramienta Maude-NPA para modelar de una forma específica el protocolo EMV [3], con el objetivo de analizar y verificar si se cumplen las propiedades de seguridad. El objetivo se obtiene mediante el alcance de ciertos estados iniciales con el uso de la técnica de búsqueda hacia atrás, que hace uso Maude-NPA.

3.1. Modelado Maude-NPA

El modelado de un protocolo con la herramienta Maude-NPA se obtiene describiendo las especificaciones del comportamiento necesarias haciendo uso de un fichero maude, Ilustración 1, compuesto por tres secciones claramente separadas y definidas. Estas secciones se hacen llamar módulos.

```
fmod PROTOCOL-EXAMPLE-SYMBOLS is
--- Importing sorts Msg, Fresh, and Public
protecting DEFINITION-PROTOCOL-RULES .
------------------------------------------------------------
--- Overwrite this module with the syntax of your protocol
--- Notes:
--- * Sorts Msg and Fresh are special and imported
--- * New sorts can be subsorts of Msg
--- * No sort can be a supersort of Msg
--- * Variables of sort Fresh denote uniquely generated data
--- * Sorts Msg and Public cannot be empty
------------------------------------------------------------
endfm

fmod PROTOCOL-EXAMPLE-ALGEBRAIC is
protecting PROTOCOL-EXAMPLE-SYMBOLS .
------------------------------------------------------------------
--- Overwrite this module with the algebraic properties of your protocol
--- * Use only variant equations eq Lhs = Rhs [variant] .)
--- * Or use equations for dedicated unification algorithm
--- * Attribute owise cannot be used
------------------------------------------------------------------
endfm

fmod PROTOCOL-SPECIFICATION is
protecting PROTOCOL-EXAMPLE-SYMBOLS .
protecting DEFINITION-PROTOCOL-RULES .
protecting DEFINITION-CONSTRAINTS-INPUT .
------------------------------------------------------------
--- Overwrite this module with the strands of your protocol and the attack states
------------------------------------------------------------
eq STRANDS-DOLEVYAO
= --- Add Dolev-Yao strands here. Strands are properly renamed
[nonexec] .
eq STRANDS-PROTOCOL
= --- Add protocol strands here. Strands are properly renamed
[nonexec] .
eq ATTACK-STATE(0)
= --- Add attack state here
--- More than one attack state can be specified, but each must be identified by a
natural number
[nonexec] .
endfm

--- THIS HAS TO BE THE LAST ACTION !!!!
select MAUDE-NPA .
```

Ilustración 1. Maude-NPA fichero de ejemplo. Extraído de [5].

El primer módulo, nombrado PROTOCOL-EXAMPLE-SYMBOLS, consiste en la sintaxis del protocolo especificado indicando todos los tipos y operadores del cual se compone.

El segundo módulo, PROTOCOL-EXAMPLE-ALGEBRAIC, está compuesto por las distintas propiedades algebraicas de los operadores necesarios que se han descrito en el módulo anterior.

Por último, el tercer módulo, PROTOCOL-SPECIFICATION, corresponde a la descripción principal del comportamiento que tiene el protocolo a través del uso de los procesos algebraicos o mediante la forma más común y la cual se va a usar: los *strands*, los cuales se ven en la Sección 3.1.3. En este último módulo también se incluyen las distintas habilidades del intruso descritas en la Sección 3.1.4. y finalmente los *attack-states*, los cuales describen el comportamiento del atacante y qué patrones se quiere comprobar si pueden cumplirse, del mismo modo se ven en la Sección 3.1.3.

3.1.1. Sintaxis del protocolo

La sintaxis en Maude-NPA del protocolo se encuentra definida en el primer módulo y está compuesta por los diferentes tipos que van a ser usados y sus distintos operadores.

Por una parte, a los diferentes tipos se les llaman *sorts*. El objetivo de los *sorts* es poder crear distintos tipos que puedan representar distinta información que se haga uso en el protocolo, primera línea de la Ilustración 2. Los *sorts*, asimismo, pueden tener a su vez subtipos a los cuales se le llama *subsort*, permitiendo una especificación más profunda de un determinado tipo de dato, segunda y tercera línea, Ilustración 2.

```
--- Sorts y Subsorts
sorts Name Nonce Nat.
subsort Name Nonce Nat < Msg .
subsort Name < Public .
```

Ilustración 2. Sort y subsort.

Por defecto, Maude-NPA tiene tres tipos predefinidos para sus distintos usos y con sus respectivas restricciones: *Msg*, *Public* y *Fresh*.

Msg: es el tipo del cual se crean el resto de tipos. Todos los tipos creados por el usuario son un subtipo por definición de Msg. Además, es un tipo que no puede estar vacío y necesariamente tiene que haber como mínimo un símbolo tipo o subtipo Msg.

Public: es el tipo para identificar los datos que se asume que el intruso sabe durante el transcurso del protocolo. Por definición, es un subtipo de Msg y tampoco puede estar vacío.

Fresh: es un tipo especial que es útil cuando se quiere indicar que tiene que ser un valor único. Normalmente se usa para nonce, valores los cuales no se van a repetir. Este tipo solo puede estar asociado a variables.

Por otra parte, en el actual módulo, también son declarados los operadores, símbolos declarados por el usuario para sus distintos usos. Respecto a su sintaxis, los operadores pueden ser de tres tipos: *prefix*, *infix* y *mix-fix*.

Prefix: es la sintaxis estándar y se escoge un símbolo cualquiera para definir el operador. En la Ilustración 3 se ha escogido el símbolo *d* el cual no recibe argumentos y devuelve un tipo Name y *nombre* que recibe un argumento Name y devuelve otro tipo Name.

```
--- Operaciones prefix
op d : -> Name .  --- David
op nombre : Name -> Name .
```

Ilustración 3. Operador prefix.

Infix y Mix-fix: utilizan distintos símbolos como la suma (+) y la barra baja (_) con la intención de representar cualquier argumento en esa específica posición. En la Ilustración 4 se muestra el operador que representa la suma con la posición de los argumentos en las barras bajas y el símbolo de la suma como operador. Los dos argumentos son de tipo Nat y devuelve un tipo Nat.

```
--- Operacion infix
op _+_ : Nat Nat -> Nat .
```

Ilustración 4. Operador infix.

Además, hay que mencionar que una sintaxis como la de *infix* se usa para declarar axiomas, que se ven en la Sección 3.1.2.

En la siguiente Ilustración 5 se puede observar los distintos atributos que tienen las operaciones como *frozen*, necesario para indicar a Maude que no reescriba los argumentos de los símbolos usados o *gather*, para indicar cómo separar los mensajes al usar la concatenación.

```
--- Atributos operaciones
op n : Name Fresh -> Nonce [frozen] .
op _;_ : Msg Msg -> Msg [gather (e E) frozen] .
```

Ilustración 5. Atributos operaciones.

3.1.2. Propiedades Algebraicas del protocolo

Las propiedades algebraicas en Maude-NPA definidas en el segundo módulo realizan el análisis necesario simbólico para conseguir expresar las distintas funciones criptográficas del propio protocolo.

En Maude-NPA existen tres tipos de propiedades algebraicas que están disponibles para su uso: *equational attributes*, *variant equations* y *metadata equations*.

Equational attributes: además conocidos como axiomas, permiten combinar el uso de cualquier símbolo declarado en los operadores con la asociatividad, conmutación e identidad. Esta propiedad es la única que se utiliza en el módulo anterior ya que se usa en los atributos de los operadores.

Variant equations: también conocida como ecuaciones. Son la especificación teórica que se necesita para el correcto comportamiento de ciertos operadores normalmente criptográficos.

Metadata equations: se utilizan para elegir las ecuaciones a los algoritmos que las necesitan. Su propósito es útil cuando, por ejemplo, la teoría no permite la combinación de axiomas y ecuaciones o, cuando se quiere mejorar la eficiencia de las teorías ya conocidas.

Un ejemplo, se muestra en la Ilustración 6, mediante el uso de ecuaciones y axiomas usando el símbolo asterisco (*) para indicar la operación XOR.

```
--- Axiomas
op _*_ : Msg Msg -> Msg [frozen assoc comm] .
op null : -> Msg .

--- Ecuaciones
eq X:Msg * X:Msg * Y:Msg = Y:Msg [variant] .
eq X:Msg * X:Msg = null [variant] .
eq X:Msg * null = X:Msg [variant] .
```

Ilustración 6. Operadores y Ecuaciones de XOR.

3.1.3. Strands del protocolo y Attack-Patterns

El último y tercer módulo del fichero se describe el principal comportamiento del protocolo en sí, mediante el uso de los *strands*, las distintas habilidades del intruso, reglas Dolev-Yao, variables y finalmente un estado el cual se quiere comprobar mediante patrones si el intruso es capaz de alcanzar, conocido como *attack-state*.

Un *strand* es tan simple como una secuencia de envíos y recibos de mensajes. A través de distintos *strands* se puede implementar al completo el comportamiento de un protocolo.

```
:: r1, . . . , rj :: [ m1 ±, . . . , mi ± | mi+1 ±, . . . , mk ± ]
```

Ilustración 7. Strand genérico.

En la Ilustración 7 se pueden apreciar símbolos como una secuencia de símbolos *m*, separado por comas correspondiente con el envío y recibimiento de mensajes. Cada mensaje puede estar con un + o un -, significando un envío de un mensaje (+) o el recibimiento de otro (-). Al mismo tiempo, se puede apreciar el símbolo *r*, de tipo Fresh, indicando una variable única del *strand*. Finalmente, se observa una barra vertical (|) la cual separa los mensajes que ya han sucedido y están en el pasado de los que están por suceder situados en el futuro.

La ecuación, STRANDS-PROTOCOL que describe el protocolo, puede componerse de tantos *strands* como sean necesarios para representar el flujo de envíos y recibimientos de los mensajes a través de los agentes del protocolo. Usualmente, todos los *strands* comienzan y acaban con un mensaje *nil*, y por convenio, se ha decidido que la barra vertical se sitúe al principio del *strand* justo después del primer mensaje *nil* para que todos los *strands* puedan comenzar en el mismo sitio.

A continuación, en las Ilustraciones 8, 9 y 10, se muestra a modo de ejemplo, el protocolo que se muestra en [5], Needham-Schroeder-Lowe con XOR, su representación informal, diagrama y sus *strands* respectivamente, haciendo uso de la terminología de Alice (A) y Bob (B).

```
1. A → B : pk(B, NA; A)
2. B → A : pk(A, NA; NB ∗ B)
3. A → B : pk(B, NB)
```

Ilustración 8. Needham-Schroeder-Lowe con XOR protocolo. Extraído de [5].

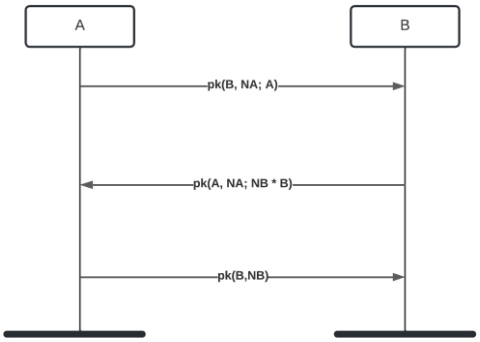

Ilustración 9. Needham-Schroeder-Lowe con XOR diagrama.

```
:: r ::
[ nil | +(pk(B, n(A,r) ; A)), -(pk(A, n(A,r) ; B * YN)), +(pk(B, YN)), nil]
:: r' ::
[ nil | -(pk(B, XN ; A)), +(pk(A, XN ; B * n(B,r'))), -(pk(B,n(B,r'))), nil]
```

Ilustración 10. Needham-Schroeder-Lowe con XOR strands. Extraído de [5].

Por otra parte, dentro del módulo actual y también del anterior se encuentran las variables, las cuales se pueden declarar de varias formas en cuanto al alcance dentro del módulo. La declaración puede ser a nivel global o local.

```
--- Variables
var X : Msg .
sk(D:Name, X) .
vars X Y Z : Msg  .
```

Ilustración 11. Variables.

Si se observa la Ilustración 11, el primer ejemplo corresponde con una declaración de una variable global al módulo; el segundo ejemplo se observa dentro de una operación de forma local y, finalmente, el tercero deja ver la declaración de varias variables del mismo tipo.

La última parte del módulo corresponde con el *attack-state*, el cual está compuesto por distintas secciones, las cuales se desean replicar o usar con el objetivo de confirmar si, mediante las habilidades del intruso y los *strands*, se puede llegar a alcanzar dicho patrón mediante una ejecución hacia atrás. El módulo no tiene que estar restringido a un único patrón, sino que pueden existir varios.

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

Ilustración 12. Attack-state protocolo Needham-Schroeder-Lowe con XOR. Extraído de [5].

A diferencia de las ecuaciones con los *strands*, los cuales corresponden con los agentes legítimos que interactúan dentro del protocolo, los *attack-state* situados en las ecuaciones ATTACK-STATE tienen que estar numeradas, cada una con un número natural, para su posterior identificación y ejecución.

Los *attack-state* están compuestos por cinco secciones como se puede ver en la Ilustración 12, las cuales están separadas por dos barras verticales (||). La primera es un *strand*, *attack-pattern*, el cual corresponde con el patrón que se quiere y se espera cumplir a lo largo de la ejecución. Por ello, la barra vertical está situada antes del último *nil* y no al principio como sucedía en los *strands* del protocolo. Mientras que la segunda, es el conocimiento que el intruso se espera que aprenda al acabar la ejecución del protocolo. Las otras tres secciones se suelen poner a *nil* ya que no son usadas de forma estándar.

3.1.4. Reglas Dolev-Yao

Las reglas Dolev-Yao están situadas en el tercer módulo del fichero maude. Estas reglas están dentro de una ecuación, STRANDS-DOLEVYAO, la cual se compone de distintos *strands* separados por el símbolo &. Las reglas consisten en una descripción mediante envíos y recibos de mensajes cuyo objetivo es definir las distintas habilidades que tiene disponible y pueda hacer uso el intruso. Además, normalmente se tiene que dar por hecho que toda operación descrita en el protocolo el intruso debería ser capaz de replicarla y utilizarla.

Las habilidades son posibles de utilizar debido a que el intruso está compartiendo un canal con los distintos agentes del protocolo, por lo que puede observar todos los mensajes que se hayan estado enviando entre sí. El comportamiento del intruso empieza quedándose a la espera de un mensaje para posteriormente enviar mediante sus habilidades los mensajes que le interesen (reenviando o creando nuevos) con el objetivo de cumplir con el patrón de ataque.

```
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 | -(X), -(Y), +(X * Y), nil ] &
:: r :: [ nil | +(n(i,r)), nil ] &
:: nil :: [ nil | +(null), nil ]
[nonexec] .
```

Ilustración 13. Habilidades del Intruso. Extraído de [5].

Por ejemplo, habilidades vistas en la Ilustración 13, como la concatenación son muy útiles para extraer mensajes, separando las dos partes concatenadas. Además, poder generar mensajes cifrados con una clave privada e incluso generar mensajes cifrados con XOR.

3.2. Comandos y Ejecución

Maude-NPA se trata de un programa con una lista de diferentes comandos los cuales se pueden usar de varias maneras. Los comandos más utilizados son los siguientes: *load*, *red*, *run*, *digest*, *summary* e *initials*.

Load: sirve para cargar Maude-NPA a Maude, el cual incluye todas las características y comandos necesarios. Además, este comando sirve para cargar todos los ficheros maude con las especificaciones de los protocolos.

```
Maude> load maude-npa
Maude> load ejemplo.maude
```

Ilustración 14. Comando load.

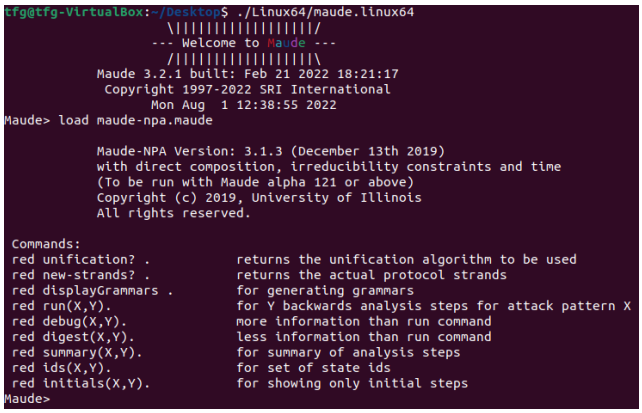

Ilustración 15. Comando load Needham-Schroeder-Lowe with XOR.

Red: se puede combinar con el resto de los comandos y se usa poniendo la palabra *red* justo antes de comando. Se utiliza para reducir la información que se muestra por salida estándar.

Run: es el comando el cual ejecuta el fichero que se le haya pasado intentando buscar hacia atrás con el fin de buscar el estado inicial. Con el objetivo de limitar la búsqueda y que no sea infinita, se puede asignar una profundidad n y un *attack-state* m. Al final de la ejecución muestra todas las soluciones por salida estándar de las hojas que genere el árbol de búsqueda, con sus *strands*, el conocimiento del intruso y el intercambio de mensajes.

```
Maude> red run(m,n) .
```

Ilustración 16. Comando run.

Digest: el comando funciona exactamente igual a *run*, pero en vez de imprimir por pantalla toda la información que muestra *run* solo muestra el intercambio de mensajes.

```
Maude> red digest(m,n) .
```

Ilustración 17. Comando digest.

Summary: la ejecución de este comando muestra para un *attack-state* m y una profundidad n, cuántos estados se han generado durante la búsqueda y cuántos de ellos son solución respecto al *attack-state*.

```
Maude> red summary(m,n) .
```

Ilustración 18. Comando summary.

Initials: la ejecución funciona de la misma forma que el *run* pero solo mostrando los estados iniciales y no las hojas del árbol de búsqueda.

```
Maude> red initials(m,n) .
```

Ilustración 19. Comando initials.

A continuación en la Ilustración 20 y 21, a modo de ejemplo, se hace uso de distintos comandos para una ejecución del protocolo Needham-Schroeder-Lowe with XOR definido en [5].

```
Maude> load examples/Needham_Schroeder_Lowe_XOR.maude
Maude> red summary (0,1).
reduce in MAUDE-NPA : summary(0, 1) .
rewrites: 19883094 in 25120ms cpu (25970ms real) (791524 rewrites/second)
result Summary: States>> 1 Solutions>> 0
Maude> red summary (0,2).
reduce in MAUDE-NPA : summary(0, 2) .
rewrites: 209817 in 228ms cpu (239ms real) (920250 rewrites/second)
result Summary: States>> 2 Solutions>> 0
Maude> red summary (0,4).
reduce in MAUDE-NPA : summary(0, 4) .
rewrites: 5224667 in 3572ms cpu (3608ms real) (1462672 rewrites/second)
result Summary: States>> 3 Solutions>> 0
Maude> red summary (0,8).
reduce in MAUDE-NPA : summary(0, 8) .
rewrites: 4038428 in 3924ms cpu (3964ms real) (1029161 rewrites/second)
result Summary: States>> 3 Solutions>> 1
Maude>
```

Ilustración 20. Comando summary Needham-Schroeder-Lowe with XOR.

Mediante el comando *summary*, en la Ilustración 20, se puede llegar a la conclusión de que hay una solución dentro de los tres estados generados para una profundidad de ocho. Por lo que, si se ejecuta *run* o *initials*, Ilustración 21, se puede analizar el ataque que ha cumplido satisfactoriamente con el *attack-state*, correspondiente en la Ilustración 12.

```
> red run(0,8).
reduce in MAUDE-NPA : run(0, 8) .
rewrites: 507 in 0ms cpu (0ms real) (~ rewrites/second)
result IdSystemSet: (< 1 . 9 . 7 . 13 . 7 . 5 . 6 . 2 . 1 > (
:: nil ::
[ nil |
   -(pk(i, n(a, #0:Fresh) ; a)),
   +(n(a, #0:Fresh) ; a), nil]  &
:: nil ::
[ nil |
   -(pk(i, b * i * n(b, #1:Fresh))),
   +(b * i * n(b, #1:Fresh)), nil]  &
:: nil ::
[ nil |
   -(n(a, #0:Fresh) ; a),
   +(pk(b, n(a, #0:Fresh) ; a)), nil]  &
:: nil ::
[ nil |
   -(n(b, #1:Fresh)),
   +(pk(b, n(b, #1:Fresh))), nil]  &
:: nil ::
[ nil |
   -(b * i),
   -(b * i * n(b, #1:Fresh)),
   +(n(b, #1:Fresh)), nil]  &
:: #1:Fresh ::
[ nil |
   -(pk(b, n(a, #0:Fresh) ; a)),
   +(pk(a, n(a, #0:Fresh) ; b * n(b, #1:Fresh))),
   -(pk(b, n(b, #1:Fresh))), nil]  &
:: #0:Fresh ::
[ nil |
   +(pk(i, n(a, #0:Fresh) ; a)),
   -(pk(a, n(a, #0:Fresh) ; b * n(b, #1:Fresh))),
   +(pk(i, b * i * n(b, #1:Fresh))), nil] )
|
pk(a, n(a, #0:Fresh) ; b * n(b, #1:Fresh)) !inI,
pk(b, n(a, #0:Fresh) ; a) !inI,
pk(b, n(b, #1:Fresh)) !inI,
pk(i, n(a, #0:Fresh) ; a) !inI,
pk(i, b * i * n(b, #1:Fresh)) !inI,
(n(a, #0:Fresh) ; a) !inI,
n(b, #1:Fresh) !inI,
(b * i) !inI,
(b * i * n(b, #1:Fresh)) !inI
|
+(pk(i, n(a, #0:Fresh) ; a)),
-(pk(i, n(a, #0:Fresh) ; a)),
+(n(a, #0:Fresh) ; a),
-(n(a, #0:Fresh) ; a),
+(pk(b, n(a, #0:Fresh) ; a)),
generatedByIntruder(b * i),
-(pk(b, n(a, #0:Fresh) ; a)),
+(pk(a, n(a, #0:Fresh) ; b * n(b, #1:Fresh))),
-(pk(a, n(a, #0:Fresh) ; b * n(b, #1:Fresh))),
+(pk(i, b * i * n(b, #1:Fresh))),
-(pk(i, b * i * n(b, #1:Fresh))),
+(b * i * n(b, #1:Fresh)),
-(b * i),
-(b * i * n(b, #1:Fresh)),
+(n(b, #1:Fresh)),
-(n(b, #1:Fresh)),
+(pk(b, n(b, #1:Fresh))),
-(pk(b, n(b, #1:Fresh)))
|
nil)
```

Ilustración 21. Comando run Needham-Schroeder-Lowe with XOR.

4. Protocolo EMV Estándar

El protocolo EMV es el estándar el cual se implementa dentro de más de 9 mil millones de tarjetas de crédito y débito mediante un seguro circuito integrado, documentación específica en [13], con el único objetivo de poder ser usadas diariamente como método de transacción de pagos. Las especificaciones del protocolo fueron publicadas por primera vez en el año 1996 y son controladas y actualizadas bajo la organización EMVCo [3].

El protocolo requiere de la intervención de tres agentes: una tarjeta con su determinado propietario emitida por el banco emisor, una terminal de pago y el banco emisor, entidad monetaria.

La tarjeta, la cual está asociada a un propietario legítimo, es el agente que interactúa directamente con la terminal intercambiando los datos estáticos y dinámicos necesarios de la tarjeta y del banco con el objetivo de efectuar el pago a través de una transacción fiable y correcta.

La terminal, el agente el cual se encarga de tramitar y verificar mediante certificados, claves públicas y métodos de verificación de propietario (CVM), todos los datos estáticos y dinámicos de la tarjeta para una exitosa transacción bancaria entre el banco y la tarjeta.

Finalmente, el banco es el agente encargado en aceptar y validar que la transacción sea válida mediante el recibo de los datos y mensajes desde la terminal, respondiendo con un mensaje de confirmación o rechazo en cuanto al estado de la transacción.

A lo largo de los años el protocolo ha estado en constante evolución a causa de las distintas brechas de seguridad que se han ido descubriendo haciendo posibles ataques como por ejemplo un Man in the Middle (MITM), permitiendo a intrusos interponerse entre la comunicación de los agentes, ataques de *relay* según [14]: "*consistente en capturar una transmisión de datos correcta y reproducirla posteriormente. Es un ataque típico para capturar secuencias de autenticación correctas y reproducirlas luego para que el atacante logre los mismos derechos de acceso*", o incluso la posibilidad de la clonación de tarjetas.

Pero a pesar de los constantes cambios y actualizaciones que ha recibido el protocolo se demuestra como actualmente el protocolo sigue teniendo brechas de seguridad. En el trabajo actual se estudia, analiza y comprende el estudio *The EMV Standard: Break, Fix, Verify* realizado en el Departamento de Ciencias de la Computación, ETH Zúrich por David Basin, Ralf Sasse, y Jorge Toro-Pozo [4], con el objetivo de poder reproducir y comprobar alguna de las brechas de seguridad las cuales se estudian.

4.1. Descripción

Los primeros pasos antes de empezar con el intercambio de mensajes en el protocolo EMV consiste en que cada agente genere u obtenga acceso a datos previos que son necesarios a lo largo del protocolo.

La tarjeta genera una clave maestra conocida únicamente por la tarjeta y el banco emisor de la tarjeta, el número de la transacción y, por último, un número nonce generado como un valor único antes de iniciar la interacción con la terminal.

La terminal únicamente necesita su propio nonce. El resto de los datos los obtiene de la tarjeta y del banco.

El banco necesita la misma clave maestra compartida con la tarjeta y el número de la transacción de la tarjeta.

El protocolo EMV se ha dividido y nombrado de la misma forma que lo hacen en el estudio [4]. Principalmente se divide en cuatro fases: Inicialización, Autenticación de datos *offline*, Verificación del propietario y Autorización de la transacción.

La primera fase, Sección 4.3.1, consiste mediante la transacción de mensajes entre la terminal y la tarjeta, en llegar a un acuerdo e intercambiar datos estáticos y dinámicos.

La segunda fase, Sección 4.3.2, como indica el propio nombre, consiste en una validación de autenticación de la tarjeta por parte de la terminal de forma *offline* haciendo uso de claves públicas, nonces y sus correspondientes certificados sobre los datos de la fase anterior.

La tercera fase, Sección 4.3.3, la terminal hace uso de los CVM para verificar que el propietario de la tarjeta es la persona la cual está haciendo uso de dicha tarjeta. Esta verificación puede ser *offline*, sin necesidad de comunicarse con el banco, u *online*, comunicándose con el banco.

Finalmente, la fase cuatro, Sección 4.3.4, se compone de una solicitud por parte de la terminal a la tarjeta de qué tipo de transacción usar para posteriormente obtener una respuesta por parte del banco, rechazando o aceptando la transacción que se haya llevado a cabo entre la tarjeta y la terminal.

4.2. Modelos

El chip EMV integrado dentro del circuito cerrado de las tarjetas de crédito puede usarse de dos diferentes formas, el modelo de contacto (*contact*) o el modelo sin contacto (*contactless*) [3][13].

El modelo *contact* se hace uso cuando se inserta directamente la tarjeta con el chip EMV dentro de la terminal. Mientras que, en el *contactless* se hace uso de tecnologías como NFC [2], funcionado simplemente con una ligera aproximación de la tarjeta a la terminal para efectuar la conexión.

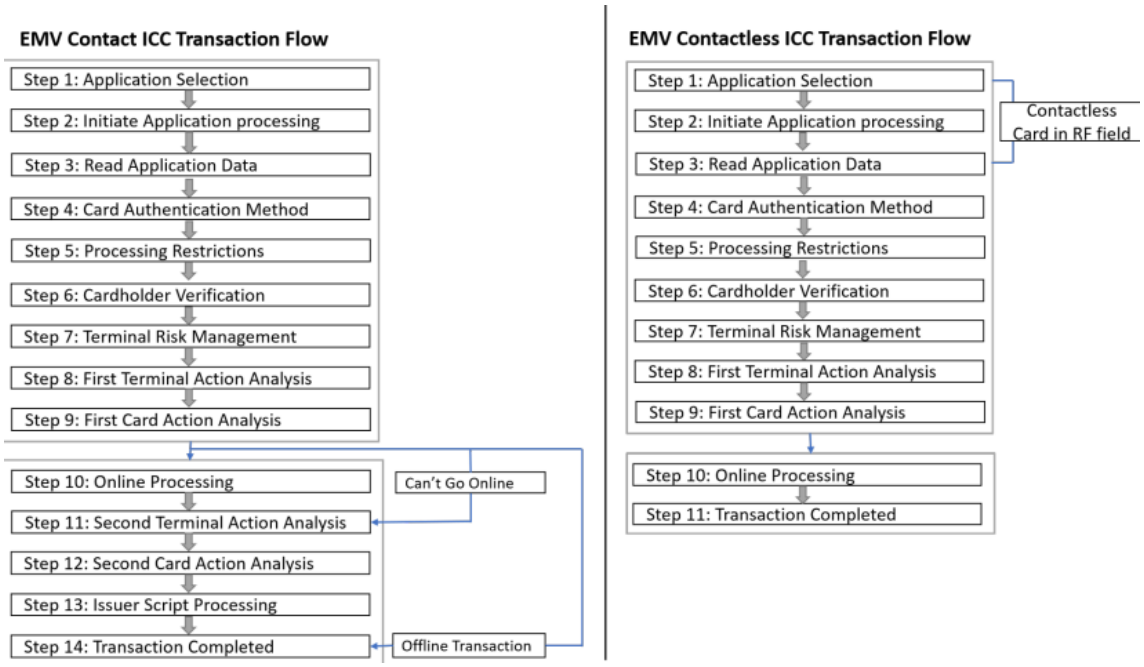

Ilustración 22. Flujo del protocolo EMV para contact y contactless. Extraído de [15].

La principal diferencia entre *contact* y *contactless* reside en la conexión y el tiempo en contacto que hay entre la tarjeta y la terminal. En la Ilustración 22 se observa que solo los tres primeros pasos en *contactless* son los que están la tarjeta y la terminal a una cercana proximidad. Por otro lado, en *contact* todo el proceso está en contacto.

Por una parte, en el modelo *contact* al haberse introducido y por ende establecido una conexión física, las fases que se realizan intercambian datos e información a través del chip EMV, de forma completa y sin ningún tipo de optimización, permitiendo verificar, autenticar y procesar cada mensaje usando todo el tiempo que sea necesario.

Por otra parte, el modelo *contactless* se entra brevemente en contacto e incluso ni llegando a haber una conexión física entre la terminal y la tarjeta. En consecuencia se obliga a que el intercambio de datos e información se vea afectado a un breve instante teniendo que optimizar el proceso para que algunos de los pasos de la transacción sean realizados después de que la tarjeta haya dejado de estar cerca del lector de proximidad o incluso eliminados.

4.3. Fases

A continuación, se ve en más detalle cada una de las fases del protocolo EMV. En la primera Ilustración 23 se observa un flujo de mensajes para el modelo *contact*, y por otro lado, en la segunda Ilustración 24 se muestra el modelo *contactless*.

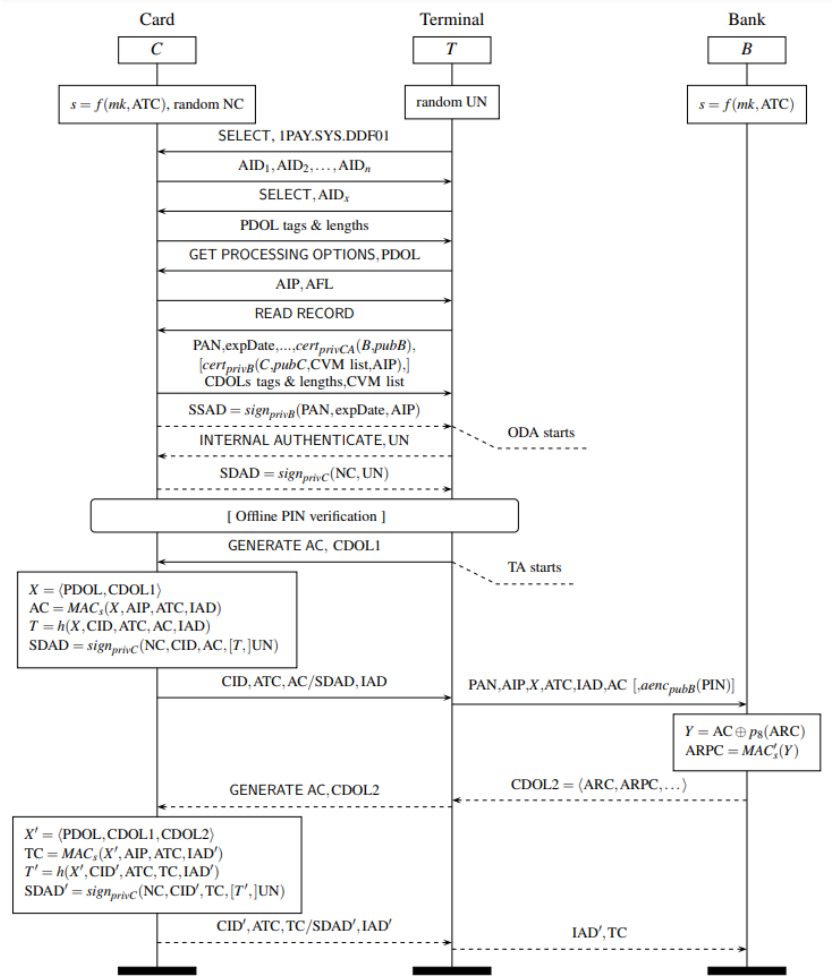

Ilustración 23. EMV protocolo modelo contact. Extraído de [4].

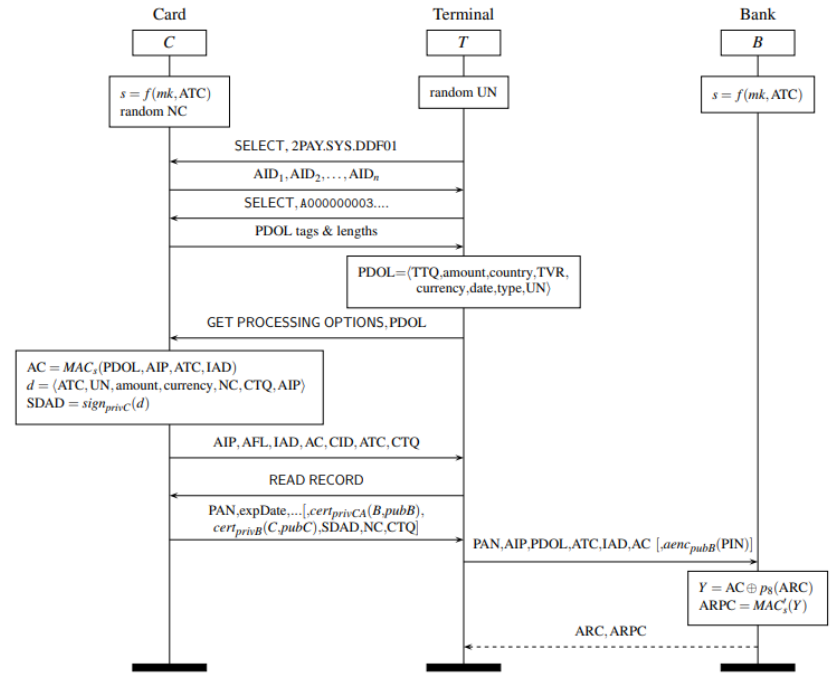

Ilustración 24. EMV protocolo modelo contactless. Extraído de [4].

4.3.1. Inicialización

La Inicialización es la primera fase, la cual empieza con un mensaje por parte de la terminal hacia la tarjeta y se compone de tres principales pasos: selección de la aplicación, procesamiento de opciones y lectura de registros.

El primer paso, la selección de la aplicación, consiste en un envío por parte de la terminal haciendo uso de mensajes SELECT 1PAY.SYS.DDF01 o SELECT 2PAY.SYS.DDF01, para indicar cuál modelo espera usar, *contact* o *contactless* respectivamente. Instantáneamente después, la tarjeta contesta con una lista de aplicaciones, Application Identifier (AID), de la cual la terminal selecciona una de ellas que sean compatible para ambos.

El segundo paso, procesamiento de datos, comienza con una solicitud por parte de la tarjeta a la terminal del Processing Data Object List (PDOL), y esta misma le contesta con GET PROCESSING OPTIONS y el PDOL. El PDOL, generado por la terminal, incluye todos los datos necesarios para la transacción e información como la cantidad, el país, la fecha, el tipo de transacción y un número nonce, Unpredictable Number (UN), generado por la terminal. Seguidamente, la tarjeta responde con la Application Interchange Profile (API) y el Application File Locator (AFL), indicando información sobre las aplicaciones compatibles y sus localizaciones dentro del chip.

Finalmente el último paso, lectura de registros, consiste en un envío del comando READ por parte de la terminal para que posteriormente la tarjeta conteste con datos necesarios para una transacción efectiva. Los datos son, por ejemplo, el Personal Account Number (PAN), la fecha de expiración, los certificados, las claves públicas y SDAD.

Es importante mencionar como se observa en las Ilustraciones 23 y 24, que a pesar de que estos pasos se encuentren dentro de la fase de inicialización de ambos modelos dependiendo del modelo puede cambiar e intercalarse más mensajes o incluso cambiar el orden debido a las distintas optimizaciones que recibe el modelo *contactless*.

Por ejemplo, en la Ilustración 24, la cual muestra *contactless*, dentro del PDOL, también se incluye un dato tipo Terminal Transaction Qualifiers (TTQ) [13][16] el cual incluye una lista de CVM a usar que se muestra en la Sección 4.3.3 y, conjuntamente, el tipo de autorización para la transacción que se ve en la Sección 4.3.4. Otro ejemplo, es el cambio de orden cuando se envían los mensajes API y AFL o el comando READ RECORD, entregando datos como PAN y SDAD varios mensajes después de entrar en contacto.

4.3.2. Autenticación de datos *offline*

La segunda fase, Autenticación de datos *offline*, está compuesta por tres distintos métodos de los cuales se tienen que escoger uno mediante un mutuo acuerdo entre la tarjeta y la terminal: Static Data Authentication (SDA), Dynamic Data Authentication (DDA) o Combined Dynamic Data *Authentication* (CDA).

La primera opción, SDA, la tarjeta envía un mensaje Signed Static Authentication Data (SSAD) el cual se compone de datos estáticos de la tarjeta firmados por el banco, permitiendo a la terminal verificar dichos datos después de obtener la clave pública del banco.

La segunda opción, DDA, la tarjeta envía un mensaje Signed Dynamic Authentication Data (SDAD) el cual se compone del nonce de la terminal (UN) recibido anteriormente en el PDOL, un nonce propio de la tarjeta (NC) y todo firmado por la tarjeta permitiendo a la terminal saber que no se pueda replicar la transacción.

La última opción, CDA, es una combinación de SDA y DDA. Usa los datos estáticos de la tarjeta juntamente con los nonce de la tarjeta y terminal, UN y NC respectivamente para ser posteriormente firmados por la tarjeta.

En la Ilustración 23 se puede apreciar tres intercambios de mensajes en línea discontinua correspondientes con los mensajes SSAD y SDAD ya que son opcionales y dependen de la configuración escogida.

Cada método permite prevenir muchos ataques que reciben las tarjetas como puede ser la modificación de datos de la tarjeta, lo cual evita SDA o la clonación evitada por DDA. Pero, si al final, no hay mutuo acuerdo entre la terminal y la tarjeta, la transacción debe ser *online* o rechazada [3].

4.3.3. Verificación del propietario

La tercera fase, verificación del propietario, se realiza cuando la tarjeta envía a la terminal una lista de CVM para que la terminal escoja los CVM que sean compatibles a ambos. Ejemplos de CVM son: una firma digital sobre una pantalla táctil que disponga la terminal, un dispositivo móvil el cual se desbloquea por huella dactilar o, una contraseña. Pero, el CVM más utilizado y el cual se va a analizar es el Personal Identification Number (PIN), un número secreto conocido únicamente por el propietario de la tarjeta y el banco.

Existen tres tipos de verificación en cuanto al PIN concierne: PIN *offline* sin cifrar o cifrado y PIN *online* cifrado.

La principal diferencia que existe entre el *online* y el *offline* es que el PIN *online* tiene que ser verificado por el banco y no existe ninguna interacción con la tarjeta, mientras que el PIN *offline*, la propia tarjeta es la que se encarga de verificar si el PIN introducido es correcto. En cuanto a la diferencia entre el cifrado o no cifrado, afecta a la visualización del PIN a través de los mensajes, en el primer caso en texto plano y visible para cualquiera, y el segundo encriptado por la clave pública del banco.

En el modelo *contactless*, la lista de verificaciones del propietario compatibles por la terminal se incluye en el TTQ dentro del PDOL. Después, la respuesta de la tarjeta se incluye en el CTQ indicando si el CVM está realizado correctamente y el tipo usado [13][16].

En la Ilustración 23 se puede apreciar dentro de un recuadro la elección del PIN *offline* mientras que en la Ilustración 24 dentro del PDOL se observa el TTQ que luego más tarde la tarjeta responde con un CTQ.

4.3.4. Autorización de la transacción

Por último, la cuarta fase, Autorización de la transacción, después de haber analizado todos los datos recibidos en la inicialización, la autenticación de datos *offline* y la verificación del propietario, la terminal decide si solicitar a la tarjeta una autorización *online*, *offline* o declinar la transacción.

Una vez la terminal decide qué tipo de autorización solicita a la tarjeta, envía el comando GENERATE AC, Ilustración 23, o dentro del PDOL en el mensaje TTQ, Ilustración 24, para que se genere un Application Cryptogram (AC). Un AC puede ser de tres tipos: Authorization Request Cryptogram (ARQC) para solicitar una autorización *online*; Transaction Cryptogram (TC) para solicitar una autorización *offline*, y Application Authentication Cryptogram (AAC) para rechazar la transacción.

Por lo que, una vez la tarjeta reciba la solicitud del AC, genera el AC el cual puede ser después de un análisis propio, el solicitado por la terminal, o si lo considera necesario, cambiarlo a un ARQC o AAC. En cualquier caso, si la terminal decide que necesita un ARQC, la tarjeta nunca podrá contestar con un TC, necesitando forzosamente una verificación *online*.

Un AC en realidad es un Message Authentication Code (MAC), un mensaje el cual se compone del PDOL y AIP recibidos en la Inicialización, el Application Transaction Counter (ATC), el número de la transacción y el Issuer Application Data (IAD), Ilustración 23 y 24. El MAC está cifrado por una clave simétrica únicamente conocida por la tarjeta y el banco, ya que se genera a partir de la clave maestra, *mk*, y el ATC.

Además, la tarjeta envía junto al AC todos los datos en texto plano para que la terminal pueda reenviarlos al banco para completar la verificación. También, se envía un SDAD firmado por la tarjeta con los nounces transmitidos previamente, UN y NC, y datos necesarios para la transacción (CTQ en el caso de *contactless*) permitiendo autenticar a la tarjeta ante modificaciones.

El MAC y el SDAD son los dos mensajes más importantes que envía la tarjeta a la terminal, pero dependiendo de qué tipo de AC esté contestando la tarjeta, la terminal procede de una forma u otra.

Cuando la tarjeta contesta con un AC tipo ARQC, la terminal reenvía todos los datos necesarios al banco para que este sea el responsable de aceptar o denegar la transacción mediante el Authorization Response Cryptogram (ARPC) y, en consecuencia, la terminal comunicándole el resultado a la tarjeta para que posteriormente la tarjeta contesta con un TC (si la transacción fue aceptada) o un AAC (si la transacción fue rechazada). No obstante, este segundo AC solo sucede en el modelo *contact*.

Por otro lado, si la tarjeta contesta un AC tipo TC, la terminal acepta la transacción como válida sin consultar con el banco para posteriormente enviar el TC, en forma de resguardo, al banco informando de la transacción.

El ARPC, se obtiene a partir de una operación XOR entre el AC y Authorization Response (ARC), un código para autorizar o declinar el AC recibido.

Una diferencia que se puede apreciar entre las Ilustraciones 23 y 24, es que en el modelo *contactless* no se espera un segundo envío de un AC tipo TC, sino que da por concluido la transacción. Mientras que por otra parte la Ilustración 23, muestra una nueva generación de un TC para enviárselo al banco después de haber recibido un ARPC correcto. Otra de las diferencias entre el *contact* y *contactless* es que mientras que en *contact* el AC se solicita con el comando GENERATE AC, en *contactless* esta solicitud viene incluida dentro del PDOL en el TTQ. La última de las diferencias y una de las más notables en cuanto a seguridad, es que dependiendo de la configuración escogida en *contactless* el SDAD de Ilustración 24 que está entre corchetes podría ser omitido, mientras que en *contact* sí es enviado y comprobado por la terminal [4].

4.4. Análisis del protocolo

En este apartado se muestra cómo en el estudio [4] haciendo uso de un lenguaje Tamarin, una herramienta muy similar a Maude-NPA para la verificación de protocolos criptográficos, se quiere comprobar si se cumplen tres propiedades de seguridad.

La primera propiedad consiste en comprobar que todas las transacciones aceptadas por la terminal tienen que ser aceptadas también por el banco.

La segunda propiedad dice que todas las transacciones que la terminal acepte han debido ser autenticadas. La autenticación puede ser *offline,* y por tanto, debe ser la tarjeta la encargada de la autenticación y, si se trata de una autenticación *online,* es el banco el encargado de hacerlo.

La última propiedad que se pretende observar es que todas las transacciones aceptadas por el banco están en consecuencia autenticadas por la tarjeta y terminal.

4.4.1. Ataques

Los ataques son aquellas estrategias que se han podido llevar a cabo para conseguir romper una o varias propiedades descritas anteriormente para las tarjetas VISA. A continuación, se muestran y explican cuáles son las propiedades que han roto y algunos ejemplos de cómo se pueden aprovechar los fallos haciendo uso de distintos ataques. Existen diversas noticias [17][18] de revistas reconocidas publicando artículos sobre los descubrimientos que hicieron en [4] y los cuales se van a analizar a continuación.

El primer ataque surge en los modelos *contact* y *contactless* donde se ha podido analizar que cuando se hace uso de SDA o DDA, durante la autenticación de datos *offline*, se rompe la primera y segunda propiedad. Esto sucede ya que durante el modelo *contact,* cuando se genera un AC de tipo TC, el TC se puede modificar y la terminal no es capaz de autenticar si es correcto, ya que es una MAC cifrada por una clave simétrica entre la tarjeta y el banco, aceptando la transacción para que posteriormente el banco la rechace debido a su inconsistencia. Lo mismo sucede para el modelo *contactless*, pero en este caso cualquier tipo de AC no se

puede autenticar la veracidad ante la terminal y en consecuencia aceptando cualquier AC para que posteriormente el banco lo rechace.

Unauthenticated Offline Transactions, Ilustración 25, descrito en el estudio hace uso de este ataque. El ataque se aprovecha durante una transacción de baja cantidad para que no se requiera autenticación *online* y por tanto, solicitando un AC tipo TC en el mensaje PDOL de la Ilustración 25 incluido en el TTQ. Entonces, cuando el TC se envía, el intruso lo captura para poder modificar el TC a un nuevo TCI cuando se esté usando DDA o SDA y, en consecuencia, la terminal acepta la transacción como válida con el fin de que posteriormente el banco la rechace al existir una inconsistencia. Pero el atacante ya ha obtenido lo que está "pagando" sin ningún cobro en la cuenta de la tarjeta al ser la transacción rechazada por el banco y aceptada por la terminal.

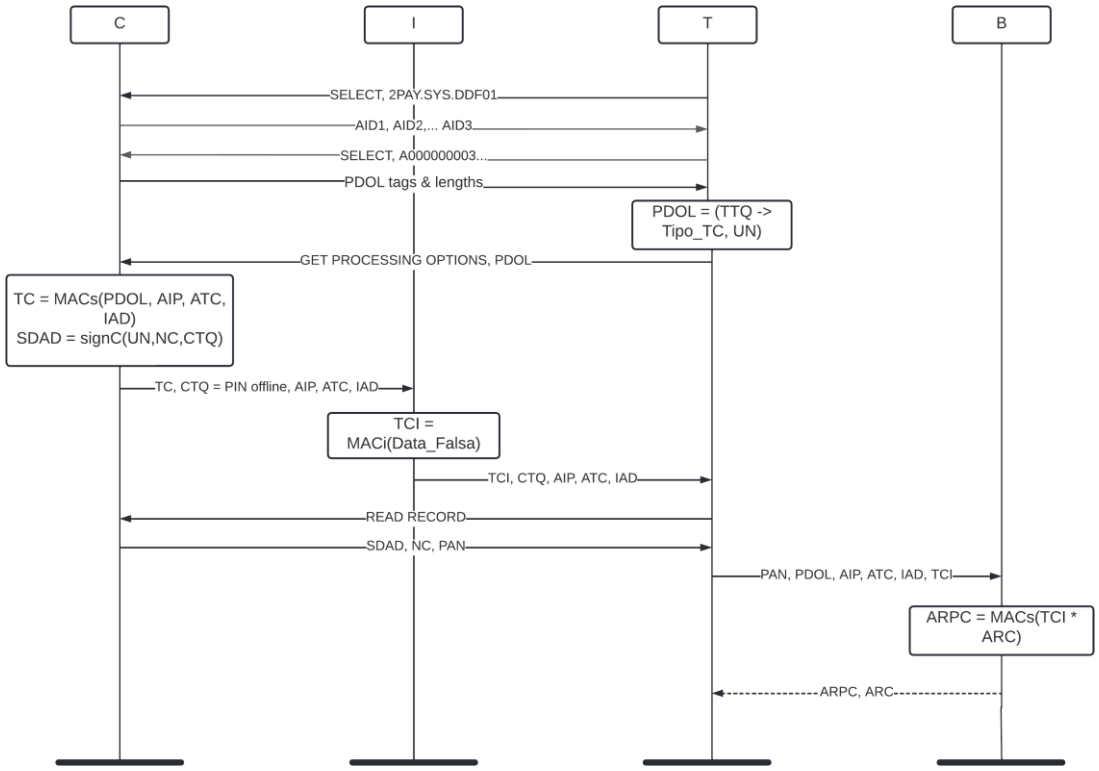

Ilustración 25. Ataque Unauthenticated Offline Transactions.

El segundo ataque no incumple ninguna de las propiedades, pero se considera una filtración de información privada que el atacante podría obtener. En el modelo *contact* cuando hay intercambio de claves públicas y privadas son todas secretas y el atacante no las va a poder obtener de ninguna forma, pero el PIN y el PAN sí se pueden obtener. El PAN se puede ver en texto claro y sin cifrar analizando los mensajes, pero el PIN es posible obtenerlo también. Para ello, se modifica y obliga a la terminal que el CVM compatible a usar tiene que ser el de PIN *offline* sin cifrar, para posteriormente de la misma forma que el PAN analizar el PIN entre los mensajes. Esto se puede conseguir si se obtiene la clave privada de un banco y acceso físico a la terminal para modificar el TTQ.

El tercer ataque, Ilustración 26, sucede en el modelo *contact* incumpliendo la segunda y tercera propiedad cuando se hace uso del PIN *offline* y SDA. El ataque permite saltarse la seguridad del PIN haciendo uso de un MITM [6], interponiéndose entre la tarjeta y la terminal.

Este ataque se lleva a cabo bloqueando la solicitud de la terminal a la tarjeta de un CVM para un PIN *offline* haciendo creer a la tarjeta que no es necesario ningún tipo de CVM y enviando a la terminal un mensaje diciendo que el PIN fue insertado correctamente y por tanto, poder seguir con la transacción autorizada.

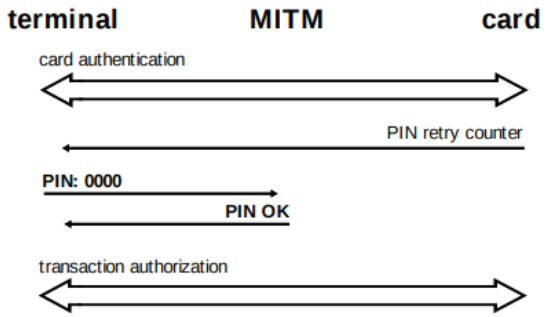

Ilustración 26. Ataque MITM al PIN. Extraído de [6].

El último ataque se ejecuta en el modelo *contactless* incumpliendo la segunda y tercera propiedad cuando el CTQ se modifica para transacciones de alto valor. Dentro del CTQ, como se ha explicado en la Sección 4.3.3, se incluye el CVM que la tarjeta va a usar y si fue realizado correctamente. Entonces el CTQ al estar solo protegido criptográficamente cuando usa DDA, la terminal no puede verificar la autenticidad del CTQ si no se ha usado SDAD. En consecuencia, acepta la verificación del propietario como correcta, donde el atacante puede haber modificado el CTQ para saltarse el CVM y que la terminal y el banco obtengan un CTQ diferente al de la tarjeta.

Bypassing Cardholder Verification, Ilustración 27, descrito en el estudio hace uso de este ataque. El ataque se aprovecha de este fallo de configuración para modificar el CTQ indicando a la terminal que no es necesario el PIN *online* (bit 8 del primer byte [16]) y que el CVM fue insertado correctamente (bit 8 bit del segundo byte [16]) para seguir con la transacción sin una correcta verificación de propietario por parte de la terminal ni del banco. El ataque se consigue cuando la configuración no usa DDA y por tanto el atacante puede capturar el CTQ para modificarlo por uno propio, Ilustración 27, ya que el mensaje SDAD no existe, el cual es necesario para comprobar si se ha modificado el CTQ. El resultado de este ataque consigue saltarse los límites de la tarjeta para transacciones *online* y sin introducir el PIN o cualquier otro CVM.

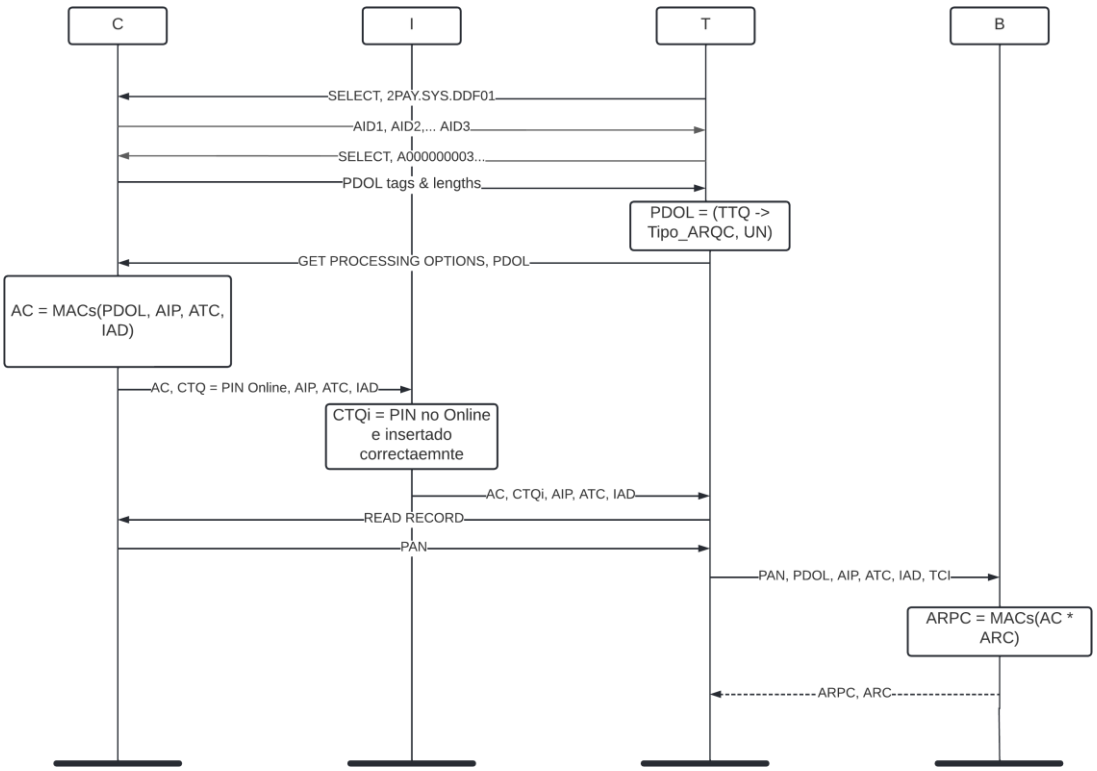

Ilustración 27. Ataque Bypassing Cardholder Verification.

4.4.2. Defensa

La defensa son las distintas estrategias para protegerse de los ataques descritos en la Sección 4.4.1. Existen muchas técnicas defensivas en [3], algunas de ellas pueden ser simplemente evitando el uso de algún tipo de función básica del protocolo, y otras añadiendo barreras para evitar modificaciones u obtención de información sensible. También, el estudio [4] sugiere soluciones para evitar cambiar el protocolo EMV de los *chips* y así poder seguir usando las tarjetas en circulación.

Una primera línea de defensa que existe se trata de la más básica, y es la comprobación por parte de la tarjeta y la terminal de que todos los datos y aplicaciones escogidos son los correctos, cumpliendo todos los mínimos de seguridad y restricciones. De esta forma se evita protocolos no actualizados y débiles que permitan que un atacante se aproveche de ellos.

La segunda línea de defensa está en la parte de la terminal y tarjeta correspondiente con el análisis basado en su propia configuración, por ejemplo, a partir de unos criterios comunicarse de forma *online* con el banco para poder proceder con la transacción. Los criterios pueden ser desde una cantidad máxima a la hora del pago, de forma aleatoria o incluso mantener una cuenta máxima de transacciones *offline* poniendo un límite de su uso. Con esta simple medida de seguridad se puede evitar robos que exploten las transacciones *offline*.

La tercera línea de defensa es añadiendo complejidad a las claves debido a que existe un ataque, que a pesar de que teóricamente se pueda obtener el PIN mediante el PIN *offline* sin

cifrar haciendo uso de claves privadas y teniendo acceso físico a la terminal, se sabe que es prácticamente imposible haciendo fuerza bruta debido a la tecnología actual la cual no permite hacerlo.

Una cuarta línea de defensa es evitar radicalmente el uso de la autenticación *offline*, ya que se ha visto que los AC son vulnerables a las modificaciones, por lo que haciendo uso de autenticación *online* la terminal se ve obligada a consultar al banco si la transacción debe de ser denegada o aceptada. Asimismo, durante las transacciones *online* usar DDA para comprobar siempre si ha habido modificación del CTQ mediante el SDAD. También, si no queda más remedio que usar autenticación *offline* se debe como medida de seguridad insertar en el SDAD el propio AC junto con los nounces, así evitando las modificaciones del AC, siendo esta configuración posible de verificar por parte de la terminal.

El estudio señala, que de todas las pruebas que se hicieron, la presencia de transacciones *offline* era prácticamente nula, por lo que el bajo uso de ellas es al mismo tiempo un buen mecanismo de defensa. Además, el uso de DDA durante transacciones *online* está prácticamente ligado haciéndose uso de esta configuración la gran mayoría de las ocasiones.

4.4.3. Conclusión del análisis

En el siguiente apartado se puede observar los distintos resultados que se obtienen después de la ejecución para las distintas combinaciones de posibles configuraciones haciendo uso de la herramienta Tamarin para el modelo *contact* y *contactless*.

TABLE I
ANALYSIS RESULTS FOR THE EMV CONTACT PROTOCOL. ALL TARGET MODELS HAVE 55 RULES. THE LAST TWO COLUMNS INDICATE, IN THAT ORDER, THE NUMBER OF LINES OF TAMARIN CODE THAT THE MODEL COMPRISES, AND THE TIME TAKEN FOR OUR ANALYSIS, USING TAMARIN V1.5.1 ON A COMPUTING SERVER RUNNING UBUNTU 16.04.3 WITH TWO INTEL(R) XEON(R) E5-2650 V4 @ 2.20GHZ CPUS (WITH 12 CORES EACH) AND 256GB OF RAM. HERE WE USED 10 THREADS AND AT MOST 20GB OF RAM PER MODEL. THE MODELS FOR WHICH ALL FOUR PROPERTIES WERE VERIFIED ARE HIGHLIGHTED IN BOLD.

No.	Target model	Properties				LoC	Time
		executable	*bank accepts*	*auth. to terminal*	*auth. to bank*		
1	Contact_SDA_PlainPIN_Online	✓	×(2)	×(1,2)	×(1)	758	13m07s
2	Contact_SDA_PlainPIN_Offline	✓	×(2)	×(1,2)	×(1)	761	11m39s
3	Contact_SDA_OnlinePIN_Online	✓	×(2)	×(1,2)	×(1)	758	13m02s
4	Contact_SDA_OnlinePIN_Offline	–	–	–	–	731	11m48s
5	Contact_SDA_NoPIN_Online	✓	×(2)	×(1,2)	×(1)	752	8m21s
6	Contact_SDA_NoPIN_Offline	✓	×(2)	×(1,2)	×(1)	755	6m37s
7	Contact_SDA_EncPIN_Online	–	–	–	–	758	12m21s
8	Contact_SDA_EncPIN_Offline	–	–	–	–	761	11m36s
9	Contact_DDA_PlainPIN_Online	✓	×(2)	×(1,2)	×(1)	766	13m48s
10	Contact_DDA_PlainPIN_Offline	✓	×(2)	×(1,2)	×(1)	769	12m20s
11	Contact_DDA_OnlinePIN_Online	✓	×(2)	×(2)	✓	775	16m04s
12	Contact_DDA_OnlinePIN_Offline	–	–	–	–	739	12m27s
13	Contact_DDA_NoPIN_Online	✓	×(2)	×(2)	✓	769	12m15s
14	Contact_DDA_NoPIN_Offline	✓	×(2)	×(2)	✓	772	8m43s
15	Contact_DDA_EncPIN_Online	✓	×(2)	×(1,2)	×(1)	766	14m07s
16	Contact_DDA_EncPIN_Offline	✓	×(2)	×(1,2)	×(1)	769	12m59s
17	Contact_CDA_PlainPIN_Online	✓	✓	×(1)	×(1)	763	1h55m31s
18	Contact_CDA_PlainPIN_Offline	✓	✓	×(1)	×(1)	766	14m10s
19	**Contact_CDA_OnlinePIN_Online**	✓	✓	✓	✓	781	6h03m05s
20	Contact_CDA_OnlinePIN_Offline	–	–	–	–	739	12m15s
21	**Contact_CDA_NoPIN_Online**	✓	✓	✓	✓	775	2h31m23s
22	**Contact_CDA_NoPIN_Offline**	✓	✓	✓	✓	778	12m16s
23	Contact_CDA_EncPIN_Online	✓	✓	×(1)	×(1)	763	1h59m44s
24	Contact_CDA_EncPIN_Offline	✓	✓	×(1)	×(1)	766	14m00s

Legend:
✓: property verified ×: property falsified –: not applicable
(1): disagrees with the card on the CVM used (2): disagrees with the card on the last AC

Ilustración 28. Configuraciones y propiedades para el modelo contact. Extraído de [4].

En el modelo *contact*, como se puede observar en la Ilustración 28, se consigue que solo cumplan tres configuraciones (las que están en negrita) de todas las combinaciones las tres propiedades de seguridad. Las tres configuraciones hacen uso de una autenticación de datos *offline*, CDA, pero únicamente una de las tres hace uso del CVM (PIN *online*), para comprobar que el propietario es quien está usando la tarjeta. Las otras dos no hacen uso de CVM por lo que en la teoría sólo sería la primera opción la configuración más segura a usar.

El resto de las configuraciones no son seguras para la autenticación de datos *offline*, SDA o CDA, debido al segundo AC necesario en el modelo *contact* de tipo TC, al ser vulnerable a modificaciones, ya que la terminal no es capaz de verificarlo. Tampoco son seguras las configuraciones que usan un CVM tipo PIN *offline* debido a la inconsistencia que se crea entre la tarjeta, la terminal y el banco, ante las modificaciones.

TABLE II
ANALYSIS RESULTS FOR THE EMV CONTACTLESS PROTOCOL. ALL TARGET MODELS HAVE 60 RULES. HERE WE USED THE SAME SETUP AS WITH THE EXPERIMENTS SHOWN IN TABLE I. AGAIN, THE MODEL(S) FOR WHICH ALL FOUR PROPERTIES WERE VERIFIED ARE HIGHLIGHTED IN BOLD.

No.	Target model	Properties				LoC	Time
		executable	*bank accepts*	*auth. to terminal*	*auth. to bank*		
1	Visa_EMV_Low	✓	✓	×(1)	×(1)	823	1m26s
2	Visa_EMV_High	✓	✓	×(1)	×(1)	823	1m30s
3	Visa_DDA_Low	✓	×(2)	×(2)	✓	832	31m41s
4	**Visa_DDA_High**	✓	✓	✓	✓	841	25m00s
5	Mastercard_SDA_OnlinePIN_Low	✓	×(2)	×(2)	✓	831	4m22s
6	**Mastercard_SDA_OnlinePIN_High**	✓	✓	✓	✓	840	12m28s
7	Mastercard_SDA_NoPIN_Low	✓	×(2)	×(2)	✓	825	4m11s
8	Mastercard_SDA_NoPIN_High	–(3)	–	–	–	793	43s
9	Mastercard_DDA_OnlinePIN_Low	✓	×(2)	×(2)	✓	837	8m18s
10	**Mastercard_DDA_OnlinePIN_High**	✓	✓	✓	✓	846	27m08s
11	Mastercard_DDA_NoPIN_Low	✓	×(2)	×(2)	✓	831	8m11s
12	Mastercard_DDA_NoPIN_High	–(3)	–	–	–	799	47s
13	**Mastercard_CDA_OnlinePIN_Low**	✓	✓	✓	✓	846	19m44s
14	**Mastercard_CDA_OnlinePIN_High**	✓	✓	✓	✓	846	47m21s
15	**Mastercard_CDA_NoPIN_Low**	✓	✓	✓	✓	840	18m38s
16	Mastercard_CDA_NoPIN_High	–(3)	–	–	–	799	49s

Legend:
✓: property verified ×: property falsified –: not applicable (1): disagrees with the card on the CVM used
(2): disagrees with the card on the AC (3): high-value transactions without CVM are not completed over the contactless interface

Ilustración 29. Configuraciones y propiedades para el modelo contactless. Extraído de [4].

Por otro lado, el modelo *contactless* obtiene los resultados en la Ilustración 29. Únicamente existe una configuración (la que están en negrita) que cumple con las propiedades de seguridad de entre solo las cuatro primeras filas, las cuales son las que corresponden a tarjetas VISA. Esta configuración es segura cuando la transacción es *online* al tratarse de una cantidad alta de dinero y hace uso de la autenticación de datos DDA para comprobar la modificación del CTQ.

Mientras que el resto de las configuraciones para VISA, no cumplen con las propiedades de seguridad. Las versiones EMV tanto para transacciones bajas como las transacciones altas son vulnerables a la modificación del CTQ debido a que la terminal no puede verificarlo al no usar DDA. Tampoco cuando se usa DDA, para evitar la modificación del CTQ, si la transacción es baja no se puede controlar la modificación del AC tipo TC.

En conclusión, el protocolo EMV tanto para el modelo *contact* y *contactless* tienen muchos problemas cumpliendo las tres propiedades de seguridad propuestas, y por tanto, en cuanto a seguridad concierne. Principalmente, el modelo *contactless* para las tarjetas VISA es el más vulnerable al tener únicamente una configuración segura.

5. Especificación del modelo EMV en Maude-NPA

El principal objetivo de este trabajo ha sido analizar, estudiar y entender el estudio realizado en [4], el cual descubre distintos ataques que gracias al modelado con la herramienta Tamarin han sido capaces de romper alguna de las propiedades de seguridad que se han establecido. En consecuencia, se comprueba si los ataques del estudio están en lo correcto y por tanto asegurar que el protocolo EMV no es seguro o por lo contrario no llegar a ningún resultado y declarar que el protocolo sí es seguro. El trabajo se replica haciendo uso de la herramienta Maude-NPA, existiendo trabajos previos que demuestran la posibilidad de crear una especificación de protocolos escritos en Tamarin a Maude-NPA [11].

En la Sección 3 se ha visto como Maude-NPA funciona y sus distintas funcionalidades que permiten definir y declarar un modelado del protocolo EMV en un fichero maude compuesto por tres módulos. Mientras que en la Sección 4 se ha explicado cada una de las fases, propiedades de seguridad y los distintos ataques que el estudio ha encontrado permitiendo entender que se pretende replicar.

En los siguientes apartados se va a crear una especificación para el modelo *contactless* de las tarjetas VISA, ya que es el más optimizado de los dos modelos que existen y por ende el más directo de replicar. También, se ha decidido usar *contactless* y VISA debido a que son el modelo y la tarjeta más vulnerable de entre todas las opciones y por tanto, mayor número de ataques se han efectuado satisfactoriamente.

5.1. Sintaxis y propiedades algebraicas

El primer paso se trata de definir la sintaxis en Maude-NPA para EMV, definida en el primer módulo del fichero maude, como en la Sección 3.1.1. El primer módulo está compuesto por los diferentes *sort* y *subsorts*, Ilustración 30, y sus distintos operadores, Ilustración 31.

Los tipos que se van a usar para el protocolo son, primera línea Ilustración 30: Name, Nonce, Key, Mac, Ttq, Ctq Realizado, NoRealizado, PinOffline, PinOnline, TC, ARQC, CVM, PIN y AC.

Un *sort* Name para los distintos agentes del protocolo (tarjeta, terminal, intruso y banco) que se ven en la Sección 4.1, un Nonce con todos los valores que van a ser únicos por ejecución e irrepetibles, un Key para la clave compartida entre banco y tarjeta y un Mac para los mensajes AC del protocolo.

Un mensaje MAC [19] es la salida de un mensaje cifrado con una clave simétrica, en el caso de EMV es la clave maestra. Un mensaje MAC se espera que tenga una longitud única e independientemente del mensaje de entrada y que, al mismo tiempo, debe de ser imposible de determinar el mensaje original sin la clave, incluso teniendo otros mensajes cifrados con la

misma clave. Por lo que un mensaje MAC permite al banco identificar si ha habido modificación de algún tipo al AC y por tanto denegar la transacción en la fase de autorización de la transacción en la Sección 4.3.4.

Del mismo modo se han creado otros *sorts* como Ttq y Ctq, los cuales se usan para los mensajes TTQ y CTQ del protocolo EMV respectivamente. Realizado y NoRealizado se emplean para indicar a la terminal si el CVM ha sido insertado correctamente por el propietario de la tarjeta, y es por ello por lo que son *subsorts* del *sort* CVM. Además, existen los tipos PinOffline y PinOnline con el objetivo de indicar si el PIN escogido por la tarjeta de la lista de CVM disponibles entre la terminal y la tarjeta, visto en la Sección 4.3.3, no tiene que ser verificado por el banco (PinOffline) o si debe enviar un mensaje al banco (PinOnline), siendo estos a su vez *subsorts* del tipo PIN. Finalmente, TC y ARQC *subsorts* del tipo AC (no confundir con los tipos MAC) que se sitúa dentro de la solicitud en el mensaje TTQ en la fase de inicialización de la Sección 4.3.1, para indicar el tipo de AC que la terminal ha escogido.

Hay que mencionar, que todos los *sort* son *subsorts* de Msg y además el tipo Name es *subsort* de Public para indicar que todos los agentes son públicos.

```
--- Sort Information
sorts Name Nonce Key Mac Ttq Ctq Realizado NoRealizado PinOffline PinOnline TC ARQC
CVM PIN AC .
subsort PinOffline PinOnline < PIN .
subsort TC ARQC < AC .
subsort Realizado NoRealizado < CVM .
subsort Name Nonce Key Mac Ttq Ctq PIN CVM AC < Msg .
subsort Name < Public .
```

Ilustración 30. Sorts y subsorts EMV.

Los operadores con el fin de ser utilizados en los intercambios de mensajes son, Ilustración 31: n, c, t, i, key, kei, read, select, pdolC, pdolT, ttq, ctq, pan, data, sig, mac y _;_.

El operador n recibe un Name y un Fresh para crear y usar los Nonce, siendo un número único y aleatorio asociado a un nombre y así que no pueda ser replicado o robado.

El conjunto de operadores i,c y t se usan para los distintos agentes en el protocolo EMV correspondientes con el intruso, tarjeta y terminal, devolviendo simplemente un tipo Name y sin argumentos.

La clave compartida entre el banco y la tarjeta hace uso del operador key, como argumentos recibe dichos dos agentes que la conocen. También existe el operador kei para representar la operación donde el intruso crea su propia clave y no necesita argumentos.

Los comandos del protocolo EMV para solicitar e intercambiar datos entre la terminal y la tarjeta, también necesitan sus operadores: read, select, pdolC y pdolT. Todos los operadores reciben como mínimo dos argumentos tipo Name para indicar quién está mandando a quién el mensaje. El operador select sirve para indicar el modelo *contactless*, mientras que read es para solicitar el PAN, SDAD y NC si la configuración está lista para ello. Conjuntamente, el operador pdolC solicita los datos necesarios a la terminal y pdolT es la respuesta con los correspondientes datos a la tarjeta. El operador pdolT, a su vez, recibe dos argumentos extra, el primero para indicar en el mensaje el tipo Ttq y el segundo su propio Nonce.

El operador ttq devuelve un TTQ y tiene como argumento el tipo de AC que se quiere usar haciendo uso de TC para transacciones *offline* y ARQC para la *online*. Por otra parte, está el

operador ctq creando un CTQ y como argumento obtiene el modo del PIN (PinOffline o PinOnline) y si el CVM fue realizado o no (Realizado o NoRealizado) por parte del propietario de la tarjeta.

De la misma manera, en un momento del protocolo se necesita el PAN y para ello se usa el operador pan. Por otro lado, los datos como el ATC, AIP y IAD, se representan con un operador llamado data para simplificar el procesado y búsqueda por parte de Maude-NPA.

Del mismo modo, cuando sea necesario el SDAD, el protocolo hace uso del operador sig para firmar mensajes y asegurar su autenticidad. Los AC son mensajes tipo Mac como se ha visto anteriormente, y necesitan su propio operador mac para ser usado con su clave y respectivo mensaje como argumentos.

Finalmente, aparte de todos los operadores mencionados anteriormente se necesita un operador para la concatenación de mensajes, y por ello se aplica el axioma punto y coma (;).

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

--- Principals
op c : -> Name . --- Card
op t : -> Name . --- Terminal
op i : -> Name . --- Intruder

--- Key shared
op key : Name Name -> Key .

--- Key generada por intruso
op kei : -> Key .

--- Comandos
op read : Name Name -> Msg .
op select : Name Name -> Msg .
op pdolC : Name Name -> Msg .
op pdolT : Name Name Nonce Ttq -> Msg .

--- CTQ y TTQ
op ttq : AC -> Ttq .
op ctq : PIN CVM -> Ctq .

--- Mensajes necesarios
op pan : Name Name -> Msg .
op data : -> Msg .  ---Corresponde con el aip, atc y iad

--- Firmar un mensaje
op sig : Name Msg -> Msg .

--- MAC (Message Authentication Code) de un mensaje
op mac : Key Msg -> Mac .

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

Ilustración 31. Operadores EMV.

El segundo paso después de definir los tipos y operadores es definir las distintas propiedades algebraicas, definidas en el segundo módulo del fichero maude, como en la Sección 3.1.2. Durante el protocolo EMV, en esta ocasión, para facilitar la búsqueda y modelado de los ataques se ha omitido su uso, ya que las propiedades criptográficas que se usan durante el protocolo (XOR, clave asimétrica) no se han visto afectadas en cuanto a la seguridad se refiere,

afectando exponencialmente en el tiempo necesario de búsqueda para las posibles trazas de los ataques.

5.2. Strands

En el tercer y último módulo se van a definir los *strands* de las habilidades Dolev-Yao para los distintos agentes y también, los *attack-state*, del mismo modo que en la Sección 3.1.3 y 3.1.4.

En este módulo sobre la especificación del protocolo es necesario la declaración de variables para el correcto funcionamiento, ya que son usadas en los distintos *strands*. En la Ilustración 32 se observa una variable para la clave compartida, llamada S, también la variable TTQ de tipo Ttq y otra CTQ para el tipo Ctq. Asimismo, unas variables tipo Msg son útiles para las habilidades Dolev-Yao como DATA_FALSA necesaria para uno de los ataques y PAN. Adicionalmente se usa una variable Offline de tipo PinOffline con el fin de indicar que no se requiere comunicación con el banco para verificar el propietario, y otra Online de tipo PinOnline, donde indica que es necesario la comunicación con el banco. También, Rea de tipo Realizado, con el objetivo de indicar a la terminal que el PIN fue insertado, y NoRead de tipo NoRealizado para informar de que es necesario insertar el PIN. Estas variables, Rea y NoRea, se usan en las operaciones ctq conjuntamente las variables Offline y Online. Igualmente, se necesita una variable Tipo_TC y Tipo_ARQC de tipo TC y ARQC respectivamente para el operador ttq.

Finalmente, variables para los nombres de los agentes y los diferentes Nonce que se van a crear.

```
var S : Key .
var TTQ : Ttq .
var CTQ : Ctq .
vars X Y DATA_FALSA PAN : Msg .
var Offline : PinOffline .
var Online : PinOnline .
var Rea : Realizado .
var NoRea : NoRealizado .
var Tipo_TC : TC .
var Tipo_ARQC : ARQC .
var r : Fresh .
vars C T B : Name .
vars UN NC : Nonce .
```

Ilustración 32. Variables EMV.

5.2.1. Dolev-Yao Habilidades

Las habilidades del intruso son las que muestra la Ilustración 33 dentro de la ecuación llamada STANDS-DOLEVYAO. Como se puede observar, las habilidades son prácticamente todas las opciones que pueden hacer los agentes legítimos en el sistema, y por tanto, el intruso. Por ejemplo, algunas de las habilidades que se pueden observar son la posibilidad de extraer mensajes de una concatenación o, concatenar mensajes mediante el operador punto y coma.

El objetivo de estas habilidades es que el intruso pueda hacer uso de ellas para obtener el patrón de búsqueda, *attack-pattern*, dentro del *attack-state*. De entre todas las opciones existen habilidades como crear una MAC con una clave generada por el propio intruso, necesaria para reproducir uno de los ataques que se muestra en la Sección 5.2.3. También, la creación de un CTQ el cual ya ha realizado correctamente el CVM y negando la necesidad de usar el PIN *online*. Esta última habilidad en Maude-NPA se ha implementado con operaciones y argumentos, pero en realidad, como bien se indica en [13][16], hay una gran variedad de configuraciones para los TTQ y CTQ cambiando los distintos bits de los bytes para su diferente uso.

Finalmente, de las últimas habilidades usadas existe la posibilidad de que el intruso quiera generar nonces propios o mensajes firmados con su propio nombre.

```
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), +(mac(kei, X)), nil ] &
:: nil :: [ nil | +(ctq(Offline,Rea)), nil ] &
:: r :: [ nil | +(n(i,r)), nil ] &
:: nil :: [ nil | -(X), +(sig(i, X)), nil ]
[nonexec] .
```

Ilustración 33. Habilidades Dolev-Yao EMV.

5.2.2. Protocolo

El protocolo definido en la Ilustración 24, que se muestra en la Sección 4.3 desarrollados por [4] para el modelo *contactless*, muestra todas las posibles configuraciones que podrían usarse para este modelo. Mientras que, en el actual apartado, se ha desarrollado un modelado más específico para comprobar ataques determinados para una configuración o configuraciones concretas del modelo *contactless*, así pudiendo centrarse en obtener los resultados que se pretenden comprobar y no hacer un análisis completo al modelo.

La Ilustración 34 especifica la secuencia de envíos de mensajes del protocolo EMV para el modelo *contactless* haciendo uso de los operadores descritos en la Sección 5.1, con el objetivo de poder replicarlo con los distintos *strands* en Maude-NPA posteriormente. También, se ha creado el diagrama de secuencia de la especificación con la finalidad de poder ilustrarlo gráficamente, Ilustración 35.

```
1. T → C : select(T,C)
2. C → T : pdolC(C,T)
3. T → C : pdolT(T,C,UN,TTQ)
4. C → T : mac(S,pdolT,data) ; data ; CTQ
5. T → C : read(T,C)
6. C → T : [sig(C,UN,NC,CTQ) ; NC] ; PAN
7. T → B : PAN ; pdolT ; mac(S,pdolT,data) ; data
8. B → T : mac(S,mac(S,pdolT,data) * ARC) ; ARC
```

Ilustración 34. EMV protocolo.

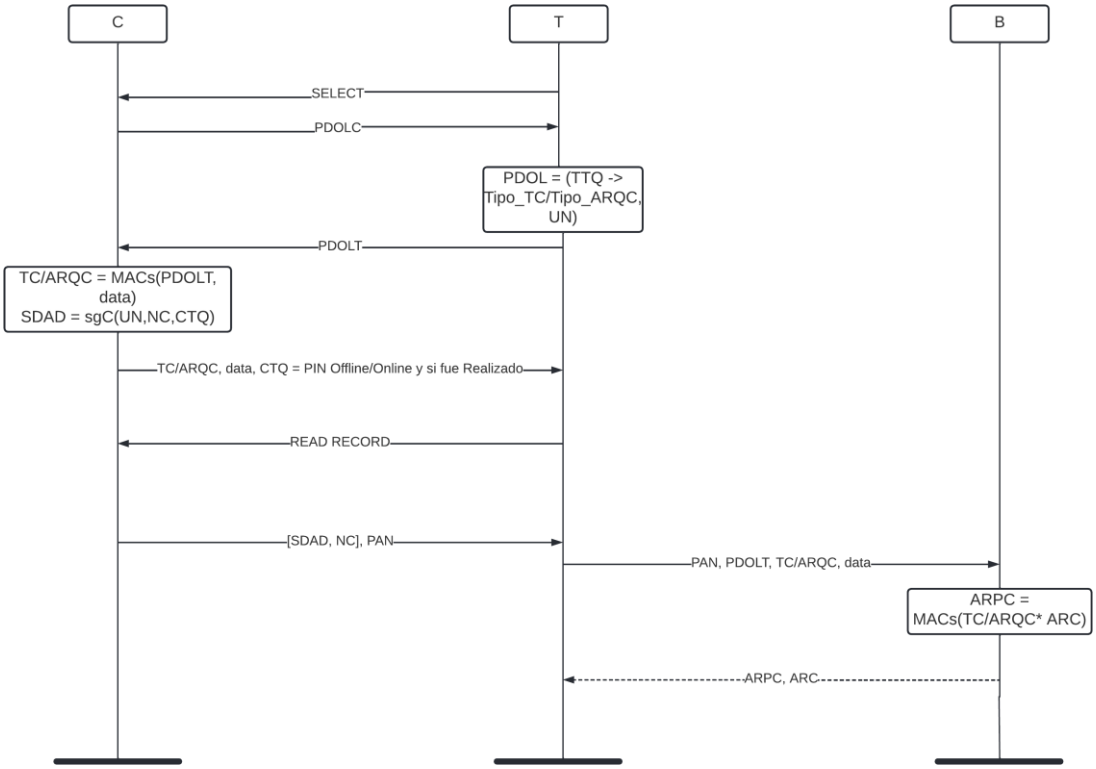

Ilustración 35. Diagrama de secuencia contactless EMV.

En la Ilustración 34 y 35 se muestra el modelado *contactless* con los términos que se usan en la especificación de Maude-NPA. El AC puede variar dependiendo de si es transacción *online* u *offline*, lo cual, está indicado en el TTQ que genera la terminal, como en la Sección 4.3.1, solicitando un tipo de transacción TC o ARQC. Por otra parte, los datos que están entre corchetes son opcionales del mismo modo que sucedía durante la Sección 4.3.4 y sólo se utilizan cuando se haga uso de autenticación DDA, visto en Sección 4.3.2. Finalmente, el mensaje en línea discontinua depende de la configuración y pasos previos escogidos.

El primer mensaje de la Ilustración 35, a diferencia de la Sección 4.3.1, la terminal empieza la comunicación con la tarjeta enviando un único mensaje SELECT, simplemente, para indicar en la fase de inicialización que se está ejecutando el modelo *contactless*.

Segundo y tercer mensaje, la tarjeta solicita el PDOL a la terminal. La terminal contesta con el PDOL el cual incluye el nonce UN y el TTQ solicitando una transacción AC tipo TC (*offline*), o ARQC (*online*). Nótese que en comparación con el *contactless* de la Sección 4.3 en Ilustración 24, la lista de AIDs se ha omitido al ser un dato constante y sin variación.

Cuarto mensaje, la tarjeta envía el AC, un mensaje tipo Mac cifrado con la clave compartida entre el banco y la tarjeta, en Ilustración 35, representado con la letra *s*. El MAC contiene el PDOL y datos extra utilizados por la operación data (AIP, ATC y IAD). También, el mensaje data se remite de forma separada y sin cifrar conjuntamente el CTQ.

Quinto mensaje, la terminal envía el comando READ con el objetivo de que la tarjeta le conteste con el PAN, y en el caso de usar DDA, el SDAD y NC también. El SDAD se trata de un mensaje firmado por la tarjeta e incluye el UN, extraído del PDOL anteriormente, el CTQ y su propio nonce NC.

Sexto, la terminal reenvía al banco todos los datos necesarios si se trata de una transacción *online* para que el banco pueda confirmar la transacción, quedándose a la espera de su respuesta o finalizando la transacción. En el caso de una transacción *offline* reenvía posteriormente todos los datos y recibe la confirmación del banco. Los datos necesarios son el PAN recibido de la tarjeta, el PDOLT, generado por la propia terminal, el AC y data, también generado por la tarjeta.

Por último, séptimo mensaje, el banco ejecuta una operación XOR entre el AC recibido y un mensaje ARC para indicar si la transacción ha sido aceptada o no, enviando la respuesta de vuelta a la terminal.

A continuación, después de haber definido la función de cada mensaje se planea convertir a Maude-NPA cada uno de los mensajes dentro de la ecuación STRANDS-PROTOCOL. Con este objetivo se muestra, haciendo uso de las distintas operaciones y variables, los *strands* creados en Maude-NPA para la terminal y tarjeta, representado todos los mensajes recibidos y enviados en Ilustración 34 y 35 en las configuraciones, Ilustración 36 y 37.

Sin embargo, el banco no se le ha incluido su *strand*, ya que en términos de ataques lo único que procesa es la confirmación o denegación de las transacciones. El atacante no puede modificar la respuesta del banco ni tampoco lo que la terminal le haya enviado, ya que se tiene el conocimiento de que la comunicación entre banco y terminal es segura, y el atacante no tiene la posibilidad de modificar ni observar los mensajes a su favor.

En la Sección 5.2.3 se van a crear los *attack-states* necesarios para reproducir los dos ataques del modelo *contactless* vistos en la Sección 4.4.1. El primero necesita una configuración que no use DDA y que sea de una alta cantidad de dinero, solicitando una transacción *online*. El segundo, por otra parte, hace uso de una configuración con DDA, pero con una cantidad baja de dinero y por ende, sin necesidad de contactar con el banco tratándose de una transacción *offline*. Por esta razón, se han modelado dos distintas configuraciones mediante dos ecuaciones STRANDS-PROTOCOL para cada uno de los ataques.

Por un lado, la primera de las configuraciones, Ilustración 36, donde no se está usando DDA y por tanto no existen los mensajes en corchetes de la Ilustración 35. Así como, al tratarse de una transacción *online*, exige un AC tipo ARQC y un CTQ con PIN *online*.

```
eq STRANDS-PROTOCOL
= :: r ::
--- TERMINAL
[ nil |       +(select(T,C)),
-(pdolC(C,T)),
+(pdolT(T,C,n(T,r),ttq(Tipo_ARQC))),
-(mac(S, pdolT(T,C,n(T,r),TTQ) ; data) ; data ; CTQ),
+(read(T,C)),
-(pan(C,T)), nil ] &
:: r ::
--- CARD
[ nil |       -(select(T,C)),
+(pdolC(C,T)),
-(pdolT(T,C,UN,TTQ)),
+(mac(key(C,B), pdolT(T,C,UN,TTQ) ; data) ; data ; ctq(Online,NoRea)),
-(read(T,C)),
+(pan(C,T)), nil ]
[nonexec] .
```

Ilustración 36. Strand Visa_EMV _Online.

Por otro lado, la segunda de las configuraciones, Ilustración 37, donde sí se está usando DDA, y por ende se crea un SDAD, se hacen uso de los mensajes entre corchetes. Así como, usando una transacción *offline* necesitando un AC tipo TC y un CTQ con PIN *offline*.

```
eq STRANDS-PROTOCOL
= :: r ::
--- TERMINAL
[ nil |       +(select(T,C)),
-(pdolC(C,T)),
+(pdolT(T,C,n(T,r),ttq(Tipo_TC))),
-(mac(S, pdolT(T,C,n(T,r),TTQ) ; data) ; data ; CTQ),
+(read(T,C)),
-(sig(C, n(T,r) ; NC ; CTQ) ; NC ; pan(C,T)), nil ] &

:: r ::
--- CARD
[ nil |       -(select(T,C)),
+(pdolC(C,T)),
-(pdolT(T,C,UN,TTQ)),
+(mac(key(C,B), pdolT(T,C,UN,TTQ) ; data) ; data ; ctq(Offline,Rea)),
-(read(T,C)),
+(sig(C, UN ; n(C,r) ; ctq(Offline,Rea)) ; n(C,r) ; pan(C,T)), nil ]
[nonexec] .
```

Ilustración 37. Strand Visa_DDA_Offline.

En las Ilustraciones 36 y 37, lo que está en negrita son las variables y mensajes que se diferencian la una de la otra: el tipo del AC, los bits del CTQ y el uso, o no, del SDAD.

5.2.3. Attack-states

En este apartado, dentro de la ecuación ATTACK-STATE, se explican los distintos patrones de ataque que se han deseado replicar.

El *attack-state* en Maude-NPA para el primer ataque es el siguiente, Ilustración 38. Donde lo que pretende efectuar el atacante es interceptar todos los mensajes de la terminal y en el momento en el que recibe el CTQ cambiarlo a un nuevo CTQ forzando a que no se necesite la verificación *online*, en este caso se ha optado por usar la palabra *offline*, e indicando que el CVM está insertado correctamente por el propietario. Las modificaciones son posibles gracias al hacer uso de las habilidades de la Sección 5.2.1. El CTQ generado por el intruso se puede observar en la Ilustración 38 en negrita.

```
eq ATTACK-STATE(0)
= :: r ::
--- TERMINAL (Protocolo con intruso crea un nuevo CTQ, MITM)
[ nil ,       +(select(t,c)),
-(pdolC(c,t)),
+(pdolT(t,c,n(t,r),ttq(Tipo_ARQC))),
-(mac(S, pdolT(T,C,n(T,r),TTQ) ; data) ; data ; ctq(Offline,Rea)),
+(read(t,c)),
-(pan(c,t)) | nil ]
|| empty
|| nil
|| nil
|| nil
[nonexec] .
```

Ilustración 38. Attack-state Bypassing Cardholder Verification.

El *attack-state* en Maude-NPA para el segundo ataque es el siguiente, Ilustración 39. El atacante del mismo modo que el ataque anterior, intercepta todos los mensajes de la terminal. Pero en esta ocasión, con el objetivo de bloquear el AC original generado por la tarjeta, creando un MAC completamente nuevo a partir de las habilidades de la Sección 5.2.1. El AC recibido en la terminal no se encuentra dentro del SDAD, como es el caso del CTQ, por lo que se acepta el AC como correcto al no poder verificar si se ha producido algún tipo de modificación. El AC creado por el intruso se puede observar en la Ilustración 39 en negrita.

```
eq ATTACK-STATE(0)
= :: r ::
--- TERMINAL (Protocolo con intruso crea una mac falsa, MITM)
[ nil ,       +(select(t,c)),
-(pdolC(c,t)),
+(pdolT(t,c,n(t,r),ttq(Tipo_TC))),
-(mac(kei, DATA_FALSA) ; data ; CTQ),
+(read(t,c)),
-(sig(c, n(t,r) ; NC ; CTQ) ; NC ; pan(c,t)) | nil ]
|| empty
|| nil
|| nil
|| nil
[nonexec] .
```

Ilustración 39. Attack-state Unauthenticated Offline Transaction.

5.3. Ejecución del protocolo

En este apartado se usa la herramienta Maude-NPA para su ejecución y obtención de resultados para los dos *attack-state* que se han descrito en la Sección 5.2.3, haciendo uso de los comandos que disponen Maude-NPA que se han explicado y se han descritos en la Sección 3.2.

5.3.1. Ejecución de comandos

El primer paso antes de obtener algún tipo de resultado, se trata de cargar mediante el comando *load*, Ilustración 40, las opciones que ofrece Maude-NPA y el fichero maude correspondiente a la primera configuración declarada en la Sección 5.2.2 con el primer *attack-state* de la Sección 5.2.3, *Bypassing Cardholder Verification*.

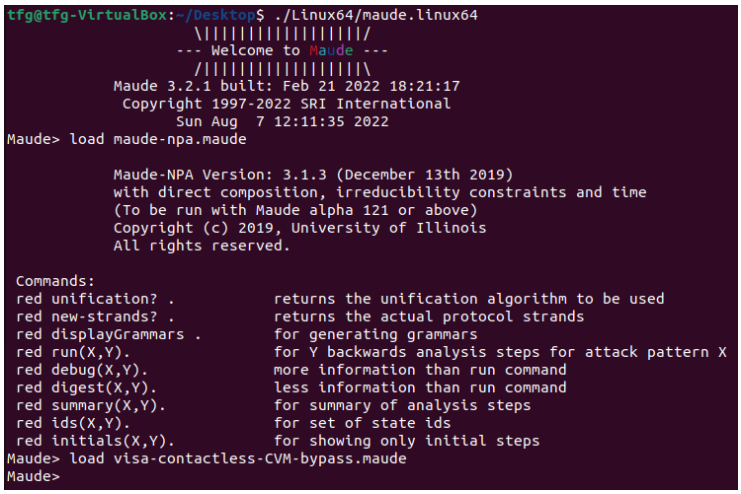

Ilustración 40. Load maude-npa y fichero maude.

El segundo paso es ejecutar *summary* en busca de soluciones para cierto nivel de profundidad. Este proceso puede llevar varios minutos, e incluso horas, hasta que Maude-NPA haya computado todas las opciones posibles mediante su búsqueda hacia atrás. En la Ilustración 41, el protocolo es ejecutado para distintas profundidades llegando hasta la profundidad once con 106 estados creados donde se obtiene una solución.

```
Maude> red summary(0,1).
reduce in MAUDE-NPA : summary(0, 1) .
rewrites: 138 in 0ms cpu (0ms real) (~ rewrites/second)
result Summary: States>> 2 Solutions>> 0
Maude> red summary(0,2).
reduce in MAUDE-NPA : summary(0, 2) .
rewrites: 210 in 0ms cpu (0ms real) (~ rewrites/second)
result Summary: States>> 4 Solutions>> 0
Maude> red summary(0,4).
reduce in MAUDE-NPA : summary(0, 4) .
rewrites: 897 in 0ms cpu (0ms real) (~ rewrites/second)
result Summary: States>> 26 Solutions>> 0
Maude> red summary(0,8).
reduce in MAUDE-NPA : summary(0, 8) .
rewrites: 8949 in 4ms cpu (7ms real) (2237250 rewrites/second)
result Summary: States>> 126 Solutions>> 0
Maude> red summary(0,10).
reduce in MAUDE-NPA : summary(0, 10) .
rewrites: 13571 in 16ms cpu (26ms real) (848187 rewrites/second)
result Summary: States>> 109 Solutions>> 0
Maude> red summary(0,11).
reduce in MAUDE-NPA : summary(0, 11) .
rewrites: 15734 in 24ms cpu (48ms real) (655583 rewrites/second)
result Summary: States>> 106 Solutions>> 1
Maude>
```

Ilustración 41. Comando summary Bypassing Cardholder Verification.

El tercer y último paso es hacer uso del comando *run* o *initials* para obtener la traza de mensajes que han sido necesarios para la reproducción del ataque. Los mensajes de la Ilustración 42 en azul oscuro indican la recepción de mensajes, mientras que los rojos son el envío de mensajes y, el azul claro, los mensajes generados por el intruso.

```
+(select(t, c)),
-(select(t, c)),
+(pdolC(c, t)),
-(pdolC(c, t)),
+(pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC))),
-(pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC))),
+(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ; ctq(#5:PinOnline, #6:NoRealizado)),
-(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ; ctq(#5:PinOnline, #6:NoRealizado)),
+(data ; ctq(#5:PinOnline, #6:NoRealizado)),
-(data ; ctq(#5:PinOnline, #6:NoRealizado)),
+(data),
generatedByIntruder(ctq(#0:PinOffline, #1:Realizado)),
-(data),
-(ctq(#0:PinOffline, #1:Realizado)),
+(data ; ctq(#0:PinOffline, #1:Realizado)),
-(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ; ctq(#5:PinOnline, #6:NoRealizado)),
+(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data)),
-(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data)),
-(data ; ctq(#0:PinOffline, #1:Realizado)),
+(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ; ctq(#0:PinOffline, #1:Realizado)),
-(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ; ctq(#0:PinOffline, #1:Realizado)),
+(read(t, c)),
-(read(t, c)),
+(pan(c, t)),
-(pan(c, t))
```

Ilustración 42. Traza Bypassing Cardholder Verification, Maude-NPA.

De igual manera, se siguen los mismos pasos para la segunda de las configuraciones de la Sección 5.2.2 con el segundo *attack-state* de la Sección 5.2.3 para el ataque *Unauthenticated Offline Transactions.*

Primero, cargando el módulo de Maude-NPA si no se ha hecho previamente, y el fichero maude del protocolo para dicha configuración.

Segundo, ejecutando *summary*, Ilustración 43, hasta encontrar una solución en el caso de que existiera para el ataque mencionado. En esta ocasión la solución se encuentra a una profundidad ocho con una creación de 112 estados.

```
Maude> red summary(0,1).
reduce in MAUDE-NPA : summary(0, 1) .
rewrites: 154 in 0ms cpu (0ms real) (~ rewrites/second)
result Summary: States>> 3 Solutions>> 0
Maude> red summary(0,2).
reduce in MAUDE-NPA : summary(0, 2) .
rewrites: 298 in 0ms cpu (0ms real) (~ rewrites/second)
result Summary: States>> 8 Solutions>> 0
Maude> red summary(0,4).
reduce in MAUDE-NPA : summary(0, 4) .
rewrites: 1124 in 0ms cpu (0ms real) (~ rewrites/second)
result Summary: States>> 28 Solutions>> 0
Maude> red summary(0,8).
reduce in MAUDE-NPA : summary(0, 8) .
rewrites: 8152 in 8ms cpu (5ms real) (1019000 rewrites/second)
result Summary: States>> 112 Solutions>> 1
Maude>
```

Ilustración 43. Comando summary Unauthenticated Offline Transactions.

Finalmente, obtener la traza de mensajes mostrada en la Ilustración 44, entre la terminal, intruso y tarjeta, con el fin de comprobar si se ha llevado a cabo con éxito el ataque de la forma esperada.

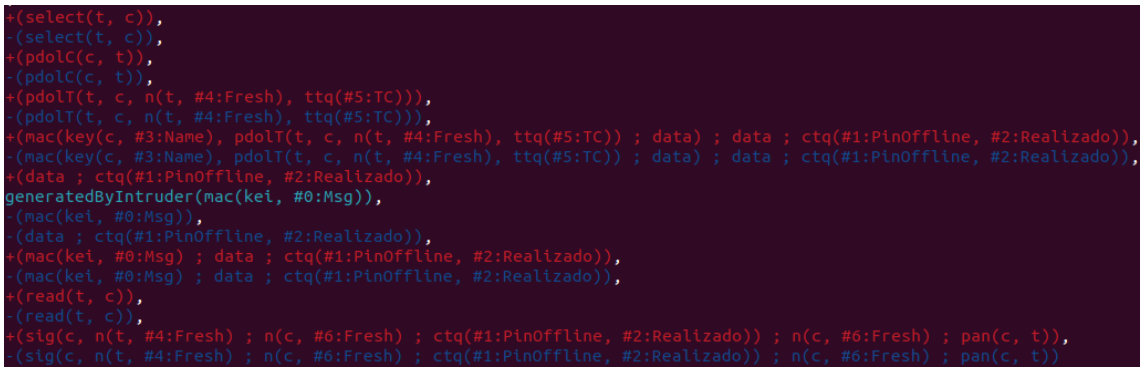

Ilustración 44. Traza Unauthenticated Offline Transactions, Maude-NPA.

5.3.2. Resultados de los ataques

El primer ataque, Ilustración 42, es un MITM, interponiéndose entre la tarjeta y la terminal para una configuración que no usa DDA, y por tanto tampoco CDA. Asimismo, el ataque se realiza con una transacción de cantidad de dinero que supere el límite del *offline*, y por ende, tenga que ser un AC tipo ARQC, visto en la Sección 4.4.1 llamado por [4], *Bypassing Cardholder Verification*.

El intruso reenvía todos los mensajes de la tarjeta a la terminal y viceversa, pero cuando la tarjeta genera el CTQ indicando el uso del PIN *online* necesario de verificación por el banco, el intruso lo modifica para evitar la verificación del CVM e indicar que fue ingresado correctamente. El ataque es posible hacerlo debido a que no usa DDA, y por tanto el CTQ no está protegido ante modificaciones.

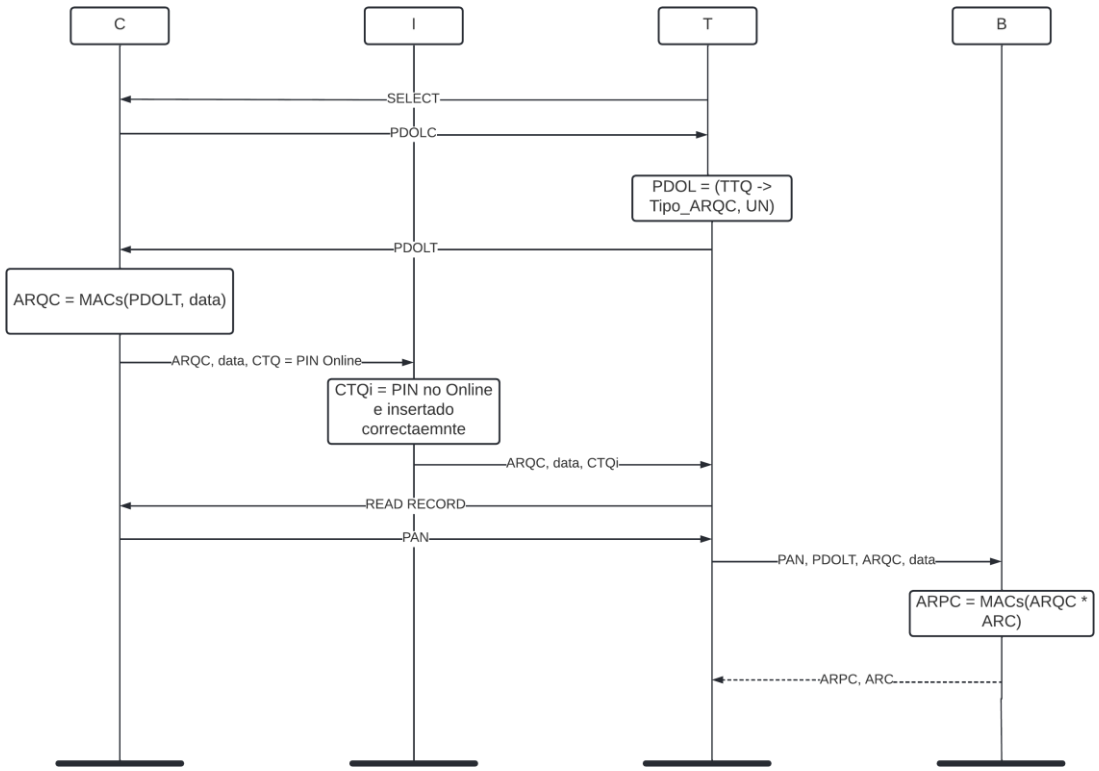

Ilustración 45. Ataque Bypassing Cardholder Verification, Maude-NPA.

El ataque que se obtiene se ha ilustrado mediante un diagrama de secuencia en la Ilustración 45. El primer mensaje, como se puede observar, se inicia de forma correcta por parte de la terminal enviando el comando select. La contestación, segundo mensaje, por parte de la tarjeta solicita los datos necesarios para la transacción con el comando pdolC. La terminal genera un PDOL el cual contiene los datos necesarios como el nonce UN y un TTQ, que solicita un AC tipo ARQC, para posteriormente enviarlo todo en el tercer mensaje con el operador pdolT.

No obstante, cuando la tarjeta genera un ARQC con la data y el PDOL se manda en un cuarto mensaje el CTQ con las opciones PinOnline conjuntamente el ARQC y data en texto plano. En este momento, el intruso extrae el CTQ de la concatenación de mensajes con el AC tipo ARQC y la data, con el objetivo de concatenar su propio CTQi generado con las opciones PinOffline, para que la terminal no solicite la verificación del banco y Realizado, para informar a la terminal que no es necesario hacer nada más en cuanto a verificación de propietario.

Seguidamente, una vez la terminal reciba el mensaje modificado con el nuevo CTQi, la secuencia de mensajes continúa de forma normal con un quinto mensaje haciendo uso del comando read por parte de la terminal a la tarjeta y finalizando con el envío en un sexto mensaje del PAN, concluyendo la traza del ataque que se había obtenido en la Ilustración 42.

Aún así, aunque, no se ve reflejado en la traza de la Ilustración 42, la terminal continúa como se observa en la Ilustración 45, enviando en un séptimo mensaje el PAN, PDOLT, ARQC y data, para que la transacción pueda ser confirmada por el banco llevándose a cabo una operación XOR y contestando a la terminal con un último mensaje ARPC y ARC. Finalmente, dando éxito a una transacción sin verificar ni insertar el CVM para una cantidad superior al límite establecido.

El segundo ataque, Ilustración 44, es también un MITM, visto en la Sección 4.4.1, nombrado *Unauthenticated Offline Transactions* por [4] entre la tarjeta y la terminal para una configuración que usa DDA, y por tanto, comprueba mediante el SDAD el CTQ. Además, el ataque funciona cuando se trata de una transacción de cantidad de dinero baja de modo que se necesite un AC tipo TC, el cual corresponde con el modo *offline*.

El intruso reenvía los mensajes entre la terminal y la tarjeta, y viceversa, de la misma forma que hacía en el ataque anterior, pero en esta ocasión el intruso crea un AC completamente nuevo a partir de una nueva clave propia. Este ataque es posible de realizar ya que es imposible de verificar por parte de la terminal si el AC tipo TC es correcto, haciendo que el banco no acepte la transacción posteriormente.

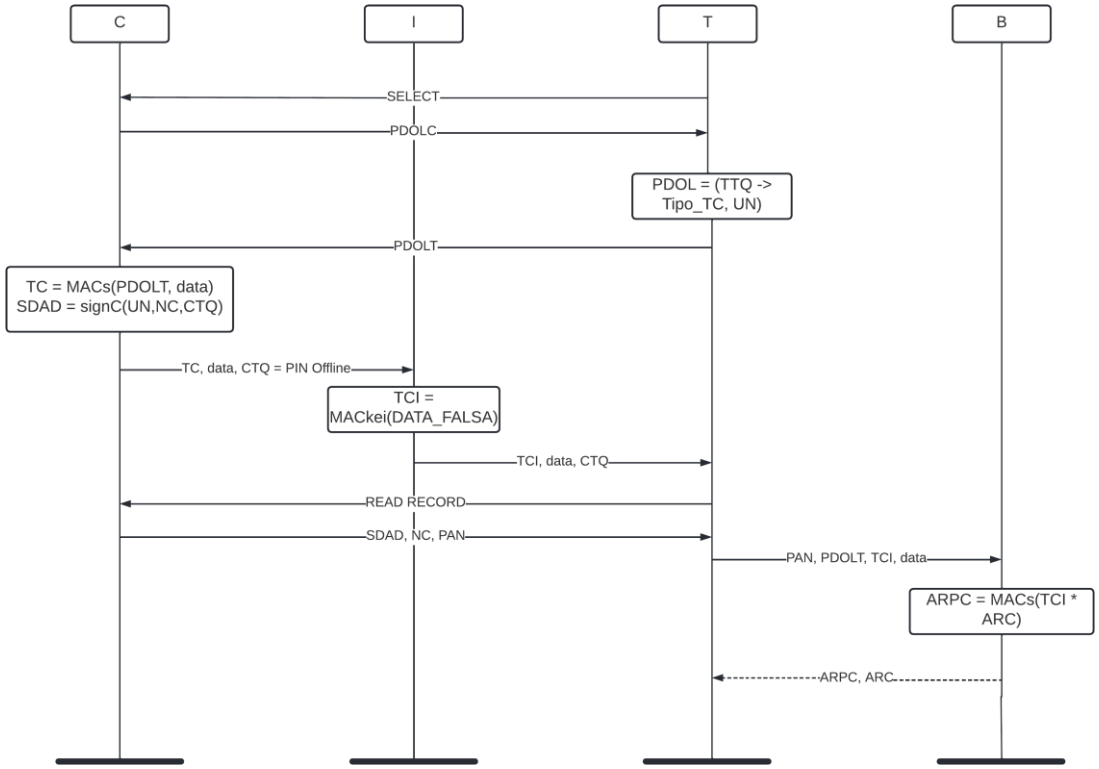

Ilustración 46. Ataque Unauthenticated Offline Transactions, Maude-NPA.

El ataque que se obtiene se ha ilustrado mediante un diagrama de secuencia en la Ilustración 46. De la misma forma que sucedía con el primer ataque, los dos primeros mensajes suceden exitosamente, siendo el primer mensaje una inicialización de forma correcta por parte de la terminal enviando el comando select, y también, con la contestación a través del segundo mensaje por parte de la tarjeta solicitando los datos necesarios para la transacción con el comando pdolC.

La terminal genera de la misma forma un PDOL el cual contiene los datos necesarios como el nonce UN y un TTQ, pero en esta ocasión se solicita un AC tipo TC, para posteriormente enviarlo todo en el tercer mensaje con el operador pdolT.

La gran diferencia entre los ataques es debido al uso del DDA y una transacción *offline*, teniendo que generar un SDAD con el UN, un propio nonce NC, el CTQ y asimismo, un AC tipo TC con la data y el PDOL. Todo se manda en un cuarto mensaje, el CTQ con las opciones

PinOffline conjuntamente el TC y data. En ese preciso instante, el intruso extrae el TC de la concatenación de mensajes con el CTQ y la data, con la intención de concatenar su propio TCI, un MAC con la clave *kei* generada por el intruso con información falsa. La terminal acepta el nuevo AC tipo TC como correcto, ya que no puede comprobar de ninguna manera al estar cifrado por una clave común entre el banco y la tarjeta si ha habido modificación.

A continuación, la terminal recibe el mensaje modificado con el nuevo TCI, la secuencia de mensajes continúa de forma normal con un quinto mensaje read por parte de la terminal a la tarjeta de la misma forma que el ataque anterior. Finalizan con el envío en un sexto mensaje del PAN con el SDAD y el NC, con el objetivo de que la terminal compruebe si ha habido modificación en el CTQ, concluyendo la traza del ataque que se había obtenido en la Ilustración 44.

Al tratarse de una transacción *offline* la tarjeta acabaría su función y el atacante se iría sin ningún problema en el proceso. Posteriormente, de la misma forma que el ataque anterior, no se ve reflejado en la traza de la Ilustración 44, la terminal continúa como se observa en la Ilustración 46, enviando en un séptimo mensaje el PAN, PDOLT, TCI y data, para que la transacción pueda ser denegada posteriormente por el banco. El banco lleva a cabo una operación XOR contestando a la terminal con un último mensaje ARPC y ARC denegando la transacción debido al AC incorrecto. Finalmente, con la denegación de la transacción el ataque es un éxito, no cobrándose ningún tipo de cargo a la tarjeta para la transacción *offline*.

En el ataque anterior modificar el AC tipo ARQC no sería beneficioso, ya que lo que se intenta es que la transacción se realice con éxito, mientras que en este ataque se busca que la transacción falle de manera *offline*, haciendo que el banco la decline posteriormente.

5.3.3. Verificación del estudio realizado

El modelado del protocolo EMV en Tamarin [4] ha sido posible ser adaptado a una versión con mayor especificación mediante Maude-NPA como se ha visto en trabajos previos [11] permitiendo la adaptación de los ataques, *Unauthenticated Offline Transactions* y *Bypassing Cardholder Verification* los cuales han sido replicados de forma efectiva.

Los resultados realizados en [4] han sido idénticos a los que se han obtenido mediante el entendimiento, el modelado y la ejecución en Maude-NPA de las distintas versiones y configuraciones que ofrece el protocolo EMV. El resultado de la ejecución del modelado de los distintos ataques demuestra y verifica la existencia de estos, confirmando gran parte del trabajo realizado por el Departamento de Ciencias de la Computación, ETH Zúrich por David Basin, Ralf Sasse, y Jorge Toro-Pozo.

En consecuencia, al tratarse de una verificación positiva del trabajo, se afirma la existencia de los distintos fallos que rompen las propiedades de seguridad, creando un riesgo en el uso del protocolo modelo *contactless* para las tarjetas VISA. Por lo que, dichos fallos, deben ser solucionados para asegurar el correcto funcionamiento para cada una de las configuraciones del protocolo EMV.

En la Sección 4.4.2 se sugieren mecanismos de defensa, los cuales se pueden usar para asegurar las tarjetas VISA y que además son posibles de ser adaptados al actual modelo en Maude-NPA. Las transacciones *online* deben ser usadas conjuntamente DDA para evitar modificaciones del CTQ denegando la posibilidad de efectuar el ataque *Bypassing Cardholder Verification.* Mientras que si se usa una transacción *offline* se debe insertar el AC dentro del SDAD y usar en consecuencia DDA, evitando la posibilidad del ataque *Unauthenticated Offline Transactions*.

5.3.4. Conclusión de la ejecución

En los últimos apartados se ha observado cómo ejecutar con Maude-NPA el protocolo modelado EMV, la obtención de los distintos resultados y la comparativa y verificación con el estudio [4].

Los resultados del protocolo, posteriores a la ejecución, son los esperados y no inciden en ningún tipo de confusión. El protocolo EMV, para el modelo *contactless* y tarjetas VISA, es vulnerable para la configuración que no hace uso de DDA rompiéndose la segunda y tercera propiedad al no coincidir el CVM usado por la tarjeta con el de la terminal y banco. Y, también, para las configuraciones que utilizan transacciones *offline* se rompe la primera y segunda propiedad al no coincidir el AC generado por la tarjeta con el AC recibido por el banco y terminal. Esto es posible de confirmar, ya que mediante la búsqueda que hace Maude-NPA hacia atrás ha sido capaz, a partir de un *attack-pattern,* recorrer cada uno de los estados generados hasta llegar a un estado inicial.

En conclusión, el estudio explica cómo se han llevado a cabo ataques sobre el modelo *contactless* usando distintos métodos para alterar mensajes, mediante las habilidades Dolev-Yao, que permiten modificar el propietario de la tarjeta, o incluso, la transacción en sí. En los últimos apartados, se ha demostrado cómo se puede hacer uso de Maude-NPA para modelar y verificar dichos ataques. Por tanto, gracias a Maude-NPA se han encontrado patrones de ataque a las configuraciones planteadas de una forma realista y concordando con las mismas conclusiones realizadas en la Sección 4.4.3.

6. Conclusiones y trabajos futuros

En este apartado se realiza un resumen el cual menciona las conclusiones que el alumno ha obtenido gracias a los resultados durante la ejecución del proyecto, junto con los distintos posibles trabajos futuros que se podrían aplicar para perfeccionar el trabajo actual.

Una vez finalizado el proyecto, obteniendo como objetivo principal la especificación del protocolo EMV con Maude-NPA, los distintos objetivos que se había propuesto el alumno han sido todos satisfactoriamente completados. Aunque para ello, ha sido necesario realizar distintos procesos de solución ante los diferentes grados de dificultad de cada uno de ellos.

La comprensión y modelado del estudio [4] fue el objetivo principal y uno de los más complicados de realizar, debido a la larga inversión de tiempo que se tuvo que utilizar para comprender cada uno de los ataques que fueron descubiertos y cómo funcionaba su ejecución. Puesto que primero se debía entender qué era el protocolo EMV conjuntamente todas sus configuraciones para poder posteriormente entender sus fallos y por tanto, sus ataques.

Asimismo, los modelos de EMV *contact* y *contactless* son completamente distintos y, aunque el comportamiento final del modelo fuera similar, ataques que funcionaban en uno de ellos no funcionaban en el otro y, viceversa. En consecuencia, problemas de entender cómo una fase y por tanto un ataque podían ser ejecutados en uno de los modelos mientras que no se podía aplicar al otro, han sido nuevos desafíos a los que el alumno ha tenido que encontrar solución durante la especificación.

Del mismo modo, para poder crear un modelo en Maude-NPA se necesitó una previa investigación y distintas pruebas con la herramienta para obtener finalmente el comportamiento deseado de la especificación. Asimismo, Maude-NPA tiene una gran variedad de opciones, que a la hora de recrear alguno de los ataques, se tenía que escoger la forma más plausible para no realizar optimizaciones o saltarse pasos necesarios en los ataques no cumpliendo con la teoría básica y establecida por el protocolo. Antes de obtener un modelo final se plantearon varias ideas erróneas que afortunadamente fueron arregladas con rapidez y eficacia a lo largo del trabajo.

Los dos ataques que se han reproducido correctamente han sido definidos para el modelo *contactless*, por lo que fue necesario añadir y entender los mensajes TTQ y CTQ [16], importantes cuando se usa tecnología NFC [2]. Las vulnerabilidades que existen para este modelo fueron cuidadosamente estudiadas, ya que había que entender al detalle cómo estaba funcionando EMV y en consecuencia saber por qué podía fallar y encontrarse dichas vulnerabilidades.

Finalmente, después de la especificación del trabajo [4] mediante Maude-NPA de una manera elegante y correcta, los resultados obtenidos han coincidido con los esperados y se puede afirmar los descubiertos hechos por el estudio.

En conclusión, se ha demostrado cómo el alumno ha podido resolver todos los problemas planteados ante él, desde el entendimiento de un nuevo protocolo y uso de una nueva herramienta de verificación automática, hasta el modelado de EMV con sus respectivos ataques descubiertos en el *paper* [4] de cierto nivel de prestigio y obteniendo los mismos resultados. Por

todo ello, se concluye que el alumno ha sido capaz de realizar un buen trabajo y obtener unos resultados visibles y correctos.

Por otra parte, aun siendo un trabajo completo y correctamente realizado, el alumno se propone objetivos futuros con el fin de poder perfeccionar el trabajo presente. La especificación del protocolo EMV solo se incluye a dos de los agentes legítimos debido a que no existe la presencia del banco, ya que no afectaba a los resultados de ninguna de las maneras. Por esta razón, crear un nuevo modelo incluyendo al banco puede ser interesante si en el futuro existe la posibilidad de modificar o afectar al comportamiento del EMV de alguna manera.

Además de la posible inserción del banco, se pretende poder hacer una especificación del modelo *contact*, pues existen varios de los ataques presentados en este trabajo que no han sido capaces de ser reproducidos al ser únicamente modelado el modelo *contactless*.

Finalmente, explorar de una forma más amplia todas las posibles combinaciones con las nuevas técnicas que surgen diariamente para conseguir encontrar nuevos ataques que no se hayan ni siquiera planteado o, imposibles de recrear actualmente.

7. Bibliografía

[1] Biurrun A. La Razón [Internet]. España ya prefiere pagar con tarjeta a hacerlo con dinero en efectivo; 26 de octubre de 2021 [consultado el 9 de agosto de 2022]. Disponible en: https://www.larazon.es/tecnologia/20211026/2exixsexo5euvkjctik3kxd3ym.html

[2] VISA [Internet]. Visa Contactless; [consultado el 10 de agosto de 2022]. Disponible en: https://www.visa.es/paga-con-visa/pagos-moviles/visa-contactless.html

[3] EMVCo [Internet]. A Guide to EMV Chip Technology; Noviembre de 2017 [consultado el 8 de junio de 2022]. Disponible en: https://www.emvco.com/wp-content/uploads/documents/A-Guide-to-EMV-Chip-Technology-v3.0-1.pdf

[4] Basin D, Sasse R, Toro-Pozo J. IEEE Xplore [Internet]. The EMV Standard: Break, Fix, Verify; 26 de Agosto de 2021 [consultado el 13 de abril de 2022]. Disponible en: https://doi.org/10.1109/SP40001.2021.00037

[5] Escobar S, Meadows C, Meseguer J. The Maude System [Internet]. Maude-NPA, Version 3.1; 13 de noviembre de 2017 [consultado el 26 de mayo de 2022]. Disponible en: http://maude.cs.illinois.edu/w/images/9/90/Maude-NPA_manual_v3_1.pdf

[6] Murdoch SJ, Dirmer S, Anderson R, Bond M. IEEE Xplore [Internet]. Chip and PIN is Broken; 8 de julio de 2010 [consultado el 20 de julio de 2022]. Disponible en: https://doi.org/10.1109/SP.2010.33

[7] Radu AI, Chothia T, Newton CJ, Boureanu I, Chen L. IEEE Xplore [Internet]. Practical EMV Relay Protection; 27 de julio de 2022 [consultado el 14 de agosto de 2022]. Disponible en: https://doi.org/10.1109/SP46214.2022.9833642

[8] Hajji T, Ouerdi N, Azizi A, Azizi M. ScienceDirect [Internet]. EMV Cards Vulnerabilities Detection Using Deterministic Finite Automaton; 12 de marzo de 2018 [consultado el 14 de agosto de 2022]. Disponible en: https://doi.org/10.1016/j.procs.2018.01.152

[9] Martínez Martín D. Handle Proxy [Internet]. Verificación automática del protocolo DP-3T asociado a las aplicaciones COVID-19; 28 de septiembre de 2021 [consultado el 29 de abril de 2022]. Disponible en: http://hdl.handle.net/10251/173383

[10] Renzo Rondo Van Ysseldyk N. Handle Proxy [Internet]. Verificación automática de protocolos criptográficos de seguridad; 6 de octubre de 2016 [consultado el 29 de abril de 2022]. Disponible en: http://hdl.handle.net/10251/71279

[11] Lluch Palop J. Handle Proxy [Internet]. Verificación automática del protocolo TLS 1.3 usando Maude-NPA; 30 de octubre de 2019 [consultado el 29 de abril de 2022]. Disponible en: http://hdl.handle.net/10251/130041

[12] Clavel M, Durán F, Eker S, Escobar S, Lincoln P, Martí-Oliet N, *et al*. The Maude System [Internet]. Maude Manual; Febrero de 2022 [consultado el 26 de mayo de

2022]. Disponible en: http://maude.cs.illinois.edu/w/images/6/65/Maude-3.2.1-manual.pdf

[13] EMVCo [Internet]. EMV Contactless Specifications for Payment Systems; Marzo de 2021 [consultado el 16 de agosto de 2022]. Disponible en: https://www.emvco.com/wp-content/uploads/documents/C-6_Kernel-6-v2.10.pdf

[14] CCN-CERT [Internet]. ATAQUES DE REPRODUCCIÓN; [consultado el 1 de agosto de 2022]. Disponible en: https://www.ccn-cert.cni.es/publico/seriesCCN-STIC/series/400-Guias_Generales/401-glosario_abreviaturas/index.html?n=97.html

[15] Jain S. Cognitive and Data Scientist [Internet]. Guide to EMV - Contact & Contactless Payments; 18 de mayo de 2016 [consultado el 21 de julio de 2022]. Disponible en: http://srutisj.in/img/GuidetoEMV-Contact-ContactlessPayments.pdf

[16] EFTLab - Breakthrough Payment Technologies [Internet]. The Use of CTQs and TTQs in NFC Transactions; [consultado el 5 de agosto de 2022]. Disponible en: https://www.eftlab.com/the-use-of-ctqs-and-ttqs-in-nfc-transactions/

[17] Würsten F. ETH Zürich [Internet]. Outsmarting the PIN code; 1 de septiembre de 2020 [consultado el 2 de agosto de 2022]. Disponible en: https://ethz.ch/en/news-and-events/eth-news/news/2020/09/outsmarting-the-pin-code.html

[18] Lakshmanan R. The Hacker News [Internet]. New PIN Verification Bypass Flaw Affects Visa Contactless Payments; 7 de septiembre de 2020 [consultado el 2 de agosto de 2022]. Disponible en: https://thehackernews.com/2020/09/emv-payment-card-pin-hacking.html

[19] NIST Computer Security Resource Center | CSRC [Internet]. Message Authentication Code (MAC) algorithm. [consultado el 2 de agosto de 2022]. Disponible en: https://csrc.nist.gov/glossary/term/message_authentication_code_algorithm

8. Anexos

8.1. Anexo 1. Objetivos de desarrollo sostenible

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**				**X**
ODS 4. **Educación de calidad.**				**X**
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**				**X**
ODS 8. **Trabajo decente y crecimiento económico.**				**X**
ODS 9. **Industria, innovación e infraestructuras.**				**X**
ODS 10. **Reducción de las desigualdades**.				**X**
ODS 11. **Ciudades y comunidades sostenibles**.				**X**
ODS 12. **Producción y consumo responsables.**				**X**
ODS 13. **Acción por el clima.**				**X**
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**				**X**

Reflexión sobre la relación del TFG con los ODS y con el/los ODS más relacionados.

En este anexo se trata la relación existente entre el TFG actual y los Objetivos de desarrollo sostenible, aprobados en el año 2015 por la Organización de las Naciones Unidas (ONU), con el objetivo de poder reflexionar si el trabajo cumple con una serie de objetivos globales para erradicar ciertos puntos beneficiosos para la supervivencia humana como es la pobreza, proteger los derechos humanos, el medio ambiente y el sistema económico entre otros.

El trabajo actual hace uso de la herramienta Maude-NPA para modelar el protocolo EMV, el cual usan las tarjetas de crédito cuando se desea hacer una transferencia bancaria, entre el propietario de la tarjeta y el banco, a la hora de pagar por un servicio o producto. Una vez modelado el protocolo, de forma automática, se intenta reproducir mediante una búsqueda inversa varios ataques que afecten a la seguridad del protocolo.

Después de analizar cada uno de los puntos existentes en la tabla anterior compuesta por los ODS, ninguno de los objetivos mencionados ha conseguido ser relacionados por el alumno con el tema del trabajo actual. Esto se debe a que ningún ODS se ve reflejado en las características del TFG, al tratarse de un trabajo, que verifica un protocolo en base a otro estudio previo que también trata el mismo tema sobre las tarjetas de crédito.

El ODS que podría obtener un mayor acercamiento al trabajo es el número ocho, *Trabajo decente y crecimiento económico*. Esto se debe al ser un ODS que tiene como objetivo analizar los riesgos para la economía mundial y poder reducir los distintos cambios bruscos en el sistema económico y financiero. EMV es uno de los protocolos más usados y se busca que sea lo más seguro posible, ya que está ligado directamente a los recursos financieros de las personas. Por lo que, si existiera alguna vulnerabilidad no arreglada, mediante un ataque masivo se podría afectar a miles de países y ciudadanos relativamente rápido. Además, siendo muy costoso en cuanto a tiempo y recursos revertir el problema.

Otro ODS que podría acercarse mínimamente es el número trece, *Acción por el clima*. Este ODS intenta reducir todos los efectos negativos que se producen en el planeta, por ejemplo, el cambio climático. La relación indirecta con el trabajo actual se puede aplicar si se consigue un protocolo EMV sin fallos ni vulnerabilidades dando la completa certeza de poder ser usado sin ningún problema, y así poder evitar la creación de dinero en efectivo. De esta manera, se disminuyen las deforestaciones de bosques, acción que agrava el nivel de dióxido de carbono y, en consecuencia, el cambio climático en el planeta Tierra. Este mismo argumento se puede aplicar al ODS número quince, *Vida de ecosistemas terrestres*, para evitar la destrucción de hábitats naturales que buscan la obtención de un mayor número de recursos para la creación del dinero en efectivo. Por tanto, si se consigue un protocolo EMV seguro se evitaría la extinción de muchas especies salvajes.

En conclusión, ningún ODS parece que encaje directamente con el trabajo, pero de una forma indirecta sí se puede apreciar que algunos de los objetivos pueden cumplirse y se pueden aplicar para la evolución y acercamiento a una situación mayormente sostenible.

8.2. Anexo 2. Ataque Bypassing Cardholder Verification

Fichero maude del ataque *Bypassing Cardholder Verificacion* en Maude-NPA:

```
fmod PROTOCOL-EXAMPLE-SYMBOLS is
--- Importing sorts Msg, Fresh, Public, and GhostData
protecting DEFINITION-PROTOCOL-RULES .

------------------------------------------------------------
--- Overwrite this module with the syntax of your protocol
--- Notes:
--- * Sort Msg and Fresh are special and imported
--- * Every sort must be a subsort of Msg
--- * No sort can be a supersort of Msg
------------------------------------------------------------

--- Sort Information
sorts Name Nonce Key Mac Ttq Ctq Realizado NoRealizado PinOffline PinOnline TC ARQC
CVM PIN AC .
subsort PinOffline PinOnline < PIN .
subsort TC ARQC < AC .
subsort Realizado NoRealizado < CVM .
subsort Name Nonce Key Mac Ttq Ctq PIN CVM AC < Msg .
--- subsort Name < Key .
subsort Name < Public .

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

--- Principals
op c : -> Name . --- Card
op t : -> Name . --- Terminal
op b : -> Name . --- Bank
op i : -> Name . --- Intruder

--- Key shared
op key : Name Name -> Key .

--- Key generada por intruso
op kei : -> Key .

--- Comandos
op read : Name Name -> Msg .
op select : Name Name -> Msg .
op pdolC : Name Name -> Msg .
op pdolT : Name Name Nonce Ttq -> Msg .

--- CTQ y TTQ
op ttq : AC -> Ttq .
op ctq : PIN CVM -> Ctq .
op pan : Name Name -> Msg .

--- Mensajes necesarios
op data : -> Msg .  ---Corresponde con el aip, atc y iad

--- Firmar un mensaje
op sig : Name Msg -> Msg .

--- MAC (Message Authentication Code) de un mensaje
op mac : Key Msg -> Mac .
```

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

endfm

fmod PROTOCOL-EXAMPLE-ALGEBRAIC is
protecting PROTOCOL-EXAMPLE-SYMBOLS .

------------------------------------------------------------
--- Overwrite this module with the algebraic properties
--- of your protocol
------------------------------------------------------------

endfm

fmod PROTOCOL-SPECIFICATION is
protecting PROTOCOL-EXAMPLE-SYMBOLS .
protecting DEFINITION-PROTOCOL-RULES .
protecting DEFINITION-CONSTRAINTS-INPUT .

------------------------------------------------------------
--- Overwrite this module with the strands
--- of your protocol
------------------------------------------------------------

var S : Key .
var TTQ : Ttq .
var CTQ : Ctq .
vars X Y DATA_FALSA PAN : Msg .
var Offline : PinOffline .
var Online : PinOnline .
var Rea : Realizado .
var NoRea : NoRealizado .
var Tipo_TC : TC .
var Tipo_ARQC : ARQC .
var r : Fresh .
vars C T B : Name .
vars UN NC : Nonce .

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), +(mac(kei, X)), nil ] &
:: nil :: [ nil | +(ctq(Offline,Rea)), nil ] &
:: r :: [ nil | +(n(i,r)), nil ] &
:: nil :: [ nil | -(X), +(sig(i, X)), nil ]
[nonexec] .

eq STRANDS-PROTOCOL
= :: r ::
--- TERMINAL
[ nil |       +(select(T,C)),
-(pdolC(C,T)),
+(pdolT(T,C,n(T,r),ttq(Tipo_ARQC))),
-(mac(S, pdolT(T,C,n(T,r),TTQ) ; data) ; data ; CTQ),
+(read(T,C)),
-(pan(C,T)), nil ] &
:: r ::
--- CARD
[ nil |       -(select(T,C)),
+(pdolC(C,T)),
-(pdolT(T,C,UN,TTQ)),
+(mac(key(C,B), pdolT(T,C,UN,TTQ) ; data) ; data ; ctq(Online,NoRea)),
-(read(T,C)),
+(pan(C,T)), nil ]
```


```
[nonexec] .

eq ATTACK-STATE(0)
= :: r ::
--- TERMINAL (Protocolo con intruso crea una CTQ, MITM)
[ nil ,       +(select(t,c)),
-(pdolC(c,t)),
+(pdolT(t,c,n(t,r),ttq(Tipo_ARQC))),
-(mac(S, pdolT(T,C,n(T,r),TTQ) ; data) ; data ; ctq(Offline,Rea)),
+(read(t,c)),
-(pan(c,t)) | nil ]
|| empty
|| nil
|| nil
|| nil
[nonexec] .

endfm

--- THIS HAS TO BE THE LAST LOADED MODULE !!!!
select MAUDE-NPA .
```

Resultados comandos, *summary* e *initials*:

```
reduce in MAUDE-NPA : summary(0, 11) .
rewrites: 15734 in 24ms cpu (45ms real) (655583 rewrites/second)
result Summary: States>> 106 Solutions>> 1
Maude> red initials(0,11).
reduce in MAUDE-NPA : initials(0,11) .
rewrites: 15518 in 28ms cpu (53ms real) (554214 rewrites/second)
result ShortIdSystem: < 1 . 3 . 1 . 5 . 6 . 8 . 4 . 4{2} . 3 . 3 . 2 . 1 > (
:: nil ::
[ nil |
-(data),
-(ctq(#0:PinOffline, #1:Realizado)),
+(data ; ctq(#0:PinOffline, #1:Realizado)), nil]  &
:: nil ::
[ nil |
-(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data)),
-(data ; ctq(#0:PinOffline, #1:Realizado)),
+(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ;
ctq(#0:PinOffline, #1:Realizado)), nil]  &
:: nil ::
[ nil |
-(data ; ctq(#5:PinOnline, #6:NoRealizado)),
+(data), nil]  &
:: nil ::
[ nil |
-(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ;
ctq(#5:PinOnline, #6:NoRealizado)),
+(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data)), nil]  &
:: nil ::
[ nil |
-(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ;
ctq(#5:PinOnline, #6:NoRealizado)),
+(data ; ctq(#5:PinOnline, #6:NoRealizado)), nil]  &
:: #3:Fresh ::
[ nil |
+(select(t, c)),
-(pdolC(c, t)),
+(pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC))),
-(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ;
ctq(#0:PinOffline, #1:Realizado)),
+(read(t, c)),
```

```
-(pan(c, t)), nil]  &
:: #7:Fresh ::
[ nil |
-(select(t, c)),
+(pdolC(c, t)),
-(pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC))),
+(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ;
ctq(#5:PinOnline, #6:NoRealizado)),
-(read(t, c)),
+(pan(c, t)), nil] )
|
data !inI,
read(t, c) !inI,
select(t, c) !inI,
pdolC(c, t) !inI,
ctq(#0:PinOffline, #1:Realizado) !inI,
pan(c, t) !inI,
mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) !inI,
(data ; ctq(#0:PinOffline, #1:Realizado)) !inI,
(data ; ctq(#5:PinOnline, #6:NoRealizado)) !inI,
(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ;
ctq(#0:PinOffline, #1:Realizado)) !inI,
(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ;
ctq(#5:PinOnline, #6:NoRealizado)) !inI,
pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) !inI
|
+(select(t, c)),
-(select(t, c)),
+(pdolC(c, t)),
-(pdolC(c, t)),
+(pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC))),
-(pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC))),
+(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ;
ctq(#5:PinOnline, #6:NoRealizado)),
-(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ;
ctq(#5:PinOnline, #6:NoRealizado)),
+(data ; ctq(#5:PinOnline, #6:NoRealizado)),
-(data ; ctq(#5:PinOnline, #6:NoRealizado)),
+(data),
generatedByIntruder(ctq(#0:PinOffline, #1:Realizado)),
-(data),
-(ctq(#0:PinOffline, #1:Realizado)),
+(data ; ctq(#0:PinOffline, #1:Realizado)),
-(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ;
ctq(#5:PinOnline, #6:NoRealizado)),
+(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data)),
-(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data)),
-(data ; ctq(#0:PinOffline, #1:Realizado)),
+(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ;
ctq(#0:PinOffline, #1:Realizado)),
-(mac(key(c, #2:Name), pdolT(t, c, n(t, #3:Fresh), ttq(#4:ARQC)) ; data) ; data ;
ctq(#0:PinOffline, #1:Realizado)),
+(read(t, c)),
-(read(t, c)),
+(pan(c, t)),
-(pan(c, t))
|
nil
```

8.3. Anexo 3. Ataque Unauthenticated Offline Transactions

Fichero maude del ataque *Unauthenticated Offline Transactions* en Maude-NPA:

```
fmod PROTOCOL-EXAMPLE-SYMBOLS is
--- Importing sorts Msg, Fresh, Public, and GhostData
protecting DEFINITION-PROTOCOL-RULES .

------------------------------------------------------------
--- Overwrite this module with the syntax of your protocol
--- Notes:
--- * Sort Msg and Fresh are special and imported
--- * Every sort must be a subsort of Msg
--- * No sort can be a supersort of Msg
------------------------------------------------------------

--- Sort Information
sorts Name Nonce Key Mac Ttq Ctq Realizado NoRealizado PinOffline PinOnline TC ARQC
CVM PIN AC .
subsort PinOffline PinOnline < PIN .
subsort TC ARQC < AC .
subsort Realizado NoRealizado < CVM .
subsort Name Nonce Key Mac Ttq Ctq PIN CVM AC < Msg .
subsort Name < Public .

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

--- Principals
op c : -> Name . --- Card
op t : -> Name . --- Terminal
op b : -> Name . --- Bank
op i : -> Name . --- Intruder

--- Key shared
op key : Name Name -> Key .

--- Key generada por intruso
op kei : -> Key .

--- Comandos
op read : Name Name -> Msg .
op select : Name Name -> Msg .
op pdolC : Name Name -> Msg .
op pdolT : Name Name Nonce Ttq -> Msg .

--- CTQ y TTQ
op ttq : AC -> Ttq .
op ctq : PIN CVM -> Ctq .
op pan : Name Name -> Msg .

--- Mensajes necesarios
op data : -> Msg .  ---Corresponde con el aip, atc y iad

--- Firmar un mensaje
op sig : Name Msg -> Msg .

--- MAC (Message Authentication Code) de un mensaje
op mac : Key Msg -> Mac .

--- Associativity operator
```

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

endfm

fmod PROTOCOL-EXAMPLE-ALGEBRAIC is
protecting PROTOCOL-EXAMPLE-SYMBOLS .

endfm

fmod PROTOCOL-SPECIFICATION is
protecting PROTOCOL-EXAMPLE-SYMBOLS .
protecting DEFINITION-PROTOCOL-RULES .
protecting DEFINITION-CONSTRAINTS-INPUT .

------------------------------------------------------------
--- Overwrite this module with the strands
--- of your protocol
------------------------------------------------------------

var S : Key .
var TTQ : Ttq .
var CTQ : Ctq .
vars X Y DATA_FALSA PAN : Msg .
var Offline : PinOffline .
var Online : PinOnline .
var Rea : Realizado .
var NoRea : NoRealizado .
var Tipo_TC : TC .
var Tipo_ARQC : ARQC .
var r : Fresh .
vars C T B : Name .
vars UN NC : Nonce .

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), +(mac(kei, X)), nil ] &
:: nil :: [ nil | +(ctq(Offline,Rea)), nil ] &
:: r :: [ nil | +(n(i,r)), nil ] &
:: nil :: [ nil | -(X), +(sig(i, X)), nil ]
[nonexec] .

eq STRANDS-PROTOCOL
= :: r ::
--- TERMINAL
[ nil |       +(select(T,C)),
-(pdolC(C,T)),
+(pdolT(T,C,n(T,r),ttq(Tipo_TC))),
-(mac(S, pdolT(T,C,n(T,r),TTQ) ; data) ; data ; CTQ),
+(read(T,C)),
-(sig(C, n(T,r) ; NC ; CTQ) ; NC ; pan(C,T)), nil ] &

:: r ::
--- CARD
[ nil |       -(select(T,C)),
+(pdolC(C,T)),
-(pdolT(T,C,UN,TTQ)),
+(mac(key(C,B), pdolT(T,C,UN,TTQ) ; data) ; data ; ctq(Offline,Rea)),
-(read(T,C)),
+(sig(C, UN ; n(C,r) ; ctq(Offline,Rea)) ; n(C,r) ; pan(C,T)), nil ]
[nonexec] .

eq ATTACK-STATE(0)
= :: r ::
--- TERMINAL (Protocolo con intruso crea una mac falsa, MITM)
```

```
[ nil ,       +(select(t,c)),
-(pdolC(c,t)),
+(pdolT(t,c,n(t,r),ttq(Tipo_TC))),
-(mac(kei, DATA_FALSA) ; data ; CTQ),
+(read(t,c)),
-(sig(c, n(t,r) ; NC ; CTQ) ; NC ; pan(c,t)) | nil ]
|| empty
|| nil
|| nil
|| nil
[nonexec] .

endfm

--- THIS HAS TO BE THE LAST LOADED MODULE !!!!
select MAUDE-NPA .
```

Resultados comandos, *summary* e *initials*:

```
Maude> red summary(0,8).
reduce in MAUDE-NPA : summary(0, 8) .
rewrites: 8152 in 4ms cpu (5ms real) (2038000 rewrites/second)
result Summary: States>> 112 Solutions>> 1
Maude> red initials(0,8).
reduce in MAUDE-NPA : initials(0,8) .
rewrites: 7924 in 4ms cpu (5ms real) (1981000 rewrites/second)
result ShortIdSystem: < 1 . 5 . 1 . 5 . 3 . 3 . 1 . 2 . 1 > (
:: nil ::
[ nil |
-(mac(kei, #0:Msg)),
-(data ; ctq(#1:PinOffline, #2:Realizado)),
+(mac(kei, #0:Msg) ; data ; ctq(#1:PinOffline, #2:Realizado)), nil]  &
:: nil ::
[ nil |
-(mac(key(c, #3:Name), pdolT(t, c, n(t, #4:Fresh), ttq(#5:TC)) ; data) ; data ;
ctq(#1:PinOffline, #2:Realizado)),
+(data ; ctq(#1:PinOffline, #2:Realizado)), nil]  &
:: #4:Fresh ::
[ nil |
+(select(t, c)),
-(pdolC(c, t)),
+(pdolT(t, c, n(t, #4:Fresh), ttq(#5:TC))),
-(mac(kei, #0:Msg) ; data ; ctq(#1:PinOffline, #2:Realizado)),
+(read(t, c)),
-(sig(c, n(t, #4:Fresh) ; n(c, #6:Fresh) ; ctq(#1:PinOffline, #2:Realizado)) ; n(c,
#6:Fresh) ; pan(c, t)), nil]  &
:: #6:Fresh ::
[ nil |
-(select(t, c)),
+(pdolC(c, t)),
-(pdolT(t, c, n(t, #4:Fresh), ttq(#5:TC))),
+(mac(key(c, #3:Name), pdolT(t, c, n(t, #4:Fresh), ttq(#5:TC)) ; data) ; data ;
ctq(#1:PinOffline, #2:Realizado)),
-(read(t, c)),
+(sig(c, n(t, #4:Fresh) ; n(c, #6:Fresh) ; ctq(#1:PinOffline, #2:Realizado)) ; n(c,
#6:Fresh) ; pan(c, t)), nil] )
|
read(t, c) !inI,
select(t, c) !inI,
pdolC(c, t) !inI,
mac(kei, #0:Msg) !inI,
(data ; ctq(#1:PinOffline, #2:Realizado)) !inI,
(sig(c, n(t, #4:Fresh) ; n(c, #6:Fresh) ; ctq(#1:PinOffline, #2:Realizado)) ; n(c,
#6:Fresh) ; pan(c, t)) !inI,
```

```
(mac(kei, #0:Msg) ; data ; ctq(#1:PinOffline, #2:Realizado)) !inI,
(mac(key(c, #3:Name), pdolT(t, c, n(t, #4:Fresh), ttq(#5:TC)) ; data) ; data ;
ctq(#1:PinOffline, #2:Realizado)) !inI,
pdolT(t, c, n(t, #4:Fresh), ttq(#5:TC)) !inI
|
+(select(t, c)),
-(select(t, c)),
+(pdolC(c, t)),
-(pdolC(c, t)),
+(pdolT(t, c, n(t, #4:Fresh), ttq(#5:TC))),
-(pdolT(t, c, n(t, #4:Fresh), ttq(#5:TC))),
+(mac(key(c, #3:Name), pdolT(t, c, n(t, #4:Fresh), ttq(#5:TC)) ; data) ; data ;
ctq(#1:PinOffline, #2:Realizado)),
-(mac(key(c, #3:Name), pdolT(t, c, n(t, #4:Fresh), ttq(#5:TC)) ; data) ; data ;
ctq(#1:PinOffline, #2:Realizado)),
+(data ; ctq(#1:PinOffline, #2:Realizado)),
generatedByIntruder(mac(kei, #0:Msg)),
-(mac(kei, #0:Msg)),
-(data ; ctq(#1:PinOffline, #2:Realizado)),
+(mac(kei, #0:Msg) ; data ; ctq(#1:PinOffline, #2:Realizado)),
-(mac(kei, #0:Msg) ; data ; ctq(#1:PinOffline, #2:Realizado)),
+(read(t, c)),
-(read(t, c)),
+(sig(c, n(t, #4:Fresh) ; n(c, #6:Fresh) ; ctq(#1:PinOffline, #2:Realizado)) ; n(c,
#6:Fresh) ; pan(c, t)),
-(sig(c, n(t, #4:Fresh) ; n(c, #6:Fresh) ; ctq(#1:PinOffline, #2:Realizado)) ; n(c,
#6:Fresh) ; pan(c, t))
|
nil
```