Universidad
Rey Juan Carlos

Escuela Técnica Superior de Ingeniería de Telecomunicación

Grado en Ingeniería en Telemática

Trabajo Fin de Grado

Utilización de Macaroons para el acceso a recursos en una red Named Data Network(NDN)

Autor: Daniel Juanes Quintana
Tutora: Eva M. Castro Barbero

Curso Académico 2016 - 2017

Trabajo Fin de Grado
Utilización de Macaroons para el acceso a recursos en una red NDN

Autor
Daniel Juanes Quintana

Tutora
Eva M. Castro Barbero

La defensa del presente Trabajo Fin de Grado se realizó el día de de 2017, siendo evaluado por el siguiente tribunal:

Presidente:

Vocal:

Secretario:

y habiendo obtenido la siguiente Calificación:

Fuenlabrada, a de de 2017

Agradecimientos

Me gustaría aprovechar para darle las gracias a mis padres y también a mi familia por el apoyo que me han dado durante estos años. También a mis amigos, tanto los de toda la vida como los nuevos que he conocido a lo largo de la carrera. Sin olvidar a mis compañeros de trabajo SnowZone que me han ayudado cambiándome turnos para que pudiera asistir a clase y poder estudiar. Y por último darle las gracias a mi tutora por la ayuda y la paciencia que ha tenido conmigo durante todo el trabajo de fin de grado.

Contenido

Índice de figuras

Acrónimos

Access Controller - AC
Content Centric Networking - CCN
Content Store - CS
Data-Singning-Key - DSK
Discharge Macaroon - DM
Domain Named System - DNS
First Party Caveat - FPC
File Transfer Protocol - FTP
Forwarding Information Base - FIB
Group Keys Distributor - GKD
Identificador de recursos uniforme - URI
Internet Assigned Numbers Authority - IANA
Internet Protocol - IP
Key-Singning-Key - KSK
Keyed-Hash Message Authentication Code - HMAC
Named Data Network - NDN
Network Address Translation - NAT
Networking Forwarding Daemon - NFD
Pending Interest Table - PIT
Rivest, Shamir y Adleman - RSA
Simple Network Management Protocol - SNMP
Third Party Caveat - TPC
Transmision Control Protocol/ Internet Protocol - TCP/IP
Transport Control Protocol - TCP
Trivial File Transfer Protocol - TFTP
User Datagram Protocol - UDP

Resumen

El propósito de este trabajo de fin de grado es la creación de un sistema de acceso a recursos a través de Macaroons de manera confidencial, segura y fiable sobre una red Name Data Network. Para ello el sistema desarrollado separa las funciones como la publicación de datos, control de acceso y autenticación entre diferentes entidades. Esta división de funciones entre diferentes entidades se hace para agilizar el intercambio de información, así como para mejorar la seguridad del sistema evitando que las entidades tengan una visión global de éste.

El sistema está compuesto por tres entidades: una entidad productora, una entidad de acceso de control y otra entidad de autenticación. La entidad productora publicará tanto los datos cifrados con la clave de datos, así como la clave de datos cifrada con una clave de grupo. Sólo las entidades que tengan acceso a la clave de grupo podrán conseguir la clave con la que están cifrados los datos. Un consumidor necesitará pedir a la entidad de control de acceso una acreditación que será un Macaroon específico para cada grupo. El Macaroon impondrá una serie de restricciones para que el poseedor pueda utilizarlo para acceder a los datos publicados. Entre esas restricciones existirá la condición de que el poseedor del Macaroon debe autentificarse en una tercera entidad, la entidad de autenticación. Si se autentica, obtendrá una prueba de que el poseedor del Macaroon cumple dicha restricción, su pertenencia al grupo, y podrá conseguir la clave de grupo que le permite acceder a los datos.

El sistema está diseñado para que el productor no necesite conocer quién le solicita los datos, la entidad de control de acceso no conozca quién solicita el Macaroon ni los datos que se solicitan al productor, y por último, la entidad de autenticación no necesite saber a los datos que se quiere acceder.

Capítulo 1
Introducción

Este proyecto se desarrolla sobre la arquitectura de red NDN[1] y utiliza un sistema de autorización descentralizado proporcionado por los Macaroons. En este capítulo se describe el entorno de red NDN y se explica el funcionamiento de los Macaroons.

1.1. Arquitectura de red de Internet

La arquitectura de red más extendida actualmente es Transmision Control Protocol/Internet Protocol (TCP/IP)[2]. Esta arquitectura TCP/IP define un conjunto de reglas y de protocolos de comunicación para Internet, para poder establecer comunicación.

TCP/IP está diseñada por niveles o capas, este sistema está estandarizado para poder establecer una comunicación independientemente del SO, y para ello cada capa lleva a cabo una tarea específica.

1.1.1. Arquitectura de red TCP/IP

Los distintos protocolos incluidos en TCP/IP dividen el intercambio de información entre una máquina emisora y una receptora en cuatro capas o niveles diferentes. Las capas están diseñadas para facilitar la comunicación entre varias máquinas, siendo cada una de ellas una parte del proceso de comunicación. Las capas en la que se divide este modelo son:

- Capa de Aplicación: los protocolos de esta capa proporcionan la interfaz de usuario necesaria para que las aplicaciones puedan acceder a la red, gestionando la transferencia de archivos, servicio de correo, control de la red, etc... . Residen en esta capa protocolos como File Transfer Protocol (FTP), Trivial File Transfer Protocol (TFTP), TELNET, Simple Network Management Protocol(SNMP), Simple Mail Transfer Protocol (SMTP) y Domain Named System (DNS).

- Capa de transporte: esta capa proporciona los protocolos necesarios para controlar el flujo y la fiabilidad de la conexión, para efectuar el transporte de datos entre varias máquinas extremo a extremo. Se han definido fundamentalmente dos protocolos, que son:

  Transport Control Protocol (TCP): es un protocolo orientado a conexión con control de errores, creando una confiabilidad de transferencia de datos, monitorizando la entrega y recepción de información a pesar de que la red a través de la se crea la conexión no sea fiable. TCP se encarga de retransmitir los paquetes en caso de que no hayan llegado a su destino o en caso de que haya ocurrido algún error en el envío de uno de ellos.

  User Datagram Protocol (UDP): es un protocolo no orientado a conexión, no garantiza un envío fiable, así como que los paquetes lleguen en el orden en el que han sido enviados, dejando la responsabilidad de ordenarlos o comprobar la integridad de los paquetes a capas superiores en el lado del receptor.

- Capa de Internet: es la encargada de encaminar los datos a través de la red. El protocolo más importante de esta capa es Internet Protocol (IP). Es un protocolo no fiable, basado en datagramas. Identifica a las máquinas receptoras de la información a través de un número de 32 bits, la dirección IP. Cada máquina tiene asignada una dirección IP diferente.

- Capa de Acceso a la Red: esta capa es la encargada de adaptar los datagramas de la capa anterior al medio físico, a través de tramas para ser enviadas por el entorno físico de la red en forma de bits. Uno de los protocolos más habituales es Ethernet. Esta capa utiliza su propio direccionamiento físico para hacer llegar los paquetes a máquinas directamente conectadas al mismo medio físico.

1.1.2. Internet Protocol

IPv4[3] es el protocolo utilizado por la capa de Internet, siendo uno de los más importantes. Este protocolo encamina el paquete o datagrama, sin garantizar que éste alcance el destino. Este protocolo no ofrece ninguna seguridad, ni fiabilidad de que el paquete llegue a su destino o llegue dañado, la comprobación de este tipo de problemas se lo deja a los protocolos que se encuentran en capas superiores.

El paquete o datagrama generado por este protocolo, tendrá una cabecera IP que contendrá la dirección de destino y la de origen, que se denominan direcciones IP. Estas direcciones IP son direcciones lógicas, que son un identificador único formado por 4 bytes, que permitirá identificar a una interfaz de red de una máquina. Por

ello las direcciones no se podrán repetir. Parte de la dirección se encargará de identificar la red a la que pertenece la máquina y la otra parte identificará la máquina. Las direcciones IP se dividen en dos tipos, las públicas y las privadas. Las direcciones públicas serán asignadas por Internet Assigned Numbers Authority (IANA) para evitar que puedan existir varias máquinas con una misma dirección, produciendo un conflicto en la red. Las direcciones privadas están destinadas a las redes locales, es decir, principalmente casas y empresas. Para que el tráfico de una red privada pueda salir a Internet, necesitará un dirección IP pública. Se usará el protocolo Network Address Translation (NAT) que convertirá direcciones IP privadas en públicas o viceversa, esto permitirá comunicar máquinas entre diferentes redes privadas, a través de una red pública.

Las direcciones IP se utilizan para encaminar los paquetes o datagramas IP a través de la red, pudiendo atravesar múltiples nodos intermedios. Los nodos intermedios poseen una tabla de encaminamiento. Esta tabla es utilizada para decidir cuál será el camino que lleve el paquete a la dirección IP de destino. Las tablas se actualizan según el protocolo de encaminamiento que usen. Los protocolos de encaminamiento utilizan diferentes métricas para calcular las rutas de encaminamiento, basándose en vectores de distancia, estado de enlace, o bien rutas introducidas manualmente.

En la actualidad, existe una nueva versión del protocolo IP, IPv6[4]. Las direcciones IP en esta nueva versión pasan de 4 bytes a 16 bytes. IPv6 permite tener mayor cantidad de direcciones, haciendo que sea muy difícil que se agoten a corto plazo.

1.2. Named Data Networking (NDN)

NDN es una nueva arquitectura de red que se centra en los datos que se quieren transmitir a través de la red, es decir, esta arquitectura se basa en el contenido en sí y no en la entidad que posee los datos, siendo capaz de localizarlos a través de nombres únicos.Cuando se solicite un dato se realiza una petición con un paquete que es denominado Interest y que contendrá el nombre de los datos solicitados, así como otra clase de información. Esta petición será respondida con otra clase de paquete, la cuál será denominada Data y que contiene los datos solicitados. La entidad que solicite un dato la llamaremos Consumidor[6] y la entidad que proporciona el dato la llamaremos Productor.

Esta arquitectura de red surgió de un proyecto llamado Content Centric Networking(CCN) presentado en 2006 por Jacobson. Básicamente CCN y NDN quieren presentar una evolución de la arquitectura de red actual TCP/IP, para dejar de centrarse en las entidades o máquinas que contienen los datos, para

centrarse en los propios datos, que tendrían que ser la parte importante de cualquier arquitectura de red.

NDN nos propone una evolución de TCP/IP eliminando el direccionamiento IP, que identifica las máquinas, y que se necesita para establecer una comunicación extremo a extremo. NDN utilizará nombres, los cuáles no identificarán máquinas o entidades únicamente, sino que pueden identificar cualquier cosa, como cualquier tipo de datos (vídeo, audio, libros, etc...), así como identificar una orden para ejecutar en un dispositivo. Siendo los nombres una parte importante de esta arquitectura de red, ya que nos permiten identificar cualquier dato sin importar su localización, dejan de tener sentido las redes con direccionamiento IP público y privado. Con este tipo de redes los datos pueden alojarse en diferentes máquinas, sin necesidad de que el dato esté ligado a la máquina que lo contiene.

Los nombres no serán la única mejora respecto a TCP/IP, ya que NDN proporciona mayor nivel de confianza, asegurándonos que los datos recibidos no han sido modificados por una tercera parte y son confiables y seguros. Esto obliga al creador del paquete Data a firmarlo, teniendo el receptor que validar dicho paquete y comprobar que el paquete proviene de una entidad o máquina en la que confía y que el dato no ha sido modificado por el camino.

1.2.1. Formato de paquete

Los paquetes necesarios para establecer una comunicación entre productor y un consumidor, son denominados Interest y Data vease figura 1.1[1]. El consumidor el cuál quiere realizar un petición a través de un nombre, lo hará enviando un Interest y el productor responderá con un paquete Data, con los datos pedidos por el Consumidor.

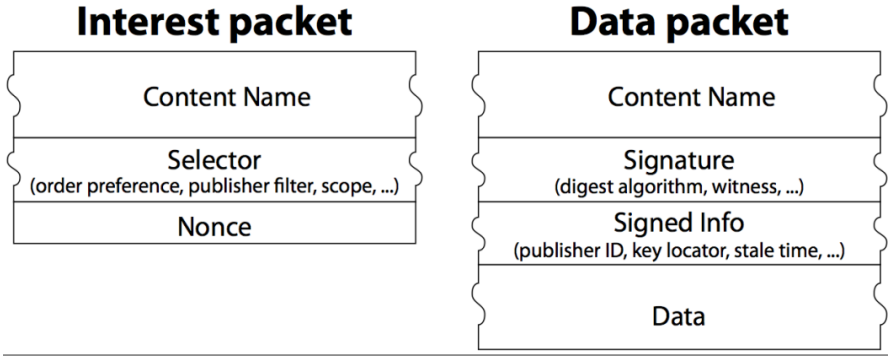

Figura 1.1. Formato de los paquetes Interest y Data

[1]Figura tomada de http://name-data.net

A continuación se explican los campos mostrados en la figura 1.1 de los paquetes Interest:

- Content Name: nombre de los datos que se solicitan en un paquete Interest. Este nombre estará formado por un identificador de recursos uniforme o URI. Un consumidor indicará el nombre del dato o acción que quiere solicitar con el Interest.

- Signature: la firma del paquete se utiliza para comprobar la integridad del paquete Interest. Permite detectar si una entidad que no sea el consumidor ha modificado el paquete Interest. Si el paquete es modificado por otra entidad y la firma no coincide, el paquete Interest deja de ser confiable. Otra propiedad de la firma, es que permite saber quién es el creador del paquete Interest, permitiendo al productor saber si el consumidor es de confianza.

- Selectors: El selector está compuesto por diversos campos, que actuarán para poder establecer ciertas características del paquete.

  — MinSuffixComponents, MaxSuffixComponents: permite obtener información del nombre si está o no completo, así como el número de componentes del que está compuesto el nombre.

  — PublisherPublicKeyLocator: si el paquete va firmado y no se dispone de la clave pública necesaria para verificar la firma, éste indicará el nombre donde encontrar la clave pública necesaria.

  — MustBeFresh: indicará si el dato puede ser obtenido de una caché o es necesario pedírselo al nodo creador del Data. Si éste es 0, se debe pedir directamente al productor que contenga el recurso que estamos solicitando. En caso contrario, dependiendo del valor un router podría devolver un Data válido que puede tener ya almacenado en su caché.

  — Exclude: este campo opcional actúa como filtro, permitiendo excluir paquetes Data que puedan contener routers intermedios en sus cachés. Esta información puede incluir versión del paquete, evitando así las que no se desean. También puede indicar quien ha creado el paquete, pudiendo tener varias entidades los mismos datos y evitar obtener los datos de una o varias entidades en concreto. Existen otra clase de filtros en función de paquetes Data que se deseen evitar, si se está buscando uno en concreto.

- Nonce: identificador único, generado de manera aleatoria y con una longitud de 4 bytes. Es utilizado principalmente por los routers para detectar posibles bucles en la red.

- Guiders: Su principal campo será el tiempo de vida que tendrá el Interest.

Los campos de los paquetes Data mostrados en la figura 1.1 son los siguientes:

- Content Name: nombre de los datos o acciones solicitadas, obtenido del campo Content Name del paquete Interest. Puede ser modificado añadiendo componentes al final del nombre para incluir información sobre la petición solicitada.
- Signature: es la firma del paquete, se utiliza para comprobar la integridad del paquete Data. Evita que una entidad que no sea el creador del Data modifique cualquiera de los campos del paquete. La firma es utilizada por el consumidor para identificar al productor del paquete Data, y establecer si el productor es una entidad confiable o no.
- Signed Info: contendrá una serie de valores, que darán información sobre el contenido del paquete. Por ejemplo, los datos solicitados se corresponden con el nombre, los datos solicitados tienen otro nombre válido, se transporta una clave pública, si es un Data de información solo para routers vecinos, así como un rango de valores más amplios aún sin asignar. Este apartado contendrá el tiempo de validez del paquete. Si los datos solicitados son demasiado grandes para ser enviados en un único paquete Data, éste indicará el número final de bloques en que está dividido.
- Data: este componente contendrá lo datos que se quieren transmitir a través de la red.

1.2.2. Encaminamiento de paquetes en NDN[7]

Los routers están formados por tres estructuras de datos, que se explican a continuación(Figura 1.2):

- Pending Interest Table(PIT): tabla donde se almacenarán los Interest pendientes de recibir los Datas. Los Interest de la tabla se irán eliminando según vayan llegando los Data que satisfacen dichos Interest, expiren los Interest o sea imposible llegar al Data desde el router.
- Content Store(CS): tabla donde se almacenarán los Data recibidos. Los data se irán eliminando según vayan expirando el tiempo de validez de dichos Data, o se cumple el tiempo máximo establecido por el propio router para mantener un Data en la caché.
- Forwarding Information Base(FIB): esta estructura contiene la información necesaria para poder procesar un Interest, es decir, a partir de esta información se decidirá por que interfaz saldrá el Interest para llegar hasta los datos de la manera más eficiente. Contendrá las rutas y estrategias encaminamiento de paquetes, además de la información recibida por routers vecinos que permitan establecer las rutas más eficientes.

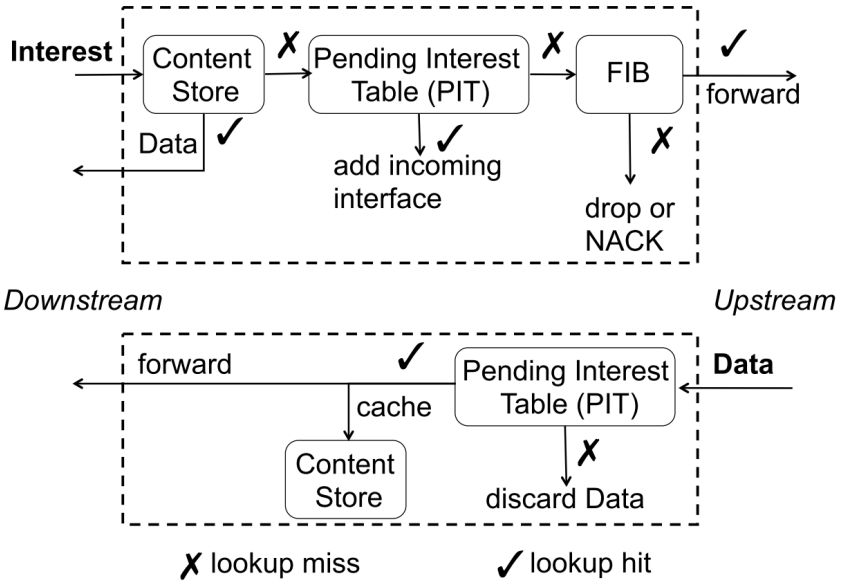

Figura 1.2. Diagrama de funcionamiento de un Router NDN

Estas tres estructuras conformarán el sistema interno del router. En la figura 1.2 se muestra el camino que sigue un paquete Interest y un paquete Data través de un router de una red NDN. Cuando se recibe un Interest, se realiza una comprobación de la caché (CS), necesario para saber si el Data solicitado se encuentra en ella. En caso de que se posea el Data referente al Interest recibido, el paquete será enviado por la interfaz por la que se ha recibido la petición. En el caso de que CS no contenga dicha petición, se revisará el PIT, donde el router comprobará si la petición ya ha sido hecha por otro consumidor. Si éste es el caso simplemente modificará la tabla añadiendo la interfaz de entrada de la petición, si no añadirá tanto la interfaz de entrada como el Interest en la tabla. Si al comprobar el NONCE del Interest, se observa que ya se había almacenado un Interest con ese NONCE por un puerto diferente de entrada, el paquete será descartado. Si un Interest es almacenado por primera vez en el PIT, el paquete pasará seguidamente al FIB, que utilizará la información contenida en él para establecer cuál será la interfaz de salida más adecuada.

En la figura 1.2 se aprecia la recepción en el router de un paquete Data. Se comprobara el PIT en primer lugar, para revisar si se encuentra el Interest correspondiente al paquete recibido. Si no se encuentra, directamente será descartado, ya que se ha recibido un Data que ha sido solicitado previamente con un paquete Interest. En caso contrario, pasará a ser almacenado en CS, y difundido por las interfaces que hayan solicitado dicho Data. Nótese que los Data viajan por el mismo camino que han seguido los Interest.

Los routers NDN pueden usar los mismos algoritmos utilizados por los protocolos de encaminamiento de TCP/IP, utilizando vectores de distancia, estado del enlace, costes, etc... . NDN utilizará los componentes incluidos en el nombre para poder resolver la ruta y encontrar los datos solicitados. El FIB es el encargado de gestionar toda la información necesaria para poder tramitar los Interest y establecer rutas eficientes, así como de informar a los routers vecinos de rutas inalcanzables. El router irá aprendiendo y adaptándose a los cambios que se puedan producir en la propia red.

Una de las ventajas de NDN, es la utilización del CS para almacenar paquetes Data, creando un sistema interno de almacenaje en la red, una gran caché. Cada router o nodo conservará los Data durante un límite de tiempo para satisfacer futuras peticiones para ese Data. Este mecanismo es beneficioso para la red, ya que el número de saltos que habrá en la ruta hasta alcanzar el Data solicitado puede variar en función de si algún router intermedio lo tiene o no, sin llegar en muchos casos a volver a realizar la misma petición Interest hasta el productor.

1.2.3. Seguridad

La seguridad es uno de los puntos fuertes de NDN. TCP/IP deja la responsabilidad de la seguridad a otros protocolos externos a IP, teniendo que decidir ellos si los datos son fiables o no. NDN obliga al creador del Data a firmarlo para que sea posible comprobar su integridad, así como la confianza que tenemos en el usuario que ha creado el paquete.

Los paquetes Data serán firmados obligatoriamente por el creador de dicho paquete. Para ello cada entidad o máquina poseerá una serie de claves Rivest, Shamir y Adleman (RSA), compuesta por una clave privada a la que solo tendrá acceso el propio creador de las claves, y otra pública, la cuál estará disponible a través de un certificado para todo el mundo que la necesite. El certificado de la clave pública podrá estar firmado por una autoridad confiable, conformando un sistema de confianza jerárquico. La firma de los paquetes tanto Interest como Data se hará con una clave privada, debiendo ser verificada con la clave pública tanto por el consumidor en el caso del Data como por el productor en el caso que el Interest lo requiera. Si el destinatario del paquete no posee el certificado necesario para validar la firma, el propio paquete en la sección KeyLocator le indicará el nombre de la clave para poder solicitarla. Al estar basado el sistema de seguridad en un modelo jerárquico, será la entidad receptora del paquete la encargada de establecer una comprobación de los certificados que sean necesarios para poder decidir si un paquete proviene de una entidad confiable.

Las claves tendrán un nombre único, que estará formado por un prefijo. El prefijo es denominado identidad. La identidad constará de dos tipos de

claves: Key-Singning-Key (KSK)[10] y Data-Signing-Key(DSK). Dichas claves se utilizarán para diferentes funciones. Aunque el uso de cada una de ellas dependerá del que la aplicación quiera darle, en general, la clave KSK firmará Interest u otras claves. DSK firmará los Data, para asegurar su integridad. El nombre de ambas claves estará formado por la identidad, más un identificativo que las hará únicas, es decir, se añadirá a la identidad un componente formado por las siglas de la clave, KSK o DSK, más una secuencia de tiempo. Esta secuencia a su vez actuará como el tiempo de vida de las claves, siendo el de KSK mayor que el DSK. Ambas claves dispondrán de un certificado, necesario para compartir la clave pública. El certificado podrá ser firmado por una autoridad confiable que valide esa clave pública asegurando la autenticidad e integridad de ese certificado.

1.3. Ventajas de NDN respecto TCP/IP

Una de las principales cualidades de NDN es la integridad y confiabilidad de los datos, ya que obliga al creador del paquete a firmarlo, pudiendo ser verificado por el consumidor y decidir si la entidad que ha firmado el paquete es confiable o no. TCP/IP no firma el paquete, y el consumidor tiene la responsabilidad de algún modo externo de comprobar que los datos son fiables. NDN proporciona una forma de asegurar la integridad de los datos a la hora de enviar paquetes, entre los consumidores y los productores, pudiendo evitar que una entidad intermedia pueda modificarlos sin detectarse.

El uso de nombres por parte de NDN, en vez de utilizar direcciones IP, permite que haya infinitos nombres, haciendo innecesarias organizaciones como IANA para la asignación de estos, así como la imposibilidad de que se acaben. Esto a su vez, hace que dejen de existir las direcciones y redes, públicas y privadas, por lo que el protocolo NAT no es necesario. Los nombres permiten la independencia de la localización del recurso, es decir, se podría mover el recurso o la máquina, a cualquier parte sin necesidad de tener que modificar el nombre.

En cierto modo, el sistema de capas deja de tener sentido en NDN, ya que el propio nombre puede identificar la aplicación, puerto, recurso y toda la información necesaria, es decir, si disponemos del siguiente nombre, /urjc/video/presentacion.avi/1/2, el nombre indica que la aplicación es de vídeo, el nombre del fichero es presentacion.avi, la parte 1 del vídeo y la versión 2.

En la Figura 1.3[2] se muestra una comparación entre la arquitectura de red de NDN y TCP/IP. La parte de la izquierda es la arquitectura TCP/IP y la parte de la derecha es la arquitectura NDN. La figura muestra que la arquitectura NDN

[2]Figura tomada de http://name-data.net

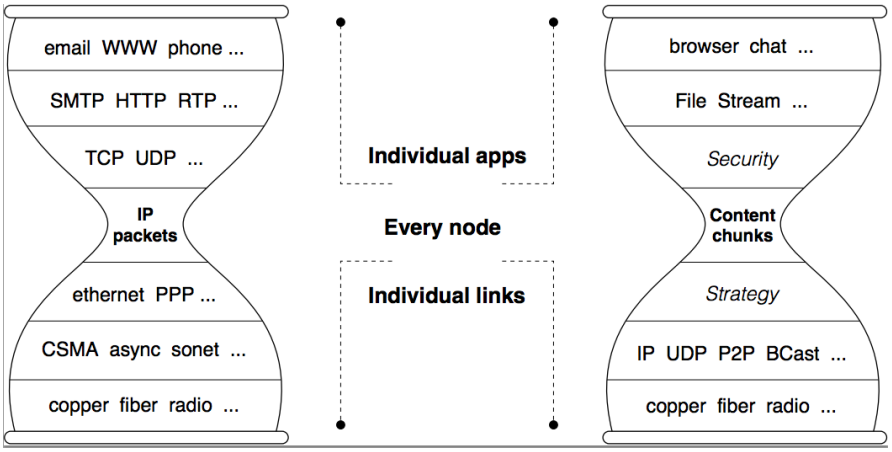

Figura 1.3. Comparación entre arquitectura NDN y TCP/IP

añade la seguridad a las capas superiores, y deja la estrategia de encaminamiento por encima de IP, permitiendo cualquier protocolo de transporte en las capas más bajas.

1.4. Macaroon

Macaroon[12] surge de la necesidad de compartir el control de acceso a recursos en sistemas distribuidos, para actualizar los mecanismos que actualmente tienen algunos servicios implementados en la Nube, la Web u otros sistemas. Macaroon es una credencial de autorización, que permite descentralizar el sistema de autorización, distribuyendo el control entre diferentes entidades, la cuáles poseerán los Macaroons con los que delegar permisos.

Los Macaroons fueron diseñados pensando en cualquier servicio o sistemas distribuidos, que de manera segura necesitan compartir información. La seguridad y el acceso es un punto crítico a la hora de autentificarse, debido a que la mayoría de estos sistemas comparten información privada o crítica.

Aunque los Macaroons son portadores de credenciales, como son las Cookies, los Macaroons poseen cabeceras para atenuar o delimitar las credenciales del Macaroon, definiendo cuándo, donde, por quién y para qué propósito debe autorizar la petición del servicio. Para que la petición sea realizada con éxito, el portador deberá cumplir todas las condiciones o predicados incluidos en el Macaroon. Para asegurar su integridad, utilizará keyed-hash message authentication code (HMAC), siendo una firma que evitará que alguna de las condiciones sea modificada por el portador o por una tercera entidad maliciosa.

Una entidad portadora de un Macaroon puede entregar el Macaroon a otra entidad. La entidad portadora puede dárselo sin atenuar o atenuado. Si el

Macaroon está atenuado restringirá más el acceso al nuevo portador del Macaroon. Este mecanismo permite delegar el control de acceso a los portadores de los Macaroons, aumentando la escalabilidad del servicio y permitiendo un mayor número de consumidores que pueden acceder al servicio.

Las cabeceras del Macaroon son predicados o condiciones, delimitan el acceso al servicio. Por ejemplo, un usuario quiere compartir una foto privada. La foto está en un servicio de almacenamiento en la red, el usuario tiene un Macaroon para acceder a todas sus fotos en dicho servicio de almacenamiento. El usuario podrá atenuar el Macaroon para permitir el acceso a una foto. El Macaroon será compartido por el usuario, para que solo los usuarios que cumplan las condiciones impuestas en dicho Macaroon puedan visualizar la foto. Un ejemplo de cabeceras son:

- La operación a realizar sea sólo de lectura.
- La petición es sobre la imagen.jpg.
- El portador del Macaroon tiene que estar conectado al servicio de almacenamiento.
- El portador del Macaroon debe pertenecer al grupo Amigos en el servicio de almacenamiento.

Si el portador de un Macaroon desea acceder a la imagen, tendrá que entregar el Macaroon en el servicio de almacenamiento. El portador está obligado a cumplir todas las condiciones impuestas para acceder a la imagen.

1.4.1. Estructura

El Macaroon estará identificado por un conjunto de bits único generados de manera aleatoria, a modo de NONCE. Un servicio que tenga múltiples Macaroons puede reconocer uno en concreto utilizando el identificador.

El creador del Macaroon tendrá una clave privada o clave secreta, que solo conocerá el creador del Macaroon. De esta clave derivarán todas las firmas necesarias para comprobar la integridad del Macaroon.

La firma del Macaroon es una HMAC. La HMAC permitirá comprobar que ninguna entidad ha modificado ninguna de las cabeceras que previamente habían sido definidas. La firma mantiene la integridad del Macaroon.

Las cabeceras con predicados o condiciones es otra de las partes esenciales que poseen los Macaroons, ya que permiten atenuar el nivel de autorización y delimitar el acceso al servicio. Para ello existen varios tipos de cabeceras: First Party Caveat(FPC) y Third Party Caveat(TPC).

- Las FPC son cabeceras con condiciones que tiene que validar la entidad creadora del Macaroon. La condición impuesta puede ser cualquier tipo de condición. El portador del Macaroon debe cumplir las condiciones impuestas para poder tener acceso al servicio que solicita.

- Las TPC son cabeceras con condiciones que tiene que validar una tercera parte. La cabecera TPC incluye una condición. La condición está cifrada con una clave compartida, entre la entidad que añade la cabecera y la entidad que tiene que validar la condición. Previamente la entidad que validará la condición y la entidad que añade la cabecera deberán haber acordado la clave compartida. La clave compartida estará incluida en la cabecera e irá cifrada con una de las HMAC del Macaroon. Únicamente el creador del Macaroon puede descifrar la clave. La cabecera TPC incluye la localización de la entidad que debe validar la condición. Si el portador del Macaroon cumple la condición la tercera parte proporcionará un Discharge Macaroon(DM) al portador del Macaroon que será la prueba de que el portador ha cumplido la condición impuesta.

  El DM es una acreditación que se añadirá al TPC. La entidad encargada de validar la condición genera una HMAC si el portador del Macaroon cumple la condición. La HMAC se crea a partir de la condición y la clave compartida. Esta HMAC es necesaria para que el creador del Macaroon pueda verificar que la condición ha sido validada. En la sección 1.4.2 se explica en detalle el funcionamiento del Macaroon y sus cabeceras.

1.4.2. Creación y Atenuación

El Macaroon es creado para permitir el control de acceso a recursos. En la figura 1.4 se observa un ejemplo de creación de un Macaroon. El Macaroon poseerá un identificador único. El identificador del Macaroon(4F..87) es utilizado junto con una clave secreta K1 para crear la primera HMAC:

(K1, 4F..87) = 23..A7.

Sólo el creador del Macaroon posee la clave secreta K1 y solo él podrá verificar la HMAC. El creador del Macaroon decidirá las condiciones mínimas que tendrá impuestas el Macaroon. Estas condiciones establecidas por el servicio tendrán que ser cumplidas por todos los portadores de los Macaroons y no podrán ser modificadas.

Cada cabecera que defina una condición modificará la última HMAC. La nueva HMAC estará formada por la última HMAC definida más la nueva condición impuesta. La nueva HMAC deriva de la anterior HMAC, por lo que a su vez deriva de la clave secreta del Macaroon y sólo podrá ser comprobada por el creador del Macaroon.

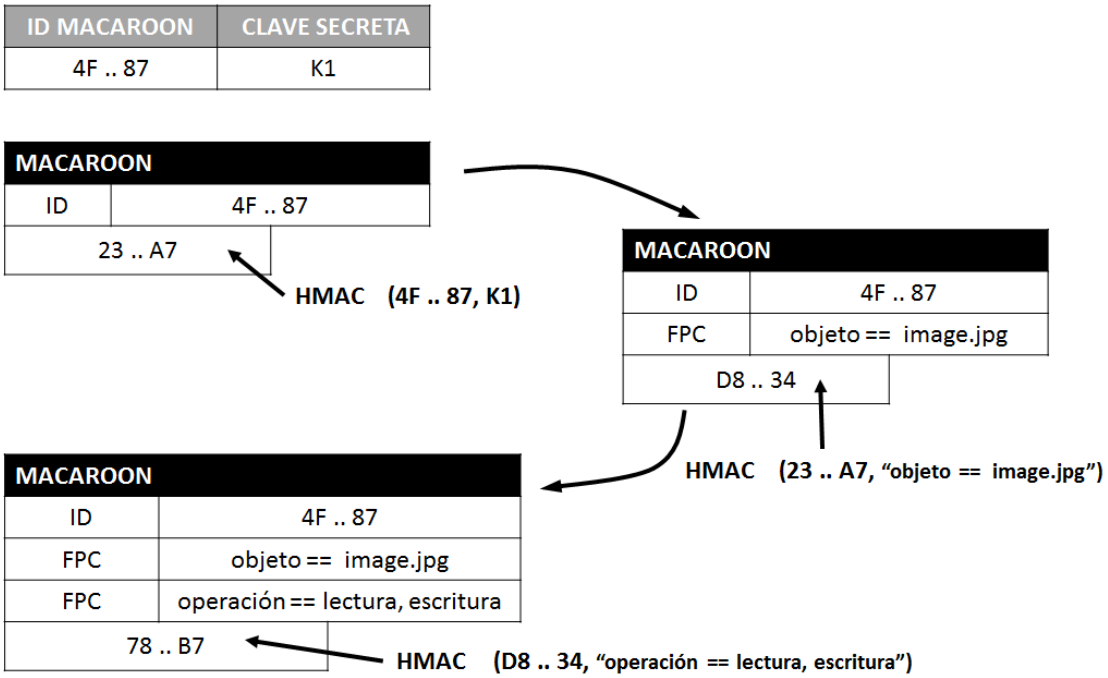

Figura 1.4. Creación del Macaroon

En el ejemplo visto en la figura 1.4, se muestra como crear un Macaroon para dar acceso a un objeto en concreto a otra entidad. El servicio crea un Macaroon usando un identificador(4F..87) y la clave secreta K1, que define la primera HMAC(23..A7). El servicio únicamente quiere dar acceso a la image.jpg. Se crea la cabecera FPC con la condición que delimite el acceso únicamente a imagen.jpg. Esta condición modifica la HMAC, partiendo de la última HMAC(23..A7) y la condición impuesta(objeto == image.jpg). Esta modificación de la HMAC evitará que la condición pueda ser alterada. La nueva HMAC calculada es D8..34. Si se desea añadir otra condición para fijar la operación a realizar a la imagen, se repetirá el proceso para añadir una cabecera FPC que permita por ejemplo la escritura y lectura del objeto, y se calculará la nueva HMAC, partiendo de la HMAC última (D8..34) y de la condición sobre la operación(operación == lectura, escritura), obteniendo la nueva HMAC(78..B7). El Macaroon ya estaría listo para ser entregado a otras entidades que quieran acceder al servicio.

Una entidad portadora puede dar el Macaroon recibido a otra entidad. La entidad puede entregar el Macaroon sin modificar, o puede atenuar el nivel de

acceso que tiene el Macaroon sobre el recurso. La entidad portadora puede atenuar el nivel de acceso del Macaroon añadiendo las cabeceras que necesite. Las cabeceras añadidas al Macaroon modificarán la HMAC, igual que si las añadiese el creador del Macaroon. La HMAC siempre será modificada al añadir las cabeceras, para poder mantener la integridad del Macaroon.

Si continuamos con el ejemplo visto en la figura 1.4, supongamos que el Macaroon ha sido enviado a un consumidor. El consumidor necesita que otros consumidores vean la image.jpg. Pero el consumidor que tiene el Macaroon no puede dejar que los otros consumidores modifiquen la imagen.jpg. El consumidor portador del Macaroon lo atenuará antes de entregárselo a otro consumidor o entidad. El portador del Macaroon, como se ve en la figura 1.5, añadirá la condición de solo lectura, evitando que otro consumidor o entidad que no sea él pueda modificar el objeto. Añadida la condición(operación == lectura) la utilizará junto con la última HMAC(78..B7) para obtener la nueva HMAC(C7..8D),

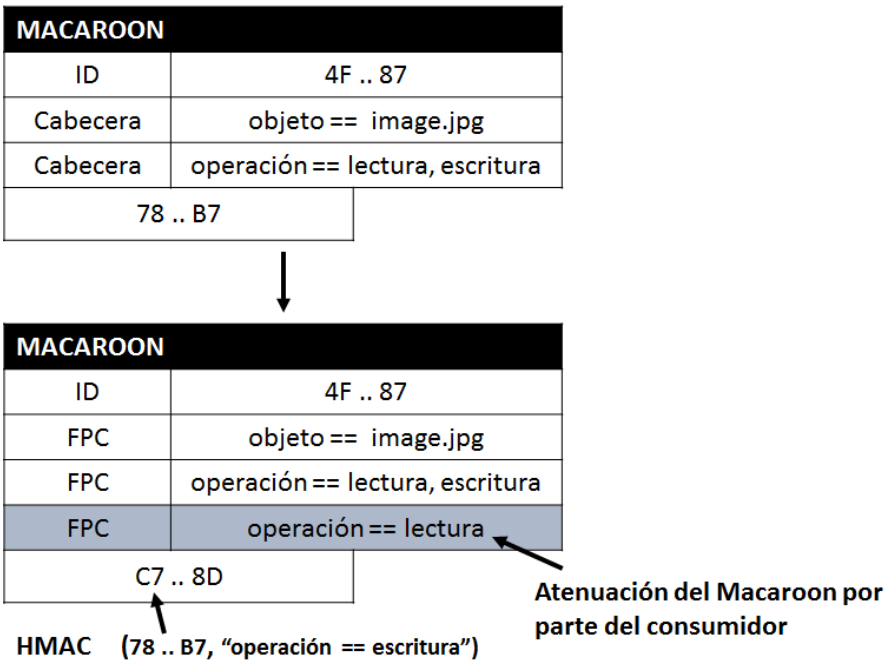

Figura 1.5. Atenuación del Macaroon por un consumidor

Las condiciones añadidas al Macaroon anteriormente son FPC. El creador del Macaroon es el único que puede validarlas. Pero tanto el creador del Macaroon como una entidad portadora del Macaroon pueden añadir una FPC. Además de FPC se pueden añadir TPC. Esta condición es necesaria que sea validada por una entidad tercera parte. Las condiciones impuestas en las cabeceras TPC no pueden ser comprobadas por el creador del Macaroon.

La entidad tercera parte proporcionará una prueba de que el portador del Macaroon ha cumplido la TPC a través de un DM.

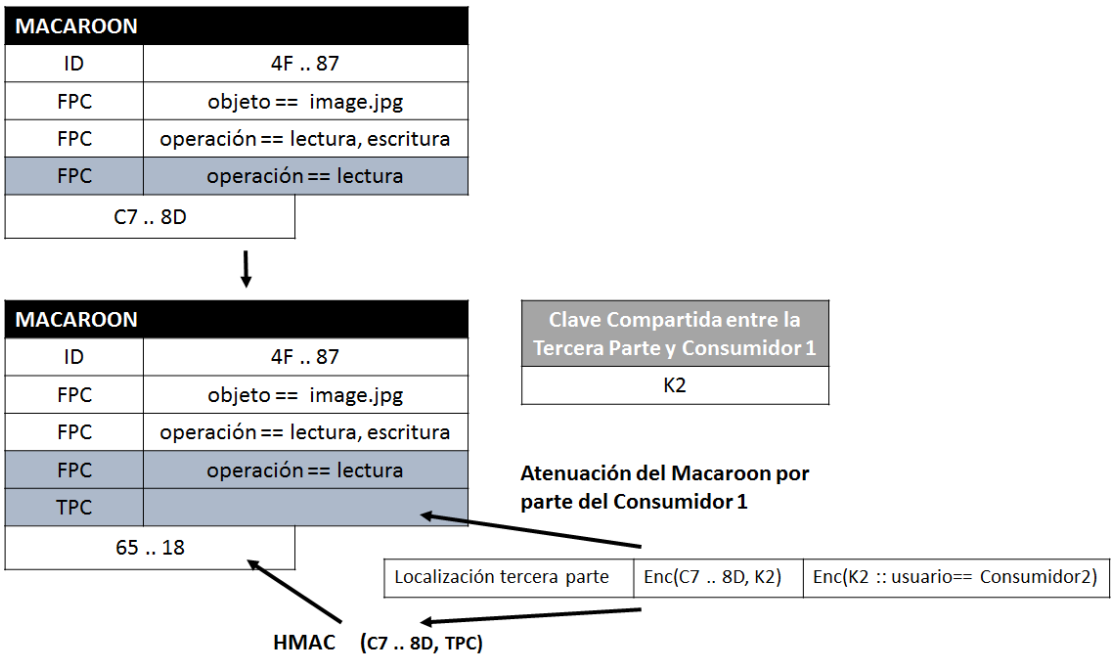

Figura 1.6. Atenuación del Macaroon con una TPC

En la figura 1.6 se continúa el ejemplo visto en las anteriores figuras. Aquí el consumidor portador del Macaroon, quiere que únicamente un usuario, Consumidor2 (autenticado en una entidad tercera parte), en concreto pueda ver la imagen. Antes de que el consumidor pueda añadir la TPC, debe ponerse de acuerdo para establecer una clave compartida simétrica ,K2, con la tercera parte. El consumidor añade una cabecera TPC, con la localización de la tercera parte, que ha de validar la condición. En el TPC irá la clave compartida que está cifrada con la última HMAC(C7..8D). Esta información es para que el creador del Macaroon pueda verificar su integridad. Y por último el TPC contendrá la condición impuesta por el consumidor en este ejemplo, que únicamente Consumidor2 pueda ver la image.jpg. La condición está cifrada con la clave compartida, K2. El portador del Macaroon ya puede enviar el Macaroon a Consumidor2 con la total certeza de que únicamente él podrá ver la imagen.jpg.

1.4.3. Validación

Para poder acceder al servicio que se solicita, el servicio creador del Macaroon deberá validar el Macaroon que recibe solicitándole una operación, por ejemplo, la visualización de una foto. El creador del Macaroon es el único que conoce la clave secreta, K1, esto hace que sea el único capaz de validar el Macaroon. La

validación del Macaroon se dividirá en dos partes: validación de la integridad y verificación de las condiciones.

La parte de validación de la integridad comprobará que el Macaroon no ha sido alterado. El servicio creador del Macaroon comprobará que la última HMAC contenida en el Macaroon se corresponde con las cabeceras del Macaroon. Como se observa en la figura 1.7, el servicio creador del Macaroon utilizará la clave secreta, K1, ligada a la identificación del Macaroon para obtener la primera HMAC. Utilizará la primera HMAC y la primera condición para obtener la siguiente HMAC. Repetirá el proceso hasta llegar a la última condición. Cuando obtenga la última HMAC comprobará si coincide con la HMAC contenida en el Macaroon. Si no coinciden las HMACs una entidad ha alterado el contenido del Macaroon, denegando así el acceso al servicio solicitado.

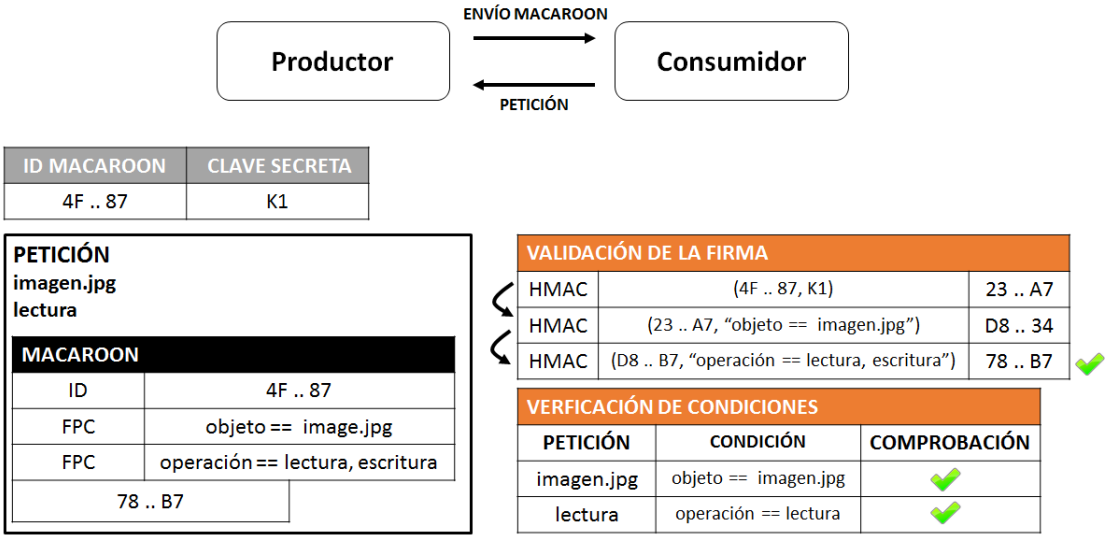

Figura 1.7. Validación de un Macaroon

Si ha sido validada la integridad del Macaroon se pasará a comprobar las condiciones incluidas en el Macaroon. Si la entidad que entrega el Macaroon no cumple alguna de las condiciones se le denegará el acceso al servicio solicitado. El creador del Macaroon únicamente podrá comprobar las condiciones incluidas en las cabeceras FPC.

En el ejemplo, un consumidor ha recibido el Macaroon de la figura 1.4 que le permite acceder al fichero image.jpg en modo lectura y escritura. El consumidor quiere visualizar la foto image.jpg. Para ello le envía al servicio de almacenamiento una petición para image.jpg de solo lectura. Como se puede ver en la figura 1.7 y se ha descrito previamente realiza la validación de la firma y de integridad. Comprueba que la HMAC que calcula coincide con la del Macaroon. Además debe

comprobar que se cumplen las condiciones, tiene permiso de lectura de image.jpg. Esto se puede comprobar a través de las dos FPC contenidas en el Macaroon. El servicio le proporciona acceso al consumidor al objeto solicitado.

Supongamos ahora que un Consumidor1 le ha entregado el Macaroon de la figura 1.5 al Consumidor2. El Consumidor1 se ha encargado de atenuar el Macaroon para que Consumer2 únicamente pueda visualizar el objeto image.jpg sin que pueda modificarlo, pues se tienen que cumplir todos los FPC contenidos en el Macaroon. Como se puede observar en la figura 1.8, si Consumidor2 envía la petición al productor, para guardar un cambio en el objeto image.jpg, el productor revisará que las HMAC coinciden y que el Macaroon no ha sido alterado. Sin embargo, al revisar las condiciones, comprueba que no cumple la última condición. Consumidor2 no tiene permiso de escritura en el objeto imagen.jpg. Al no cumplir todas las condiciones el productor le deniega el acceso al objeto a Consumidor2.

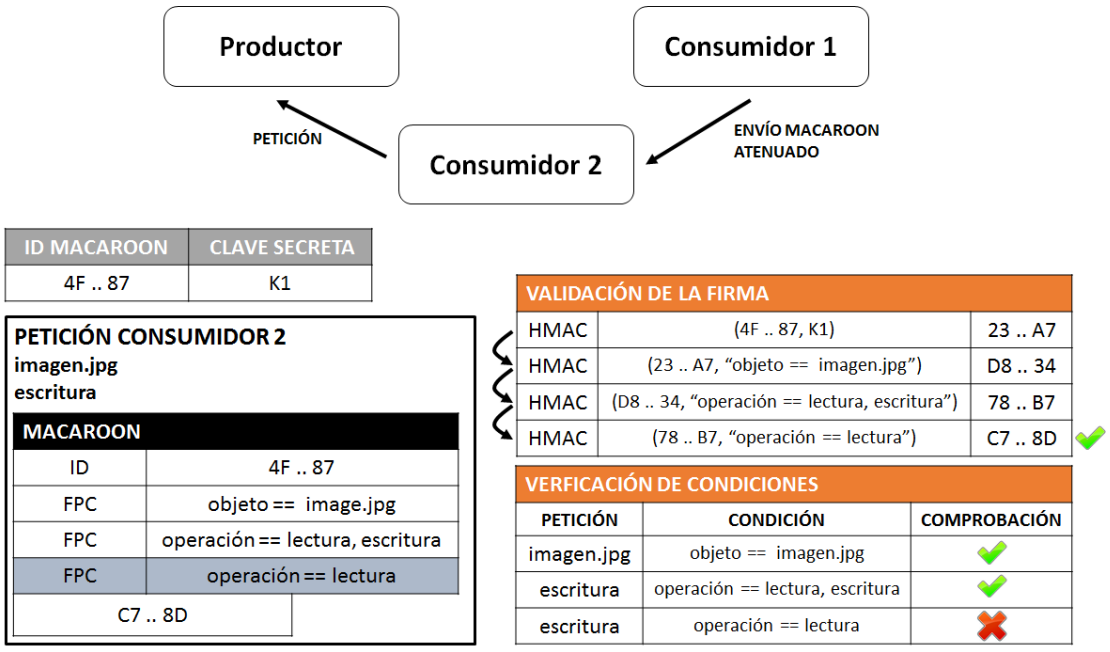

Figura 1.8. Validación fallida de un Macaroon

El servicio creador del Macaroon no puede validar las condiciones de las cabeceras TPC. Las cabeceras TPC deben ser validadas por una tercera entidad. El portador del Macaroon extraerá de la cabecera TPC la localización de la entidad que ha de validar la condición. La condición incluida en la cabecera TPC irá cifrada con una clave que la entidad portadora del Macaroon desconocerá. La clave será compartida entre la tercera parte y la entidad que haya incluido el TPC, K2. El portador del Macaroon necesita una prueba de que cumple la TPC y se comunicará con la tercera parte para conseguirla, DM.

Cuando la tercera parte haya recibido la condición cifrada, utilizará la clave compartida con la entidad que ha introducido esa cabecera para poder descifrar la condición. Comprobará que la entidad que le ha enviado la petición cumple la condición. En el ejemplo de la figura 1.9 la tercera parte debe comprobar si el usuario es Consumer2, esta condición la ha recibido cifrada con K2. El sistema utilizado por la tercera parte para verificar que se cumple la condición es independiente a los Macaroons, pudiendo utilizar cualquier otro mecanismo. Cuando la tercera parte compruebe que cumple la condición creará un DM, que firmará con la clave compartida. El DM llevará una HMAC generada a partir de K2.

Una vez el portador del Macaroon tenga el DM, como se puede observar en la figura 1.9, será añadido a la cabecera TPC, y podrá ser enviado al servicio creador del Macaroon para poder realizar la petición al servicio. El servicio creador del Macaroon verificará la integridad del Macaroon, como en casos anteriores. Revisará la cabecera TPC. En primer lugar extraerá la clave compartida entre la entidad que ha introducido la cabecera TPC y la tercera parte. La clave compartida K2 está cifrada con la HMAC de la última FPC(C7..8D) que deriva de la clave secreta K1. Utiliza la clave secreta K1 para obtener la HMAC(C7..8D) que le permitirá obtener la clave compartida K2. Comprueba que la clave compartida K2 junto con la condición cifrada da como resultado la misma HMAC contenida en el DM(AA..AA). Por último, si nada ha fallado, el servicio creador del Macaroon verificará que la entidad que realiza la petición cumple las condiciones.

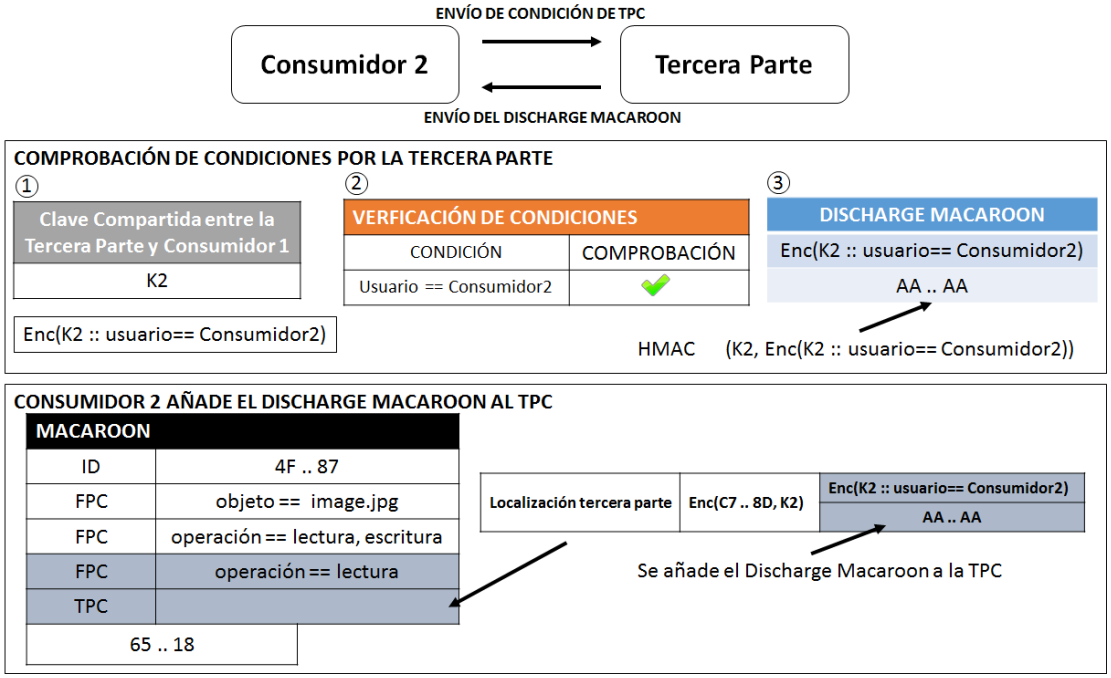

Figura 1.9. Petición de Discharge Macaroon

En el ejemplo mostrado en la figura 1.9, un Consumidor1 le ha entregado el Macaroon de la figura 1.6 a Consumer2, para que solo él pueda ver el objeto image.jpg añadiendo una cabecera TPC. Consumer2 deberá autenticarse con la tercera parte y obtener el DM. Como se puede apreciar en la figura 1.9 Consumer2 envía a la tercera parte la condición cifrada. La tercera parte descifra la condición con la clave compartida K2. Utiliza cualquier mecanismo de verificación(que es independiente del mecanismo del Macaroon) para comprobar que Consumer2 es quién dice ser, y verifica la condición. Le envía el DM a Consumer2 que incluirá la HMAC(AA..AA) calculada a partir de K2.

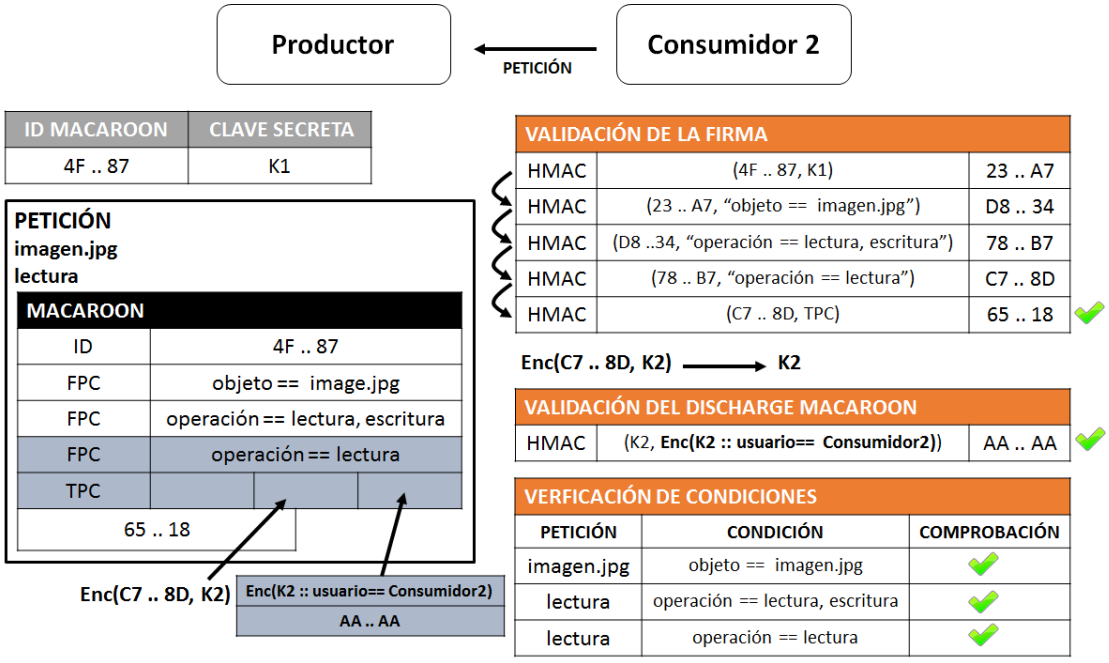

Figura 1.10. Validación de un Macaroon con una cabecera TPC

En la figura 1.9 se puede ver como Consumer2 añade el DM a la cabecera TPC. A continuación realiza la petición a Producer para poder ver el objeto image.jpg, como se puede apreciar en la figura 1.10. El productor revisará la integridad del Macaroon y a continuación extraerá la clave compartida K2. El productor utilizará la clave para comprobar si la HMAC del DM es correcta, y comprobar así que la cabecera TPC realmente ha sido validada por una tercera parte. Como se puede ver en la figura 1.10 está todo correcto, verifica las condiciones impuestas en el Macaroon y permite a Consumer2 visualizar el objeto imagen.jpg.

Capítulo 2
Objetivos y metodología

Objetivos

El objetivo principal de este trabajo de fin de grado es utilizar un sistema de autorización descentralizado para acceder a recursos a través de Macaroons en una red NDN. Para conseguir el objetivo principal se definen una serie de subobjetivos:

- Subobjetivo 1: Estudio y comprensión de la arquitectura de red NDN.
- Subobjetivo 2: Estudio y comprensión del mecanismo de autorización descentralizada proporcionada por Macaroons.
- Subobjetivo 3: Estudio y comprensión de la librerías de programación necesarias para desarrollar aplicaciones NDN.
- Subobjetivo 4: Estudio y comprensión de las librerías de programación para el manejo de los Macaroons.
- Subobjetivo 5: Diseño de un modelo de autorización descentralizado en el que se puede aplicar el mecanismo de Macaroons dentro del entorno de la arquitectura NDN.
- Subobjetivo 6: Desarrollo de una aplicación que permita probar el uso de Macaroons en una red NDN.

Metodología

Para lograr el objetivo se sigue un modelo de desarrollo en espiral a partir de un prototipo ya creado. El desarrollo en espiral está formado por ciclos divididos en cuatro fases: determinar o fijar objetivos, desarrollar el software, pruebas y planificación del próximo ciclo de desarrollo.

Los objetivos determinados en cada ciclo han sido decididos de manera conjunta entre mi tutora y yo. Determinando objetivos funcionales, estructurales y temporales. En cada ciclo los objetivos propuestos para la siguiente fase eran más complejos completando el desarrollo ya realizado. Esta manera de plantear el desarrollo del trabajo de fin de grado ha permitido realizarlo en fases graduales.

Previsión de la planificación temporal:

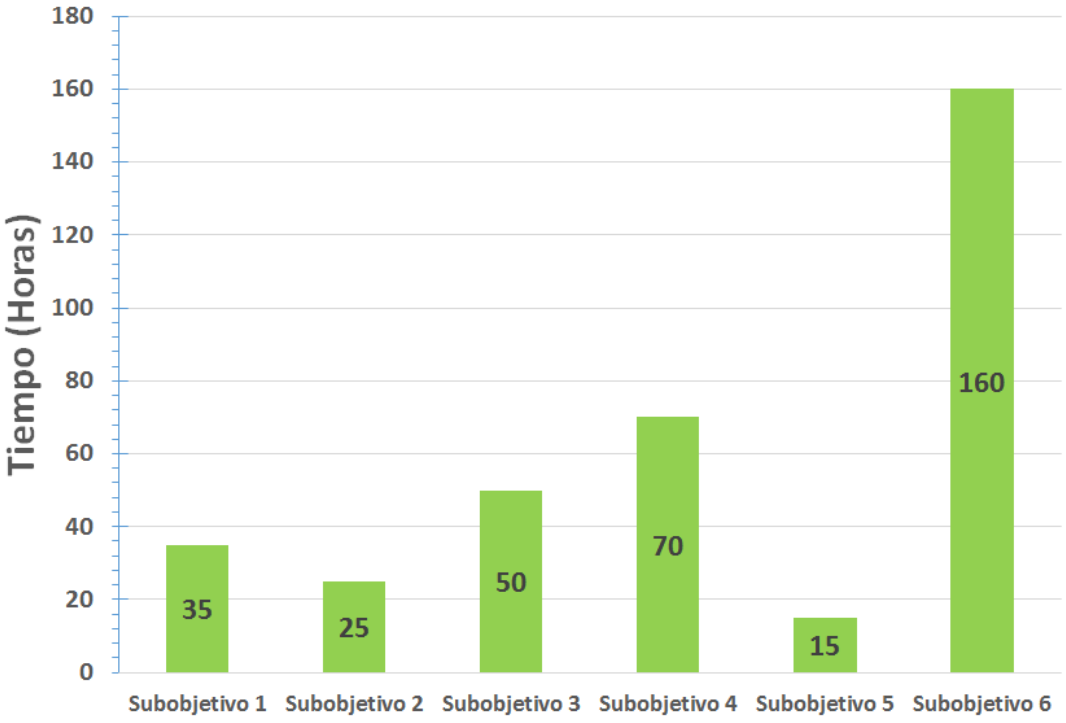

Figura 2.1. Horas dedicadas a la realización de cada subobjetivo

Herramientas

La herramientas utilizadas para el desarrollo del sistema de acceso a recursos a través de Macaroons son las siguientes:

- Librería ndn-cxx: versión 0.4.1
- Networking Forwarding Daemon(NFD): version 0.4.1
- Librería libsmacaroons.
- Librería protobuf.
- Librería libNDNMacaroons.
- Entorno de programación Sublime.

- Compilador G++
- Github

Capítulo 3
Diseño de control de acceso a recursos a través de Macaroons

En este capítulo se explica el diseño de un sistema que permite delegar el control de acceso a recursos entre diferentes entidades. Cada entidad tiene una función específica dentro del control de acceso a recursos. Una entidad publica los datos cifrados para que sólo aquellos que tengan dicha clave puedan acceder a su contenido. Además debe publicar esta clave cifrada con una clave de grupo. Solo los miembros de ese grupo tendrán acceso a la clave con la que han sido cifrados los datos. Otra entidad tiene la función de proporcionar los Macaroons. Los Macaroons son las acreditaciones necesarias para obtener la clave de grupo. Y una tercera entidad es la encargada de comprobar los Macaroons, y entregar la clave de grupo. La clave de grupo es necesaria para cifrar la clave de datos. La entidad que se encarga de publicar los datos cifrados los cifra con la clave de datos. Esto se debe a que esta entidad no comprueba quien le solicita los datos y puede estar publicando datos nuevos sin tener aún la clave de grupo.

Cuando un miembro de un grupo quiere acceder a los datos publicados para ese grupo, el miembro del grupo solicitará el Macaroon que incluirá un TPC. Este Macaroon que contiene un TPC le obligará a demostrar que es un miembro de ese grupo, en cuyo caso, le proporcionará la clave de grupo que le permitirá acceder a los datos.

Las entidades que componen el sistema son las siguientes:

- Producer: es la entidad que contiene los datos. Los datos son entregados a cualquier entidad que los solicite. Producer no necesita conocer la identidad de quién solicita el recurso. Los datos irán cifrados con una clave de datos a la que sólo tendrán acceso los usuarios de un grupo. La clave de datos se cifra con la clave de grupo que también publica. De esta forma el productor cifra los datos con la clave de datos que él elija y cuando un miembro del

grupo solicite la clave, se la cifrará con la clave de grupo. Solo las entidades que sean miembros del grupo, tienen la clave de grupo, podrán descifrar la clave de datos y podrán tener acceso a los datos solicitados.

- Access Controller(AC): es la entidad encargada de proporcionar los Macaroons. AC entrega los Macaroons a quién se lo solicite. Esta entidad no necesita conocer la identidad de quién solicita el Macaroon. AC conoce los prefijos de los nombres relativos a cada grupo. Cada grupo tiene un Macaroon único, que es la acreditación para obtener la clave de grupo.

- Group Keys Distributor(GKD): esta entidad se encarga de validar las acreditaciones obtenidas de AC. Comprueba que el poseedor de la acreditación es quién dice ser y pertenece al grupo que indica la acreditación. GKD es la entidad que conoce los miembros de cada grupo. Si la entidad que posee el Macaroon o acreditación cumple la condición de pertenecer al grupo indicado, se le entregará la clave de grupo.

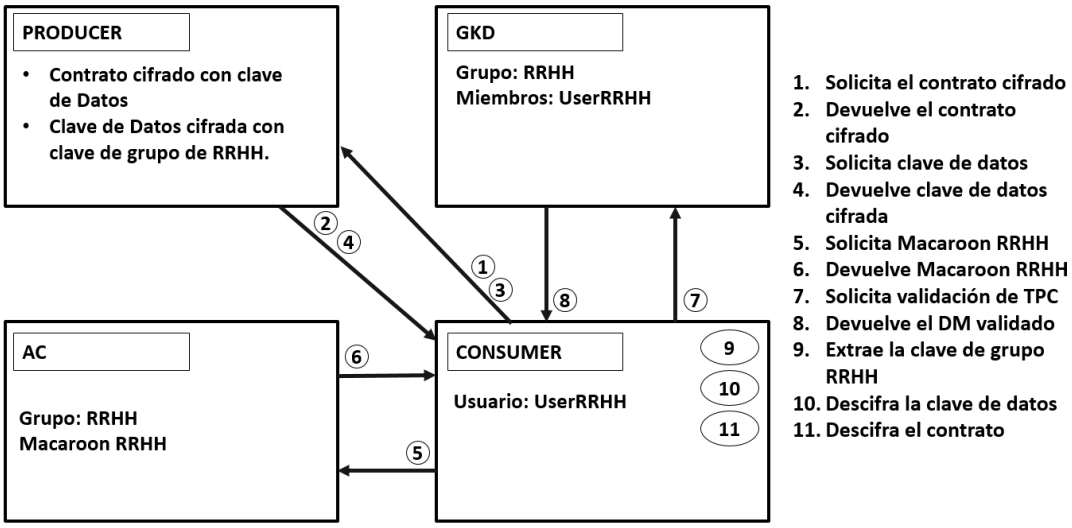

Figura 3.1. Ejemplo de intercambio de paquetes, para solicitar los datos y poder descifrarlos

El modelo diseñado en este trabajo de fin de grado puede aplicarse al siguiente escenario: una empresa con diferentes departamentos, como se puede observar en la figura 3.1. Cada departamento define un grupo. Los miembros incluidos en el grupo son los trabajadores del departamento. Un trabajador del departamento de Recursos Humanos(RRHH) le pide al servidor central, que es Producer, un contrato de un empleado. El contrato está cifrado con una clave de datos y esta clave está cifrada con la clave de grupo de RRHH, donde solo un trabajador que posea la clave de dicho grupo podrá descifrar los datos. La clave de datos se debe a

que diferentes departamentos pueden tener que acceder a un mismo dato, y por eso no debe ir cifrado con la clave de grupo. El trabajador de RRHH pedirá también la clave de datos cifrada. El trabajador del departamento de RRHH no posee la clave de grupo aún, por lo que necesita una acreditación que solo puede proporcionar AC. La acreditación es un Macaroon. La acreditación obtenida es única para cada grupo, y el trabajador del departamento de RRHH necesita validarla en una tercera parte, en este caso GKD. GKD validará la acreditación comprobando que el usuario es quién dice ser y que pertenece al grupo que viene en la acreditación. Si GKD valida correctamente la condición impuesta en la acreditación, entregará un resguardo al empleado. El resguardo es un DM, que contiene la clave de grupo necesaria para conseguir descifrar la clave de datos y por consiguiente los datos.

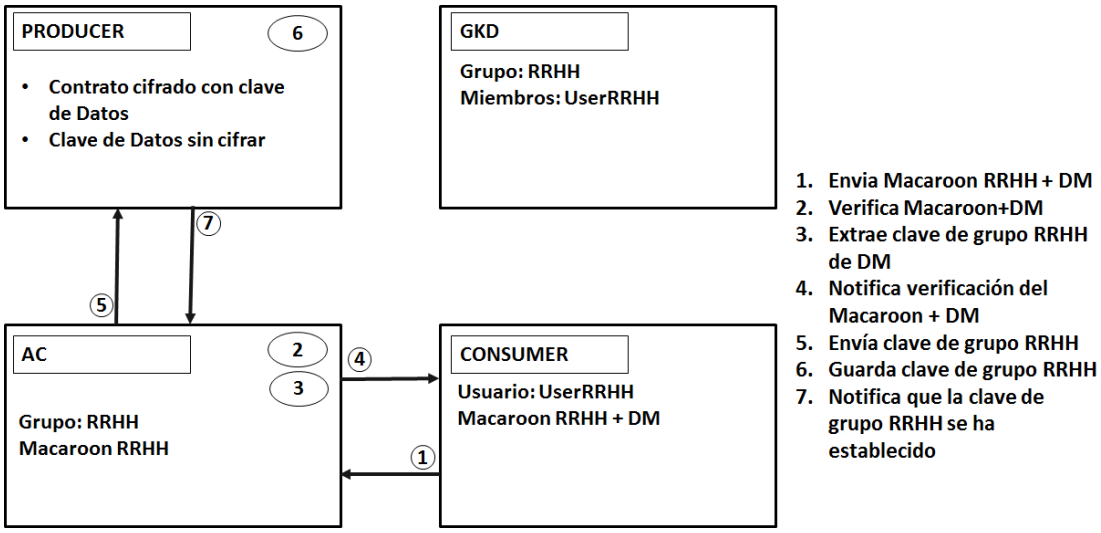

Figura 3.2. Ejemplo de establecimiento de la clave de grupo en un productor

Inicialmente el productor genera los datos y los cifra con una clave de datos que solo él conoce, no tendrá la clave de grupo para cifrar la clave de datos. Es necesario que haya una comunicación previa entre AC y Producer. Se puede observar en la figura 3.2 el intercambio de paquetes que tiene que existir para establecer la clave de grupo. El consumidor envía su Macaroon más el DM a AC. AC debe comprobar el Macaroon para ver que cumple todos los requisitos que se le piden. Si cumple los requisitos extrae la clave de grupo que se encuentra en DM y se la envía cifrada al productor. El productor establecerá la clave de grupo y podrá cifrar la clave de datos con la clave de grupo para que sea transportada de manera confidencial. Véase en más detalle sobre esta comunicación en la sección 3.7.

El sistema de control de acceso a recursos puede ser ampliado, delegando en otros usuarios el acceso a los recursos. Por ejemplo, el contrato solicitado tiene que ser revisado por alguien ajeno a la empresa, como una gestora. El trabajador

de RRHH podrá modificar la acreditación o Macaroon. Se modificará el Macaroon para atenuarlo. Cuando el Macaroon haya sido atenuado, la gestora solo podrá tener acceso al contrato indicado. La gestora externa autorizada deberá validar el Macaroon con GKD autenticándose para demostrar su identidad y así poder obtener la clave necesaria para descifrar el contrato.

A partir de este momento este escenario de uso para explicar la comunicación entre las diferentes entidades.

3.1. Macaroon

Los Macaroons son una parte importante del sistema diseñado para el control de acceso a recursos. Los Macaroon permitirán delegar el control de acceso entre diferentes entidades. AC entregará los Macaroons para que las entidades puedan obtener la clave de grupo.

Los Macaroons son acreditaciones. Las acreditaciones incluyen ciertas condiciones que las entidades han de cumplir. La condición que tiene que ser validada por una tercera entidad y que permite comprobar pertenencia a un grupo es una TPC. La TPC ha de ser validada por GKD. La condición impuesta en TPC será ser miembro de un grupo específico. Si la entidad cumple la condición recibirá por parte de GKD un DM. El DM contendrá la clave de grupo. La clave de grupo es la firma de DM. En el sistema diseñado se ha utilizado otra cabecera que establecerá el período de validez del Macaroon, y únicamente puede ser validada por AC.

3.2. Producer

Producer publica todos los datos. Los datos están cifrados con una clave de datos única para cada grupo. La clave de datos está cifrada con una clave de grupo. Producer publicará la clave de datos. Producer entregará los datos y la clave de datos cifrada con la clave de grupo a toda entidad que la solicite. Las entidades no tienen la necesidad de identificarse ante el productor. Solo las entidades que posean la clave de grupo podrán descifrar la clave de datos y por tanto los datos.

El Producer atenderá únicamente los paquetes Interest que estén nombrados con los siguiente prefijos:

- /example/producer/dptoRRHH/contracts ->En este prefijo recibirá tanto órdenes, como la petición de los datos o recursos. En este ejemplo, el prefijo al que está atado es el identificador del grupo RRHH, ya que éste debe conocer que prefijo va con que grupo.

- /ndn/keys/producer/ ->En este prefijo se recibirá la petición del certificado con las claves públicas del Producer, pudiendo ser tanto la ksk o la dsk.

Órdenes que podrá recibir en el prefijo /example/producer/dptoRRHH/contracts:

- getKeyData: se solicita a través del Interest la clave de datos cifrada de RRHH.
- setKeyGroup: se realiza una solicitud a través del Interest para establecer la clave de grupo. Si la clave de grupo ya existe el Interest es descartado, en caso contrario es almacenada.

Si se recibe alguna otra cosa que no sea una orden de las arriba indicadas, será considerado como la petición de un dato y se comprobará si existe dentro del Producer, para poder generar el Data y enviarlo.

3.3. Access Controller (AC)

AC es el encargado de proporcionar los Macaroon a quién se lo solicite. Es la única entidad del sistema que puede validar los Macaroons. La información de que dispone AC sobre los grupos es: prefijos de los nombres de los datos a los que pertenece cada grupo y sus Macaroons. AC creará un Macaroon para cada grupo, añadiendo las cabecera FPC y TPC.

El AC atenderá únicamente los paquetes Interest que estén nombrados con los siguiente prefijos:

- /example/accescontroller/ ->En este prefijo se recibirán las órdenes necesarias para solicitarle información.
- /ndn/keys/accesscontroller/ ->En este prefijo se recibirá la petición del certificado con las claves públicas del AC, pudiendo ser tanto la KSK o la DSK.

Órdenes que podrá recibir en el prefijo /example/accesscontroller :

- getMacaroon: se está solicitando un Macaroon de un grupo en concreto.
- updateGroupKey: se está solicitando establecer la clave de grupo con Producer. Se recibe el Macaroon y DM necesarios para la verificación del Macaroon, y poder extraer la clave de grupo que envía a Producer una vez verificado el Macaroon.

En el caso de que la orden recibida no corresponda con ninguna de las anteriores el Interest será inmediatamente descartado.

3.4. Group Keys Distributor (GKD)

GKD es la entidad con la función de validar las condiciones impuestas en la cabecera TPC del Macaroon. Comprueba que la entidad que desea validar la condición la cumple. La condición impuesta en la cabecera TPC es que la entidad pertenezca al grupo indicado en la condición. La información de los grupos de que dispone GKD son los miembros que componen cada grupo. Si la entidad que solicita la clave de grupo cumple la condición, GKD enviará la clave de grupo incluida en un DM.

El GKD atenderá únicamente los paquetes Interest que estén nombrados con los siguiente prefijos:

- /example/groupKeysDistributor/ ->En este prefijo se recibirán las órdenes necesarias para pedir datos o ejecutar un comando.
- /ndn/keys/groupkeysdistributor/ ->En este prefijo se recibirá la petición del certificado con las claves públicas de GKD, pudiendo ser tanto la KSK o la DSK.

Órdenes que podrá recibir en el prefijo /example/groupKeysDistributor :

- getDischargeMacaroon: se solicita a GKD que valide una condición impuesta en una TPC. Si se cumple la condición enviará un DM con la clave de grupo.
- setSharedSecret: se solicita a GKD establecer una clave compartida con AC. La clave es necesaria para cifrar y descifrar la condición impuesta en la cabecera TPC.

En el caso de que la orden recibida no corresponda con ninguna de las anteriores el Interest será inmediatamente descartado.

3.5. Estructuras de Nombres

Los nombres[8] son una parte importante de la arquitectura NDN ya que tienen que seguir una serie de reglas para poder establecer una comunicación satisfactoria. Los recursos del sistema se identifican por los nombres. El nombre se incluirá en el paquete Interest y se obtendrá en el paquete Data de respuesta. Los nombres representan datos y también representan la realización de operaciones. Por ejemplo, el nombre /example/producer/dptoRRHH/contracts/<files>solicitará un contrato y obtendrá como respuesta un paquete Data que contiene el contrato. El nombre /example/producer/dptoRRHH/contracts/setGroupKey le indica a Producer que debe establecer una clave de grupo, como respuesta a este Interest se recibe un paquete Data informando si la operación se ha realizado correctamente.

En la Figura 3.3 se muestra un esquema de los nombres diseñados para un sistema de acceso a recursos. Esta operación debe solicitarse antes de que empiece a funcionar todo el sistema.

3.5.1. Convenio para los nombres

Los nombres que utilizarán las entidades de este sistema tiene los siguientes prefijos:

- /<nombre-sistema>/<Entidad>->El <nombre-sistema>indica donde se está utilizando este mecanismo y <Entidad>sirve para identificar la entidad encargada de procesar los Interest recibidos a este nombre. En el ejemplo se han establecido tres entidades: /example/producer, /example/accesscontroller y /example/groupKeysDistributor.

- /ndn/keys/<Entidad>->A la hora de crear las claves se deberá seguir esta regla, siendo el último campo el nombre de la entidad a la que pertenecerán las claves.

En la Figura 3.3 se muestra un esquema de nombres para el sistema de ejemplo definido.

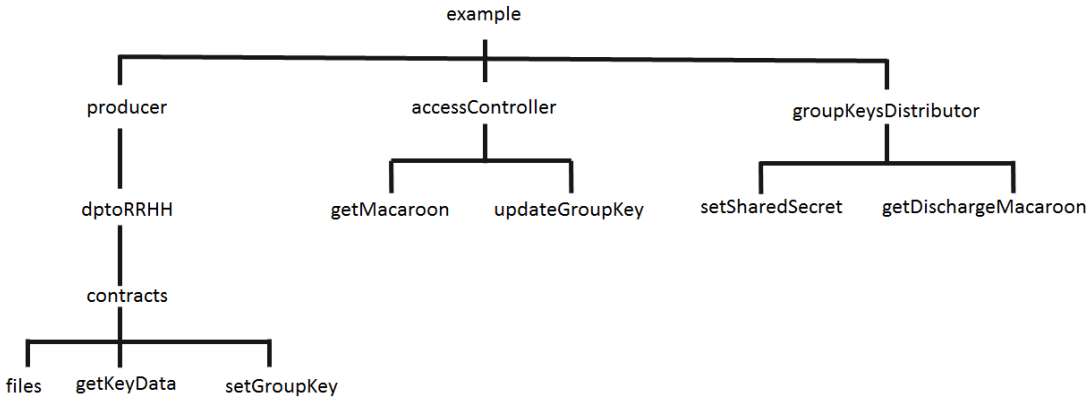

Figura 3.3. Diagrama de los nombres utilizados en las entidades del sistema de delegación.

A continuación se explican los nombres manejados por las entidades del sistema definido.

3.5.2. Interacción Consumer <−>Producer

Producer espera recibir un paquete Interest en el nombre /example/producer/dptoRRHH/contracts/<files>para proporcionar los datos de un contrato para el grupo del departamento de RRHH.

Los componentes del nombre /dptoRRHH/contracts/ crean el prefijo. Producer sabe a que grupo pertenece cada prefijo. El prefijo actuará como un sistema de archivos, e identificará dónde encontrar los datos solicitados.

En el caso de recibir el nombre de un archivo en la solicitud del Interest, Producer añadirá el siguiente componente al nombre incluido en el Data:

- notFound: el archivo <files>no existe o no ha sido encontrado.
- encryptedData: el contenido del paquete Data contiene los datos cifrados solicitados <files>.

Producer también puede recibir de Consumer la orden getKeyData:

/example/producer/dptoRRHH/contracts/getKeyData

Producer responderá añadiendo el siguiente componente al nombre:

- notEncrypted: Producer aún no tiene la clave de grupo necesaria para encriptar la clave de datos y poder enviarla de manera segura.
- keyDataEncrypted: el paquete Data contiene la clave de datos cifrada que se solicita.

3.5.3. Interacción Consumer <–>Access Controller

El AC espera recibir paquetes Interest en el siguiente nombre:

/example/accesscontroller/getMacaroon/<grupo>

Este mensaje es específico para pedir el Macaroon asociado a un grupo, siendo getMacaroon el comando, y el siguiente campo <grupo>el que le indicará a AC de que grupo se necesita la clave. En nuestro ejemplo el grupo es RRHH.

AC también espera recibir un Interest para verificar el Macaroon y establecer la clave de grupo con Producer.

/example/accesscontroller/updateGroupKey/...
.../<clave-sesión-encriptada>/<M>/<DM-encriptado>/...
.../<dataName>/<grupo>/<versión>

Los campos que aparecen en este nombre son:

- updateGroupKey: es el comando que indica la acción a realizar.
- <clave-sesión-encriptada>: Clave simétrica encriptada con la clave pública de AC, esta clave de sesión será necesaria para descifrar el apartado <DM-encriptado>.
- <M>: es el Macaroon que incluye el TPC y el cuál tendrá que ser validado por AC.
- <DM-encriptado>: es el DM necesario para comprobar que la condición impuesta en TPC ha sido validada. El DM está cifrado por la clave sesión previamente establecida, para que ninguna otra entidad maliciosa pueda utilizar esta acreditación.
- <dataName>: Nombre que identifica el prefijo donde se encuentran los datos cifrados. En este ejemplo es /example/producer/dptoRRHH/contracts.
- <grupo>: nombre del grupo al que pertenece el Macaroon.
- <versión>: versión del grupo.

3.5.4. Interacción Consumer <−>Group Keys Distributor

GKD espera recibir paquetes Interest con el siguiente nombre:

/example/groupKeysDistributor/getDischargeMacaroon/...
.../<clave-sesión-encriptada>/<condición-cifrada>/<grupo>/<versión>

GKD espera recibir este nombre en un Interest solicitando el DM. Los componentes del nombre son:

- getDischargeMacaroon: es el comando que indica la acción a realizar.
- <clave-sesión-encriptada>: Clave simétrica encriptada con la clave pública de GKD Esta clave de sesión será necesaria para encriptar el DM en caso de que la condición del TPC se compruebe con éxito.
- <condición-cifrada>: Es la parte del Macaroon que contiene la condición que tiene que validar obligatoriamente GKD.
- <grupo>: nombre del grupo al que pertenece el Macaroon.
- <versión>: versión del grupo.

GKD responderá a la solicitud con un paquete Data, el cuál modificará el nombre del Interest añadiéndole un componente en función del resultado de la condición:

- authenticated: Indicará a la entidad que realiza la petición que la condición ha sido verificada con éxito, y el contenido del Data es el DM encriptado con la clave de sesión.
- notAuthenticated: Indicará a la entidad que realiza la petición que no ha podido verificar la condición impuesta en el TPC y el contenido del Data esta vacío.

A continuación se le añadirá también la versión del paquete Data.

3.5.5. Interacción Access Controller <–>Producer

AC enviará un Interest con el siguiente nombre para poder informar de la clave de grupo a Producer:

/example/producer/dptoRRHH/contracts/setKeyGroup/<clave-grupo-encriptada>/

A continuación se describen los campos del nombre:

- setKeyGroup: indicará al Producer operación que se quiere realizar sobre el grupo, en este caso establecer la clave de grupo.
- <clave-grupo-encriptada>: Clave de grupo encriptada con la clave pública del Producer.

Producer responderá a la acción con un Data, del cuál solo modificará su nombre, ya que AC en ningún momento pedirá información o datos a éste.

- succesfull: indicará que la clave ha sido establecida con éxito.
- alreadyExist: la clave de grupo solo podrá ser almacenada una vez, si ya existe indicará a Producer que la clave para ese grupo ya existía.

3.5.6. Interacción Access Controller <–>Group Keys Distributor

AC solo iniciará una comunicación con GKD para establecer una clave compartida. Esta comunicación se tiene que establecer durante el arranque del sistema. La clave es necesaria para cifrar y descifrar la condición impuesta en la cabecera TPC en el Macaroon. Para ello AC enviará a GKD el siguiente Interest.

/example/groupKeysDistributor/setSharedSecret/...
.../<nombre-clave-compartida>/<clave-encriptada>

Los campos del nombre son descritos a continuación:

- setSharedSecret: esta orden indicará a GKD que es necesario establecer una clave compartida con este AC. Esta orden solo puede llevarse a cabo por AC, si recibe la orden de otra entidad que no sea AC se descartará el paquete.

- <nombre-clave-compartida>: Aquí se establecerá el nombre de la clave compartida.

- <clave-encriptada>: clave compartida encriptada con la clave pública de GKD.

GKD contestará con un Data sin contenido. Solo en el caso de que la clave haya sido establecido con éxito, añadirá el componente established.

3.6. Claves KSK y DSK

En la arquitectura NDN cada entidad debe tener un par de claves KSK y DSK, que son en realidad claves RSA(véase sección 1.2.3) Las claves serán utilizadas en diferentes circunstancias. Los nombres que tendrán las claves serán /ndn/keys/<entidad>/ksk-XXXXXXXX y /ndn/keys/<entidad>/dsk-XXXXXXXX. Estos nombres se utilizarán para localizar las claves y así poder firmar paquetes, cifrar datos o bien poder extraer el certificado de la clave pública y enviarlo en caso de que alguna entidad lo pida. En la Figura 3.4 se muestran los nombres de las claves para entidades definidas en nuestro sistema.

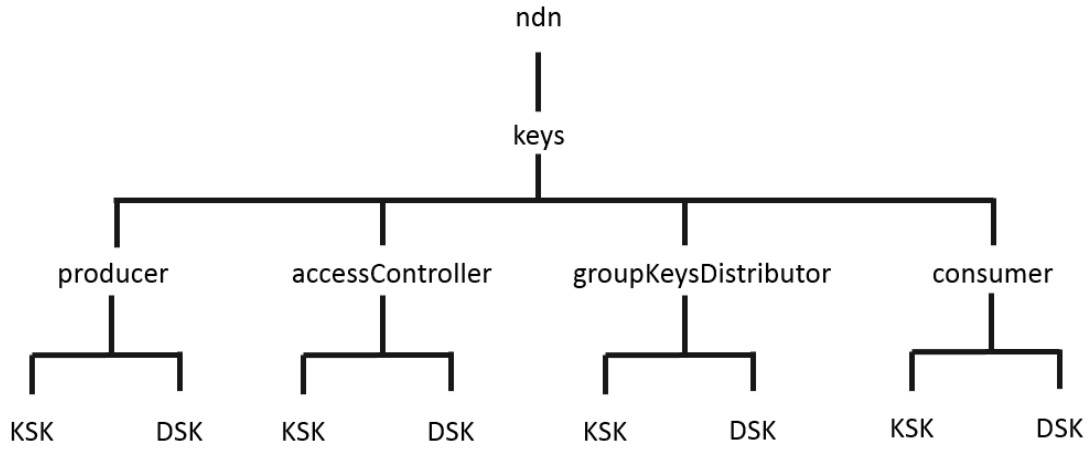

Figura 3.4. Diagrama de nombres para las identidades de las claves KSK y DSK de las entidades producer, accessController, groupKeysDistributor y consumer.

Cada una de las claves tendrá una función específica dentro del sistema de delegación. Las claves KSK se utilizarán para firmar los paquetes, tanto Interest como Data. La firma de los paquetes Interest y Data permitirá conocer la identidad del creador de los paquetes. La firma permite comprobar la integridad de los paquetes, permitiendo detectar posibles alteraciones en los paquetes por una entidad maliciosa. Las claves DSK no serán utilizadas para firmar paquetes. Las claves DSK se utilizarán para cifrar el contenido que necesitamos que se transporte de manera confidencial.

3.7. Descripción del funcionamiento del sistema de acceso

La primera entidad que se inicia es Producer. Producer se encargará de generar las claves de datos para los datos que publique. En el ejemplo descrito supondremos que solo publica un tipo de datos (los contratos) para un grupo(RRHH), con lo que solo generará una única clave de datos. A partir de este momento podrá encriptar dichos datos. Producer tiene que publicar la clave con la que está cifrando los datos. Esta clave únicamente puede estar accesible a los miembros del grupo que tengan acceso a ella. Producer cifra la clave de datos con la clave del grupo. Esta acción será imposible realizarse, ya que aún no dispone de la clave de grupo. Producer podrá publicar los datos cifrados. Aunque no pueda entregar la clave de datos para descifrar los datos.

El siguiente en arrancar será GKD, que cargará los grupos y los miembros pertenecientes a cada grupo, y se quedará esperando eventos en los prefijos de los nombres establecidos.

Por último se iniciará AC. Establecerá una clave de sesión con GKD, ésta será una clave simétrica. Una vez creada la clave, cargará los grupos y generará los Macaroons que solicitarán las entidades. Al Macaroon se le añadirá, un TPC donde la condición ira encriptada con la clave establecida anteriormente con GKD, además de unas cabeceras adicionales que contendrán los nombres de las claves públicas DSK que necesitará tener la entidad para poder solicitar el DM. Acto seguido se queda esperando recibir un evento en los prefijos de los nombres establecidos.

Un miembro del grupo RRHH, Consumer, podrá solicitar los datos de un contrato a Producer. Producer enviará los datos cifrados con una clave de datos específica para el grupo RRHH que Consumer no tiene. Antes de descifrar los datos contenidos en el Data, Consumer validará la firma para comprobar la autenticidad del Data. Consumer solicitará a Producer la clave de datos del grupo RRHH. Producer no puede enviar la clave de datos de manera confidencial (cifrada con la clave de grupo de RRHH) porque aún no le han informado de dicha clave de

grupo. Por tanto Producer contestará con un Data que contiene en el nombre el campo notEncrypted que indica que aún no puede mandarle la clave de datos cifrada.

Para conseguir la clave de grupo Consumer pide el Macaroon a AC. AC recibirá el Interest y comprobará que el grupo del que se solicita el Macaroon existe, y se lo enviará contenido en un Data. Una vez recibido y validado el Data que contiene el Macaroon, Consumer empezará a descomponer el Macaroon, los endorsement o cabeceras en las que se encuentran los nombres de las claves DSK que necesitará para cifrar las claves de sesión entre AC y GKD. En este caso las claves DSK de AC y GKD. A partir de estos nombres de claves, Consumer pedirá las claves DSK. Tanto AC y GKD recibirán los Interest solicitando las claves DSK, que enviarán a Consumer a través de un Certificado. Este certificado debe estar firmado por una entidad confiable. Si ya posee las claves DSK Consumer no las solicitará. Consumer revisará la TPC del Macaroon. Consumer enviará un paquete Interest a GKD solicitando el DM que permita verificar el TPC. El nombre del Interest contendrá: una clave de sesión cifrada con la clave DSK de GKD, condición cifrada a verificar y grupo.

GKD valida el paquete Interest recibido. A través de la firma GKD conocerá la identidad del autor del Interest. Extraerá los datos contenidos en el nombre, descifrará con su clave pública DSK la clave de sesión, y también extraerá el nombre del grupo y la condición cifrada.

La condición viene cifrada con la clave de sesión intercambiada al principio entre AC y GKD. Ya que nadie excepto ellos debe saber la condición impuesta, para así poder evitar una posible modificación. GKD tiene ya toda la información necesaria para poder comprobar la condición. La condición impuesta por AC es que el usuario debe pertenecer a un grupo en concreto. Si se cumple la condición, GKD revisará si ya tiene creado un DM para ese grupo. Si no, lo creará y lo almacenará, lo cifrará con la clave de sesión extraída del nombre del Interest y lo añadirá al contenido del paquete Data. Modificará el nombre añadiendo authorized al paquete Data de respuesta, para indicar que ha sido autorizado. Por último, firmará el paquete y lo enviará.

Cuando Consumer reciba el Data deberá validar la firma. Si todo está correcto, comprobará el último apartado del nombre, para saber si ha sido autorizado o no. Si ha sido autorizado pasará a extraer y descifrar el contenido del paquete, obteniendo así el DM.

La clave de grupo, será la HMAC del DM extraído. Si Consumer ya posee la clave de datos, intentará descifrarla con la clave de grupo obtenida, obtendrá la

clave de datos y descifrará los datos. Si no la posee, Consumer enviará un Interest a AC, incluyendo en el nombre una clave de sesión cifrada con la clave DSK de AC, el Macaroon, el DM encriptado con la clave de sesión, el dataName cifrado, grupo y versión. Este Interest es necesario que vaya firmado, para su futura validación. Esta comunicación para establecer la clave de grupo se puede ver en la figura 3.2.

AC recibirá el Interest firmado con el Macaroon y el DM, y validará el paquete. Si es validado con éxito se extraerá cada uno de los datos nombrados anteriormente incluidos en el nombre del paquete Interest. Como se aprecia en el paso 2 y 3 de la figura 3.2, AC utilizará el Macaroon más el DM para verificar la integridad del Macaroon y validar las condiciones. Verificará el TPC y comprobará que la entidad que realiza la petición se ha autenticado con éxito en GKD. AC extraerá la clave de grupo del DM verificado, y utilizará el dataname incluido en nombre de la petición recibida para identificar al productor al que tiene que enviar la clave. La clave de grupo es enviada en un paquete Interest firmado. La clave irá cifrada por la clave pública DSK de Producer.

Cuando Producer reciba el Interest con la clave de grupo cifrada, comprobará el comando recibido en el nombre. Al recibir el comando setKeyGroup para establecer una contraseña de grupo, éste validará el paquete y si todo está correcto, extraerá y descifrará con su clave privada DSK la clave de grupo. Si ya existe clave para ese grupo, descartará la clave y responderá con un Data, añadiendo al nombre el resultado de la petición. Si no existe la clave de grupo, la clave será almacenada, cifrará la clave de datos y enviará un Data como respuesta a AC confirmando que la clave ha sido establecida. Producer ya puede enviar de manera confidencial la clave de datos cifrada con la clave de grupo a cualquier entidad. Solo miembros del grupo que se hayan autenticado previamente con GKD podrán descifrar la clave de datos, porque sólo ellos tienen la clave de grupo.

Capítulo 4
Desarrollo

En este capítulo se explicarán las librerías utilizadas para el desarrollo del sistema de delegación de permisos para el acceso a los datos definido en el capítulo 3. También se explicará como se ha desarrollado el ejemplo del sistema diseñado.

4.1. Instalación de librerías y paquetes

El sistema de control de acceso a recursos con Macaroons ha sido probado sobre el sistema operativo Ubuntu 16.04 LTS, siendo necesarios los siguientes requisitos para su funcionamiento:

- ndn-cxx[11]: contiene la librerías necesarias para poder desarrollar aplicaciones para la arquitectura de red NDN. La versión utilizada es 0.4.1. Para la instalación seguir los pasos indicados en el siguiente enlace:

  http://named-data.net/doc/ndn-cxx/0.4.1/INSTALL.html

  En el siguiente repositorio se encontrará la versión de la librería especificada arriba:

  https://github.com/named-data/ndn-cxx/releases/tag/ndn-cxx-0.4.1

- Networking Forwarding Daemon(NFD): este demonio simula un nodo de la arquitectura NDN dentro de nuestro sistema operativo. Es necesario para poder probar las aplicaciones basadas en la arquitectura de red NDN. La versión de NFD a instalar se descargará del siguiente repositorio:

  https://github.com/named-data/NFD/releases/tag/NFD-0.4.1

Se instalará el NDF desde el repositorio siguiendo los pasos indicados en el siguiente enlace:

http://named-data.net/doc/NFD/0.4.1/INSTALL.html

- libMacaroons: el siguiente repositorio contiene las librerías necesarias para crear, modificar y verificar los Macaroons:

  https://github.com/rescrv/libMacaroons

  Durante el desarrollo del proyecto la librería fue modificada por su autor generando versiones posteriores incompatibles con la que se ha utilizado en este trabajo. El proyecto está probado con la versión disponible en el siguiente commit del repositorio:

  commit d93f3b3b7f7c047f10bf9982ad64296ccfbb254b
  Author: Robert Escriva <robert@rescrv.net>
  Date: Thu Jun 30 15:55:13 2016 -0400

- protobuf[13]: esta librería permite encapsular datos para facilitar el envío de datos por la red. En este proyecto es utilizada para encapsular los Macaroon permitiendo añadir información adicional. La librería se encuentra en el siguiente repositorio:

  https://github.com/google/protobuf/tree/master/src

4.2. Librería ndn-cxx

Esta librería es la encargada de implementar los métodos necesarios para realizar aplicaciones que funcionen sobre la arquitectura de red NDN.

4.2.1. Envío, recepción y filtrado de paquetes

La clase face.hpp permitirá la creación de Interest y Data, así como su envío, recepción y filtrado de estos, según sus nombres.

La creación de objetos Interest permitirá hacer la petición de los datos a través de la red. A continuación se describirán los métodos de la clase Interest.

- El constructor:

```
Interest interest(const Name& name);
```

  — name: nombre de los datos que se van a solicitar.

El sistema de control de acceso a recursos utilizará este constructor para crear el Interest que utilizaremos para hacer la petición de los datos.

- El siguiente método extraerá el nombre del recurso solicitado en el paquete Interest.

```
const Name&
getName()const;
```

- Este método establece el tiempo de vida del paquete Interest.

```
Interest&
setInterestLifetime(const::milliseconds& interestLifetime);
```

  — interestLifetime: tiempo de vida que tendrá el Interest en milisegundos.

- El siguiente método establecerá la opción del paquete Interest que indicará a los routers que el paquete Data debe provenir directamente del productor.

```
Interest&
setMustBeFresh(bool mustBeFresh);
```

  — mustBeFresh: argumento para indicar que el paquete debe provenir directamente del productor.

Los objetos Data permitirán el envío de los datos solicitados a cualquiera de las entidades a través de la red.

```
Data data;
```

El constructor permitirá la creación del objeto sin pasarle ningún argumento.

A continuación se describirán los métodos de la clase Data.

- El siguiente método se utiliza para especificar el nombre de los datos

```
Data&
setName(const Name& name);
```

  — name: nombre del los datos que se van a enviar, en respuesta al Interest.

- El método siguiente establecerá el tiempo de validez del paquete Data.

```
Data&
setFreshnessPeriod(const time::milliseconds& freshnessPeriod);
```

— freshnessPeriod: tiempo de validez del Data.

- El siguiente método devolverá el bloque de bits que contienen los datos incluidos en el paquete Data.

```
Block&
getContent();
```

- Este método permitirá incluir los datos solicitados en el paquete Interest dentro del paquete Data.

```
Data&
setContent(const ConstBufferPtr& contentValue);
```

— contentValue: datos que irán incluidos en el paquete.

La clase Face permitirá el envío de Interest y Data a través de la red, así como quedarse esperando eventos en un determinado prefijo o nombre.

```
Face face;
```

El objeto face generado permitirá el uso de los siguientes métodos.

- Este método permitirá estar esperando eventos en los prefijos que se le asigne, tratando el Interest de un modo concreto.

```
const RegisteredPrefixId*
setInterestFilter(const InterestFilter& interestFilter,
        const InterestCallback& onInterest,
        const RegisterPrefixFailureCallback& onFailure,
        const security::SigningInfo& signingInfo = security::signingInfo(),
        uint64_t flags = nfd::ROUTE_FALG_CHILD_INHERIT);
```

— interestFilter: nombre o prefijo en el que se esperará eventos.

— onInterest: función o método que se ejecutará cuando aparezca un evento en el prefijo o nombre asignado.

— onFailure: función o método que se ejecuta si el Interest tenía un comando a ejecutar y el comando no es válido.

— signingInfo: información de la firma del paquete en caso de que la contenga.

— flags: valores para establecer el filtro.

- Permitirá el envío del Interest, quedando a la espera de recibir el Data pertinente durante el período de vida del Interest, en caso de ser recibido el Data se procesará según el método o función dado, mientras que si el tiempo de vida del Interest expira sin recibir el Data, éste ejecutará otro método o función, para gestionar la situación.

```
const PendingInterestId*
expressInterest( const Interest& interest,
        const OnData& onData,
        const OnTimeout& onTimeout = nullptr);
```

— interest: paquete Interest a enviar.

— onData: función o método que procesará el Data una vez recibido.

— onTimeout: función o método que se ejecutará una vez el tiempo de vida del Interest ha expirado y no se ha recibido el Data.

- Este método permite el envío de paquetes Data a través de la red.

```
void
put(const Data& data);
```

— data: paquete Data a enviar con los datos solicitados.

- Método necesario para indicar al objeto que tiene que quedarse esperando eventos.

```
void
processEvents();
```

4.2.2. Gestión de identidades, certificados y firmas

La librería Key-Chain.hpp, permite crear, modificar y administrar las identidades y certificados necesarios para trabajar con las firmas. En el sistema diseñado las claves e identidades ya han sido creadas previamente, por lo que la librería solo será utilizada para la administración de las identidades y certificados. Las identidades de cada entidad del sistema de delegación serán extraídas del fichero keys.txt, donde se encontrará el nombre completo tanto de la clave KSK, como de la clave DSK (Vease en la sección 4.4).

Se declarará el objeto KeyChain, que permitirá utilizar los métodos necesarios para poder trabajar con las identidades y certificados.

```
KeyChain keyChain;
```

Los métodos que utilizaremos serán los siguientes:

- Este método getDefaultCertificateNameForKey devolverá un Name con el nombre completo del certificado, que será utilizado como argumento en funciones futuras.

```
Name
getDefaultCertificateNameForKey(const Name& keyName) const;
```

  — keyName: es el nombre de la clave, necesario para encontrar el certificado asociado a esa clave.

- El método getCertificate devuelve un puntero a un objeto el cuál contiene el certificado solicitado.

```
shared_ptr<v1::IdentityCertificate>
getCertificate(const Name& certificateName);
```

  — certificateName: es el nombre del certificado, necesario para buscarlo y devolverlo.

- El siguiente método sign firmará el paquete que se le pasa como argumento. La firma se realiza a través del Certificado.

```
template<typename T>
void
sign(T& packet, const Name& certificateName);
```

  — packet: es el objeto Data o Interest que se va a firmar.

  — certificateName: es el nombre del certificado, necesario para buscarlo y poder firmar el paquete.

- El método signByIdentity firmará el paquete que se le pasa como argumento. La firma se realizará con el certificado por defecto asociado a la identidad, es decir, una identidad puede tener más de una clave KSK o DSK, para ser utilizadas en diferentes aplicaciones, pero una de ellas estará asociada por defecto a la identidad, siendo ésa con la que se firmarán los paquetes si no se especifica una concreta.

```
template<typename T>
void
signByIdentity(T& packet, const Name& identityName);
```

  — packet: es el objeto Data o Interest que se va a firmar.

  — identityName: es el nombre de la identidad, necesaria para firmar el paquete.

La firma de paquetes ha de realizarse justo antes del envío del Interest o Data. La firma será diferente en función de los datos contenidos en el paquete, si cualquiera de los componentes se modifica después de firma, la firma dejará de ser válida.

4.2.3. Validación de paquetes

La clase validator-config.hpp hereda de validator.hpp y permite cargar, modificar o crear reglas para la validación de paquetes Interest y Data. El objeto permitirá utilizar y gestionar las claves.

```
ValidatorConfig validator;
```

Los métodos a utilizar serán los siguientes:

- El método load solo se le pasará como argumento el nombre del archivo de configuración de validación, con las reglas que se aplicarán a los paquetes entrantes, para poder considerarlos de confianza.

```
void
load(const std::string& filename);
```

- Los siguientes dos métodos permiten verificar la integridad del paquete entrante. Los paquetes deben cumplir las reglas previamente creadas o cargadas de un fichero. Si la firma cumple las condiciones impuestas llamará a la función indicada en onInterestValidated o OnDataValidated, en caso contrario se llamará a la función indicada en OnInterestValidationFailed o OnDataValidationFailed.

```
void
validate(const Data& data,
    const OnDataValidated& onValidated
    const OnDataValidationFailed& onValidationFailed);
```

```
void
validate(const Interest& interest,
    const OnInterestValidated& onValidated
    const OnInterestValidationFailed& onValidationFailed);
```

  — OnDataValidated o OnInterestValidated: función a la que se llamará si la firma ha sido validada correctamente

  — OnDataValidationFailed o OnInterestValidationFailed: función a la que se llamará si no ha podido validarse la firma.

4.3. libNDNMacararoon y modificaciones

Esta librería contiene las clases y métodos para crear los Macaroons y gestionar las claves simétricas necesarias para establecer intercambios de información seguros entre diferentes partes. Se ha de tener en cuenta que la funcionalidad de los Macaroons en este proyecto será utilizada a través de ella. A continuación se describirá brevemente la librería así como las funciones utilizadas en el código para implementar este proyecto.

4.3.1. Macaroon.hpp y Macaroon.cpp

A continuación se describen las diferentes formas que la librería libNDNMacaroon presenta para construir un Macaroon:

Este constructor creará el Macaroon nuevo, a partir de la información recibida:

```
NDNMacaroon
(std::string location,
uint8_t* key,
uint8_t* id,
size_t id_size);
```

- location: será la localización del creador del Macaroon, ya que éste es el único capaz de validarlo. En el ejemplo, el creador del Macaroon será: /example/accesscontroller/
- key: será una clave secreta que ha de conocer el creador del Macaroon, la cuál será utilizada para generar la HMAC necesaria para verificar el Macaroon más adelante. Sólo el creador del Macaroon puede verificar que el Macaroon es correcto.
- id: identificador único del Macaroon a crear y que junto con la clave generará la primera HMAC.
- id_size: tamaño de la constante id;

El siguiente constructor nos permitirá reconstruir un Macaroon que haya sido serializado para su transporte por red.

```
NDNMacaroon(std::string serialized);
```

- serialized: es un string con la información necesaria para recrear un Macaroon ya existente, con todos sus datos y cabeceras que previamente han sido serializadas para enviarlas por la red.

A continuación se describen los métodos que la librería NDNLibMacaroon ofrece para acceder a los campos del Macaroon.

- El método addFirstPartyCaveat añadirá las condiciones que se encargara de verificar el propio creador del Macaroon, así como tiempo de validez de éste.

```
void
addFirstPartyCaveat(std::string caveat);
```

  — caveat: string que contiene la condición a cumplir.

- El método getLocation permitirá extraer la localización del Macaroon, y la devolverá en un string.

```
std::string getLocation();
```

- Este método addThirdPartyCaveat permite añadir la condición a validar por la tercera parte.

```
void
addThirdPartyCaveat
(std::string tp_location,
std::string predicate,
uint8_t* caveat_key,
const Encryptor& encryptIdentifier);
```

  — tp_location: localización de la Third Party necesaria para validar la condición, incluido en TPC.

  — predicate: string con la condición que tiene que validar la tercera parte, en nuestro sistema la tercera parte será GKD.

  — caveat_key: nombre de la clave compartida con la que se cifrará la condición.

  — encryptIdentifier: clave compartida con la que se cifrará la condición.

- El método getThirdPartyCaveat se encarga de extraer el TPC, será utilizado por Consumer para enviar a GKD la condición a validar.

```
void
getThirdPartyCaveat
(unsigned n,
std::string& third_party_location,
ndn::ConstBufferPtr* tp_id_sp) ;
```

  — n: indica el número de TPC que se quiere extraer, ya que pueden existir varios TPC a validar dentro de éste.

  — third_party_location: variable en la que se devolverá la localización de la tercera parte.

  — tp_id_sp: devuelve un puntero a un buffer con el TPC.

- El método getNumThirdPartyCaveats devolverá el número total de TPCs que contiene el Macaroon.

```
unsigned getNumThirdPartyCaveats();
```

A continuación se describen los métodos que permiten manejar el DM.

- El método addDischarge añade el DM al Macaroon, sin llegar a modificar la HMAC del Macaroon. Esto es necesario para la verificación del Macaroon, ya que esta parte es añadida por AC antes de validar el Macaroon.

```
void
addDischarge(std::string d);
```

  — d: un DM serializado, previamente obtenido de la tercera parte, que es GKD.

- El método addDischargeAndPrepare añade el DM al Macaroon, preparándolo y modificando la última HMAC, para su futura verificación.

```
void
addDischargeAndPrepare(std::string d);
```

  — d: un DM serializado, previamente obtenido de la tercera parte.

- El método getDischargeMacaroon devolverá un DM serializado. A partir del DM se obtendrá la clave de grupo.

```
std::string
getDischargeMacaroon(unsigned i);
```

  — i: indica el DM que queremos extraer, en caso de que haya más de uno.

- El método getNumDischargeM devolverá el número total de DM que contiene el Macaroon.

```
unsigned
getNumDischargeM();
```

- El método verify permitirá la validación y verificación del Macaroon, devolviendo un entero en función del resultado de la validación. Si la operación ha sido exitosa el método devolverá un 0, en caso contrario devolverá un 1.

```
int
verify(NDNMacaroonVerifier* V,
      uint8_t* secret,
      std::string& errorCode);
```

— V: objeto NDNMacaroonVerifier que contiene las reglas que ha de validar el Macaroon.

— secret: clave secreta con la que se creó el Macaroon.

— errorCode: string donde devuelve el error ocurrido en caso de que no haya sido posible la validación del Macaroon.

Esta librería no solo permite construir el objeto NDNMacaroon, también permite crear el objeto NDNMacaroonVerifier necesario para la validación y verificación del Macaroon. NDNMacaroonVerifier contiene las reglas necesarias para la validación del Macaroon.

El constructor del objeto NDNMacaroonVerifier es el siguiente:

```
NDNMacaroonVerifier();
```

El método del objeto utilizado para la verificación de las condiciones del Macaroon será satisfyGeneral, para modificar el objeto y establecer una referencia que será utilizada después por el objeto NDNMacaroon para la verificación del Macaroon a través de la función NDNMacaroon::Verify.

```
void
satisfyGeneral
(int(*general_check)(void *f, const unsigned char* pred, size_t pred_sz),
void* f);
```

Al método satisfyGeneral se le pasa como argumento el puntero a la función necesaria para la verificación de la condición impuesta.

4.3.2. Modificaciones realizadas a libNDNMacaroon

A continuación se describen las modificaciones realizadas a la librería.

La librería no contaba con un método capaz de extraer la HMAC. La HMAC es la firma del Macaroon. Se creó el siguiente método en la librería, necesario para extraer la HMAC que es utilizada como clave de grupo del DM. El método devolverá la HMAC en un string.

```
std::string getSignature(){
  const unsigned char* signature;
  size_t sig_sz;

  macaroon_signature(M, &signature, &sig_sz);
  return std::string(signature, signature + sig_sz);
}
```

4.3.3. Macaroon-utils.hpp y Macaroon-utils.cpp

De esta parte de la librería se utilizará la siguiente función que permite crear la clave de sesión cifrada. La clave es necesaria para compartir datos de manera segura.

```
ndn::ConstBufferPtr
generateSessionKey
(ndn::Name& public_key_name,
ndn::Name& session_key_name,
size_t key_size) ;
```

- public_key_name: clave pública con la que se cifra la clave de sesión.
- session_key_name: nombre con el que será almacenada la clave de sesión generada.
- key_size: tamaño de la clave de sesión a generar.

4.3.4. sec-tmp-file-enc.hpp y sec-tmp-file-enc.cpp

En estos ficheros se permite la creación de un objeto capaz de manejar las claves públicas, privadas y claves simétricas.

Los dos métodos más utilizados serán los que se usan para encriptar y desencriptar datos. El primer método encriptará los datos que se le pasan como argumento. Devuelve un puntero a un buffer con los datos encriptados. El segundo método recibirá los datos encriptados y nos devolverá los datos desencriptados en un puntero a un buffer, si tenía los medios necesarios para poder desencriptarlos.

- El método encryptInTmp:

```
virtual ConstBufferPtr
encryptInTmp( const uint8_t data,
      size_t dataLength,
      const Name& keyName,
      bool isSymmetric);
```

  — data: datos no encriptados que se quieren encriptar.

  — dataLength: tamaño de los datos que se le pasan como parámetro.

  - keyName: nombre de la clave, necesario para buscarla y poder cifrar los datos.
  - isSymmetric: indica si se va a utilizar una clave simétrica para cifrar los datos, o por el contrario utilizará una clave pública para ello.

- El método decryptInTmp:

```
virtual ConstBufferPtr
decryptInTmp( const uint8_t data,
      size_t dataLength,
      const Name& keyName,
      bool isSymmetric);
```

  - data: datos encriptados que se quiere desencriptar.
  - dataLength: tamaño de los datos que se le pasan como parámetro.
  - keyName: nombre de la clave, necesario para buscarla y poder descifrar los datos.
  - isSymmetric: indica si se va a utilizar una clave simétrica para desencriptar los datos, o por el contrario utilizará un clave pública para ello.

El método setSymmetricKeyToTmp almacena la clave simétrica de forma segura. El nombre identificativo de la clave debe ser guardado para poder recuperar la clave. Este nombre es el que se utilizará en otras funciones para poder encriptar o desencriptar con la clave simétrica almacenada.

```
virtual void
setSymmetricKeyToTmp( const Name& keyName,
          const uint_t* keybits,
          size_t key_size);
```

- keyName: nombre con el que se almacenará la clave simétrica y a través del cuál se accederá a la misma.
- keybits: es la clave.
- key_size: tamaño de la clave.

El método doesKeyExistInTpm comprueba la existencia de la clave, verificando si existe el nombre de la clave que le pasamos. En caso de que exista devolverá un boolean con el valor true, mientras que en caso contrario devolverá false.

```
virtual bool
doesKeyExistInTpm( const Name& keyName,
        KeyClass keyClass);
```

- keyName: nombre que será utilizado para comprobar si existe la clave.
- keyClass: tipo de clave que se va a comprobar, pública, privada o simétrica.

Genera un bloque de bits de manera aleatoria. Este bloque de bits obtenido es utilizado como clave o identificador. Devolverá true si la operación ha salido bien y false en caso contrario.

```
virtual bool
generateRandomBlock( uint8_t* res,
        size_t size);
```

- res: devolverá el bloque de bits generado de forma aleatoria.
- size_t: tamaño del bloque de bits.

4.4. Generación de claves KSK y DSK

Para que cada entidad del sistema pueda firmar paquetes Interest y Data, y cifrar claves de sesión, se necesitarán crear las claves KSK y DSK para dicha tarea. Para crear las claves necesarias se utilizarán los siguientes comandos:

- La orden ndnsec-key-gen permite crear las claves KSK y DSK, por defecto, si no se le indica nada, las claves generadas serán las claves KSK. En el caso de poner la opción -d le indicaremos que queremos generar las claves DSK. La opción -n permite indicar que no queremos que la identidad generada no se establezca como la identidad por defecto. El nombre de la identidad estará definido por las reglas establecidas. En este caso /ndn/keys/<entidad>.

```
$ndnsec-key-gen -n -d <nombre identidad>
```

- La orden ndnsec-sign-req genera un certificado firmado. En el caso de no indicarle nada, será autofirmado por la propia identidad.

```
$ndnsec-sign-req <nombre de la clave>
```

- La orden ndnsec-cert-install permite instalar un certificado en PublicInfo, pasándole como argumento el certificado a instalar.

```
$ndnsec-cert-install <certificado.cert>
```

- La orden ndnsec-cert-gen generará un certificado firmado por una identidad. La opción -s permite añadir la identidad que se utilizará par firmar el certificado. La opción -N permite añadir un nombre o descripción. La última opción -r permite seleccionar el certificado que se desea firmar. La salida será un certificado firmado.

```
$ndnsec-cert-gen -s <identidad> -N <definición> -r <certificado>
```

- La orden ndnsec-dump-certificate permite mostrar el certificado en la consola. La opción -p nos muestra el certificado para que sea posible leerlo por un humano. La opción -f indicada que lo ha de obtener de un fichero que se le pasará como argumento.

```
$ndnsec-dump-certificate -p -f <certificado.cert>
```

La primera identidad que se generara será la de la autoriadad de confianza, con la que firmaremos el resto de certificados, haciendo que sean fiables.

```
$ndnsec-key-gen -n /ndn/keys/trust
$ndnsec-sign-req /ndn/keys/trust > trust.cert
```

Se creará la identidad y acto seguido, se autofirmará el certificado, guardando la salida en el fichero trust.cert.

Los siguientes pasos serán crear las identidades de cada entidad, tanto las claves KSK como DSK, de Producer, AC, GKD y Consumer.

```
$ndnsec-key-gen -n /ndn/keys/<entidad> > <entidad>-unsigned.cert
$ndnsec-cert-install <entidad>-unsigned.cert
$ndnsec-cert-gen -s /ndn/keys/trust -N "<entidad>"
  -r <entidad>-unsigned.cert > <entidad>-ksk.cert
$ndnsec-cert-install <entidad>-ksk.cert
$ndnsec-key-gen -d /ndn/keys/<entidad> > <entidad>-dsk.cert
$ndnsec-cert-install -f <entidad>-dsk.cert
```

Se crearán en primer lugar las claves KSK, guardando salida en un fichero. El fichero creado es un certificado sin firmar, será instalado y posteriormente firmado por la identidad de la autoridad de confianza. La salida se guarda en otro fichero. El nuevo fichero es un certificado firmado y será instalado. Una vez creada la clave KSK con un certificado firmado, se creará la clave DSK, que será instalada sin la necesidad de ser firmada. Cuando se comparta la clave DSK será través de un paquete cifrado y firmado. Cuando las claves KSK y DSK de cada entidad estén creadas, el nombre de cada clave es guardado en el fichero keys.txt siguiendo el siguiente orden: Producer, Consumer1, Consumer2, AC y GKD.

Este fichero será leído por cada entidad para conocer los nombres de las claves con las que se trabaja en el sistema.

4.5. Ficheros de validación de paquetes

Los ficheros de validación se definen dentro de la arquitectura NDN para definir las reglas con los que tienen que venir firmados los paquetes Data y si fuera necesario también los paquetes Interest.

El fichero de validación para Producer es el siguiente, validation-producer.conf:

```
rule
{
  id "Validator Access Controller"
  for interest
  filter
  {
    type name
    name /example
    relation is-prefix-of
  }
  checker
  {
    type fixed-signer
    sig-type rsa-sha256
    signer
    {
      type file
      file-name ./accesscontroller-ksk.cert
    }
  }
}
```

El único paquete que tiene que validar Producer, es el Interest que recibe de AC para establecer la clave de grupo necesaria para cifrar la clave de datos. Validará el Interest que produzca un evento en el prefijo /example, y solo será validado, si el paquete ha sido firmado por AC, en caso contrario la validación fallará.

El fichero de validación utilizado por AC, es validation-ac.conf:

```
rule
{
  id "Validation Interest Entity"
  for interest
  filter
  {
    type name
    name /example
    relation is-prefix-of
  }
  checker
  {
    type fixed-signer
    sig-type rsa-sha256
    signer
    {
      type file
      file-name ./consumer1-ksk.cert
```

```
    }
  }
}

rule
{
  id "Validation Data Entity"
  for data
  filter
  {
    type name
    name /example
    relation is-prefix-of
  }
  checker
  {
    type fixed-signer
    sig-type rsa-sha256
    signer
    {
      type file
      file-name ./groupkeysdistributor-ksk.cert
    }
    signer
    {
      type file
      file-name ./producer-ksk.cert
    }
  }
 }
```

El fichero de validación está compuesto por dos reglas. La regla Validation Interest Entity validará los Interest recibidos por Consumer1 al prefijo /example de AC. La regla Validation Data Entity validará los paquetes Data recibidos por GKD y Producer en el prefijo /example. Ambas reglas validarán las firmas a través de los certificados de las entidades.

El fichero de validación utilizado por AC, es validation-gkd.conf:

```

rule
{
  id "Validation Entity"
  for interest
  filter
  {
    type name
    name /example
    relation is-prefix-of
  }
  checker
  {
    type fixed-signer
    sig-type rsa-sha256
    signer
    {
      type file
      file-name ./consumer1-ksk.cert
```

```
    }
    signer
    {
      type file
      file-name ./accesscontroller-ksk.cert
    }
  }
}
```

La única regla es Validation Entity, esta regla comprobará que los Interest recibidos en /example son de Consumer1 y de AC. Validará la firma a través de los certificados de las entidades.

El fichero de validación utilizado por Consumer, será validationConsumer.conf:

```
rule
{
  id "Validation Data Entity"
  for data
  filter
  {
    type name
    name /example
    relation is-prefix-of
  }
  checker
  {
    type fixed-signer
    sig-type rsa-sha256
    signer
    {
      type file
      file-name ./accesscontroller-ksk.cert
    }
    signer
    {
      type file
      file-name ./groupkeysdistributor-ksk.cert
    }
    signer
    {
      type file
      file-name ./producer-ksk.cert
    }
  }
}

rule
{
  id "Validation Cert Entity"
  for data
  filter
  {
    type name
    name /ndn/keys
    relation is-prefix-of
  }
  checker
  {
```

```
    type fixed-signer
    sig-type rsa-sha256
    signer
    {
      type file
      file-name ./accesscontroller-ksk.cert
    }
    signer
    {
      type file
      file-name ./groupkeysdistributor-ksk.cert
    }
  }
}
```

El fichero consta de dos reglas. La regla Validation Data Entity validará los paquetes Data que en el nombre contengan el prefijo /example, de las solicitudes realizadas a las entidades Producer, AC y GKD. La regla Validation Cert Entity validará los paquetes Data recibidos con el prefijo /ndn/keys. Estos paquetes contendrán los certificados solicitados a las entidades. Las firmas en ambas reglas se validarán a través de los certificados de las entidades.

4.6. Entidades que participan en el sistema de autorización con Macaroons

Ahora se explicarán las partes fundamentales del código implementado en cada una de las entidades que forman el sistema de autorización con Macaroons. Solo se verán las partes más relevantes del código, para poder consultar el código completo se puede ver en el siguiente repositorio:

https://github.com/djuanes87/PFG-NDNMacaroons

4.6.1. Producer

Este programa se encarga de publicar datos, y para ello realiza una serie de tareas necesarias para garantizar que los datos se envían de manera segura.

Lo primero que hará esta aplicación, será crear el objeto Producer. El constructor del objeto generará las claves de datos, cifrando los datos utilizados en este ejemplo y que serán solicitados más adelante. Por último cargará del fichero keys.txt las claves KSK y DSK generadas anteriormente.

La función run() deja el objeto Producer esperando un evento en los prefijos indicados:

- /example/producer/dptoRRHH/contracts/

En el momento que se produzca un evento, se pasará a ejecutar el método onInterest. Obtendrá como argumento un Interest del que extraerá el nombre. El comando será extraído del nombre.

```
void
onInterest(const InterestFilter& filter,
    const Interest& interest)
{
  Name interestName = interest.getName();
  std::string command = interestName.at(COMMAND_POS).toUri();
  if(command == "getKeyData"){
    getKeyData(interest);
  }else if(command == "setKeyGroup"){
    m_validator.validate(interest,
                         bind(&Producer::setKeyGroup, this, interest),
                         bind(&Producer::onValidationFailed, this, _1, _2));
  }else{
    sendEncryptedData(interest);
  }
}
```

Recibida la orden getKeyData se ejecutará la función con el mismo nombre pasándole el Interest recibido como argumento. Se comprueba si Producer tiene la clave de grupo. Si tiene la clave de grupo enviará un paquete Data con la clave de datos cifrada. En caso de que no tenga la clave de grupo enviará un paquete Data indicando que no es posible enviar de manera segura la clave de datos.

```
void
getKeyData(const Interest& interest){
  Name dataName = interest.getName();

  if (!m_secTpmFile.doesKeyExistInTpm(key_group_name,
                                      KEY_CLASS_SYMMETRIC))
  {
    dataName.append("notEncrypted");
    sendData(dataName, NULL);
  }else{
    dataName.append("keyDataEncrypted");
    sendData(dataName, enc_key_data);
  }
}
```

En caso de que la orden recibida sea setKeyGroup. Validará el paquete Interest. El Interest contiene en el nombre información importante. Solo será validado el paquete Interest si está firmado por AC, si no el paquete será descartado. Si el paquete es validado, se ejecutará la función setKeyGroup pasándole el Interest como argumento. Esta función comprobará si ya existe una clave para el grupo. Si la clave de grupo ya ha sido establecida con anterioridad se desechará la clave nueva. Enviará un Data para hacerle saber que la clave ya existía. Si no existe, ésta será almacenada y acto seguido cifrará la clave de datos con la clave de grupo. Enviará un Data, haciendo saber a la entidad que realizó la petición que todo está correcto.

```
void
setKeyGroup(const Interest& interest){
Name dataName = interest.getName();

  if (!m_secTpmFile.doesKeyExistInTpm(key_group_name,
                                      KEY_CLASS_SYMMETRIC))
  {
    storeKeyGroup(interest.getName().at(KEY_GROUP_POS));
    encryptKeyData();
    dataName.append("sucefull");
  }else{
    dataName.append("alreadyExist");
  }
  sendData(dataName, NULL);
 }
```

Si el comando extraído del nombre, no pertenece a ninguno de los anteriores, se asumirá que se están pidiendo los datos referenciados con el nombre del Interest. Ejecutará la función sendEncryptedData, con el Interest recibido como argumento. La función se encargará de comprobar la existencia de los datos. En ambos casos enviará un Data. Si no encuentra el dato solicitado el Data estará vacío, indicando que no existen los datos solicitados. Si los datos existen se enviarán encriptados dentro del paquete Data.

```
void
sendEncryptedData(const Interest& interest){
  std::map<std::string, ndn::ConstBufferPtr>::iterator it;

  Name dataName = interest.getName();
  ndn::ConstBufferPtr enc_data;
  std::string name_data = interest.getName().at(COMMAND_POS).toUri();
  it = enc_files.find(name_data);
  if(it == enc_files.end()){
    dataName.append("notFound");
    sendData(dataName, NULL);
  }else{
    enc_data = enc_files[name_data];
    dataName.append("encryptedData");
    sendData(dataName, enc_data);
  }
}
```

4.6.2. Access Controller

Esta entidad es la encargada de proporcionar el Macaroon, para que las entidades consumidoras puedan solicitar la clave de grupo a GKD y descifrar los datos.

Al iniciar la entidad se creará el objeto AccessController. El constructor cargará las claves KSK y DSK de las entidades del sistema de control de acceso a recursos. Establecerá una clave simétrica con GKD, que enviará a través de un Interest firmado. Esta clave es necesaria para cifrar la condición contenida en las cabeceras

TPC de los Macaroons. La clase ManagesGroup ayudará a manejar la información de cada grupo. Se creará un objeto ManagesGroup con el nombre del grupo RRHH. El objeto RRHH contendrá los prefijos a los que tiene acceso el grupo y creará el Macaroon. El Macaroon está encapsulado por e_Macaroon, necesario para facilitar su transporte, además de incluir la información de las claves DSK que necesitará la entidad que solicita el Macaroon.

Cuando el objeto AccessController haya sido creado se ejecutará el método run(), el cuál dejará al objeto AccessController esperando un evento en los prefijos indicados:

- /example/accesscontroller/
- /ndn/keys/accesscontroller/

Cuando reciba un evento en /example/accesscontroller se procesará en el método onInterest. Extraerá el comando que se encuentra dentro del nombre del paquete Interest.Si el comando no está registrado, el paquete será descartado y el método no hará nada. Si el comando extraído coincide con alguno de los comandos utilizados por AC se ejecutará la función específica para ese comando.

```
void
onInterest(const InterestFilter& filter,
           const Interest& interest)
{
  std::string command = interest.getName().at(COMMAND_POS).toUri();
  if(command == "getMacaroon"){
    getMacaroon(interest);
  }else if(command == "updateGroupKey"){
    m_validator.validate(interest,
               bind(&AccessController::updateGroupKey, this, interest),
               bind(&AccessController::onValidationInterestFailed, this, _1, _2));
  }
}
```

Si el comando extraído es getMacaroon, se ejecutará el método con el mismo nombre pasándole el Interest recibido como parámetro. Extraerá el nombre del grupo del Interest y lo buscará en la lista de grupos que tiene. Si el grupo no existe, el grupo devuelto será NULL. En este caso se enviará un paquete Data sin contenido, modificando el nombre para informar que el grupo no existe. En caso de que el grupo exista, obtendrá del grupo el e_Macaroon, será serializado y almacenado en el Data, modificando el nombre para hacerle saber a la entidad que realiza la petición que el paquete contiene los datos solicitados.

```
void
getMacaroon(const Interest& interest){
  Name dataName = interest.getName();
  ndn::OBufferStream os;
```

```
  std::shared_ptr<ManagesGroup> g =
  getGroup(interest.getName().at(COMMAND_POS+1).toUri());
  if(g == NULL){
    dataName
      .append("MacaroonGroupNotFound")
      .appendVersion();
    sendData(dataName, NULL);
  }else{
    Macaroons::e_Macaroon  e = g->getMacaroon();
    e.SerializeToOstream(&os);
    dataName
      .append("result")
      .appendVersion();
    sendData(dataName, os.buf());
  }
}
```

Si el comando recibido es updateGroupKey, se ejecutará la función updateGroupKey pasándole el Interest como parámetro. La función extraerá del Interest el nombre del grupo. Buscará el grupo en la lista, devolviendo un Data sin contenido en caso de no encontrarlo, e indicando en el nombre del Data que el grupo no existe. En caso contrario extrae la información contenida en el nombre. La información extraída del nombre es: clave de sesión con la que está encriptado el DM, la cuál será descifrada y guardada con el nombre /sesssion-key-ac-consumer, el Macaroon y el DM encriptado. El objeto ManageGroup se encargará de verificar la autenticidad y validez del Macaroon a través de un método, junto con el DM encriptado, para comprobar si cumple las condiciones impuestas, pasándole a la función verificateMacaroon esos dos parámetros más el nombre de la clave con la que el DM se encuentra encriptada. Ésta devolverá True en caso de que todo esté correcto y False en caso contrario. En caso de False, se descarta el Interest. Si el Macaroon es verificado, se extraerá la clave de grupo del DM y del nombre del Interest se extraerá el dataname encriptado, siendo éste el prefijo de los nombres, verificado que el prefijo pertenece grupo, y enviando un Data de respuesta, indicando en el nombre que todo está correcto. Acto seguido envía un Interest, utilizando el prefijo obtenido del dataname extraído del nombre, y añadiéndole el comando para establecer la clave y la clave de grupo encriptada con la clave pública de Producer.

```
void
updateGroupKey(const Interest& interest){
  Name dataName = interest.getName();
  std::shared_ptr<ManagesGroup> g =
  getGroup(interest.getName().at(GROUP_POS).toUri());
  if(g == NULL){
    dataName
    .append("groupNotFound")
    .appendVersion();
    sendData(dataName, NULL);
  }else{
    ndn::Name session_key_name("/session-key-ac-consumer");
    ndn::name::Component encrypted_session_key =
```

```
    interest.getName().at(SESSION_KEY_POS);
    getSessionKeyFromInterest(encrypted_session_key, session_key_name);

    std::string serializated_Macaroon =
    interest.getName().at(Macaroon_POS).toUri();
    ndn::name::Component encrypted_dm = interest.getName().at(DM_POS);

    if(!g->verificateMacaroon(serializated_Macaroon,
                              encrypted_dm,
                              session_key_name)){
      ndn::Name public_key_name(m_princKeyNames[PRODUCER_DSK]);
      ndn::ConstBufferPtr enc_key_group =
      g->extractKeyGroup(encrypted_dm,
                         m_princKeyNames[PRODUCER_DSK],
                         session_key_name);
      ndn::name::Component encrypted_dataname =
      interest.getName().at(DATANAME_POS);
      ndn::ConstBufferPtr decrypted_dataname =
      m_secTpmFile.decryptInTpm(encrypted_dataname.value(),
          encrypted_dataname.value_size(),
                                      session_key_name,
                                        /*symmetric*/ true);
      std::string dataname_scope =
      std::string(decrypted_dataname->buf(),
              decrypted_dataname->buf() + decrypted_dataname->size());
      if(g->checkScope(dataname_scope)){
        dataName.append("authorized");
        sendData(dataName, NULL);
        sendKeyGroupProducer(dataname_scope,
                  enc_key_group,
                  ndn::Name(interest.getName()));
      }else{
        dataName.append("non_authorized")
        .appendVersion();
        sendData(dataName, NULL);
      }
    }
  }
}
```

4.6.3. Group Keys Distributor

Entidad encargada de validar las condiciones incluidas en las cabeceras TPC. Las entidades consumidoras le solicitan a GKD validar la condición para poder obtener el DM con la clave de grupo.

Los eventos recibidos serán procesados por la función onInterest, que validará los paquetes recibidos. Si los paquetes son validados con éxito, obtendrá el comando. Si el comando es setSharedSecret extraerá la clave compartida y la almacenará. Si GKD recibe la orden getDischargeMacaroon, procederá a procesar la condición recibida a través de la función proccessDischargeMacaroon.

```
void
onInterest(const InterestFilter& filter, const Interest& interest, bool validated)
{
  if(!validated){
    m_validator.validate(interest,
```

```
        bind(&GroupKeysDistributor::onInterest, this, filter, interest, VALIDATED),
        bind(&GroupKeysDistributor::onValidationFailed, this, _1, _2));
  }else{
    std::string command = interest.getName().at(COMMAND_POS).toUri();
    if ( command == "getDischargeMacaroon") {
      proccessDischargeMacaroon(interest);
    }else if(command == "setSharedSecret"){
      setSharedSecret(interest);
      ndn::Name dataName(interest.getName());
      dataName.append("established");
      sendData(dataName, NULL);
    }
  }
}
```

La siguiente función procesará la condición que recibirá en el Interest. Extraerá del nombre los datos necesarios: nombre de la clave compartida con AC, el componente identifier es condición incluida en TPC del Macaroon y nombre del grupo. Utilizará la firma para extraer la identidad de la clave con la que se firmó. La identidad le permitirá reconocer a la entidad consumidora. Una vez tenga todos los datos extraídos, comprobará que el grupo existe, y guardará la clave de sesión que establece con Consumer. El siguiente paso es comprobar la condición incluida en el TPC y que será descifrada con la clave compartida entre AC y GKD, a través de la función checkPredicate. La condición impone que el usuario pertenezca al grupo en concreto, para poder obtener la clave. Utilizará la identidad que ha obtenido anteriormente para ver si está asociada a algún miembro del grupo. Si el usuario pertenece al grupo, obtendrá el DM que cifrará y enviará en el contenido de un Data. En caso de que no sea miembro del grupo, se enviará un paquete Data indicando en el nombre que no está autorizado.

```
void
proccessDischargeMacaroon(const Interest& interest){

  std::string location = interest.getName().getPrefix(4).toUri();
  ndn::Name session_key_ac_gkd(interest.getName()[HINT].toUri());
  ndn::name::Component identifier = interest.getName().at(TPC_POS);
  std::string nameGroup = interest.getName().at(GROUP_POS).toUri();
  std::string userName = whoIsSignature(interest);

  std::shared_ptr<Group> g = listGroups[nameGroup];

  ndn::Name session_key_name("session-key-consumer-gkd");
  ndn::name::Component encrypted_session_key =
  interest.getName().at(SESSION_KEY_POS);
  getSessionKeyFromInterest(encrypted_session_key, session_key_name);

  ndn::ConstBufferPtr c =
  m_secTpmFile.decryptInTpm(reinterpret_cast<const uint8_t*>(identifier.value()),
          identifier.value_size(), session_key_ac_gkd, true);

  std::string caveat_keyPredicate = std::string(c->buf(), c->buf() + c->size());

  Name dataName(interest.getName());
```

```
  bool authenticated = checkPredicate(caveat_keyPredicate, g, userName);
  if (authenticated) {
    std::string serialize_disMacaroon =
    g->getDischargueMacaroon(location,
          (uint8_t*)c->buf(),
          (uint8_t*)identifier.value(),
          identifier.value_size());

    ndn::ConstBufferPtr encrypted_serialized_disMacaroon =
    m_secTpmFile.encryptInTpm((uint8_t*) serialize_disMacaroon.c_str(),
          serialize_disMacaroon.size(),
          session_key_name,
          true);
    dataName
      .append("authenticated")
      .appendVersion();
    sendData(dataName, encrypted_serialized_disMacaroon);
  }else{
    dataName
      .append("notAuthenticated")
      .appendVersion();
    sendData(dataName, NULL);
  }
}
```

4.7. Pruebas

Las pruebas al sistema de acceso a recursos se realizan a través del Consumer, encargado de probar el correcto funcionamiento de las entidades que conforman el sistema.

4.7.1. Consumer

La entidad Consumer solicitará la información que necesita a Producer. Los datos solicitados irán cifrados con la clave de datos y la clave de datos estará cifrada con la clave de grupo. Consumer solicitará a AC el Macaroon con el TPC. Con el Macaroon se autenticará ante GKD para obtener la clave de grupo necesario para poder descifrar la información solicitada anteriormente.

Consumer se iniciará extrayendo los nombres de sus claves KSK y DSK del fichero keys.txt. Pasando a continuación a realizar una solicitud de los datos a Producer. Cuando recibe el paquete Data como respuesta a la solicitud Interest realizada, será tramitada por la siguiente función:

```
void
receivedData(const Interest& interest, const Data& data) {
  if(data.getName()[-1].toUri() == "notFound"){
    std::cout << "DATA FILE NOT FOUND" << std::endl;
  }else{
    std::cout << "ENCRYPTING DATA RECEIVED" << std::endl;
    enc_data = std::string(data.getContent().value(),
              data.getContent().value() + data.getContent().value_size());
    if(!decryptingData()){
      requestKeyData();
```

```
    }
  }
}
```

Si la firma del paquete Data es validada con éxito, significará que el paquete procede de una entidad en la que se confía y que el paquete no ha sido alterado por una entidad maliciosa. Una vez validado el paquete Data se comprobará el resultado de la petición realizada a través del nombre del recurso. Si al comprobar el último campo del nombre es encriptedData, extraerá el contenido del Data, almacenándolo en la variable enc_data. Tratará de descifrar los datos, si no es posible enviará una solicitud para pedir la clave de datos a Producer.

Cuando Consumer reciba el paquete Data de la petición de la clave de datos y sea validada su firma con éxito, se comprobará el nombre para ver el resultado de la solicitud realizada. Si el último campo del nombre contiene notEncrypted, ejecutará la función keyDataNotEncrypted, debido a que Producer está indicando que no tiene la clave de grupo para poder cifrar la clave de datos y poder enviarla. Esta función se encargará de solicitar la clave de grupo. En caso de que el último campo del nombre sea orden keyDataEncrypted, se extraerá el contenido del paquete Data, que contendrá la clave de datos cifrada. La clave de datos cifrada se intentará descifrar. Si Consumer posee la clave de grupo podrá desencriptar la clave de datos y seguidamente descifrará los datos. Si Consumer no posee la clave de grupo no podrá descifrar la clave de datos cifrada y almacenarla, teniendo que solicitar el Macaroon del grupo, necesario para obtener la clave de grupo correspondiente y así poder descifrar la clave de datos.

```
void
receivedKeyData(const Interest& interest, const Data& data) {
  if(data.getName()[-1].toUri() == "notEncrypted"){
    keyDataNotEncrypted();
  }else{
    enc_key_data = std::string(data.getContent().value(),
                data.getContent().value() + data.getContent().value_size());
    if(setKeyData()){
      decryptingData();
    }else{
      requestMacaroon();
    }
  }
}
```

La función keyDataNotEncrypted comprobará si tiene la clave de grupo, en caso de que no la tenga, pedirá el Macaroon a la entidad AC.

En cualquiera de los casos en los que Consumer no posea la clave de grupo, Consumer solicitará el Macaroon necesario para obtener la clave de grupo a la

entidad AC, a través de un Interest. Cuando Consumer reciba el paquete Data de la solicitud realizada y valide la firma, extraerá el Macaroon del contenido del paquete. Comprobará si el Macaroon contiene cabeceras con información de las claves DSK que debe solicitar. Si no contiene cabeceras con información de las claves DSK necesarias, Consumer considerará que el Macaroon no es válido y lo descartará. En caso de que tenga las cabeceras de las claves DSK que necesita, guardará el Macaroon, extraerá el TPC y la localización de GKD necesario para la verificación de la condición del TPC y poder obtener el DM.

```
void
receivedMacaroon(const Interest& interest, const Data& data) {
  Macaroons::e_Macaroon e_Macaroon;
  e_Macaroon.ParseFromArray(data.getContent().value(),
                            data.getContent().value_size());
  if (e_Macaroon.endorsements_size() > 0) {
    std::set<std::string> valid_names;
    std::string serializedMacaroon = e_Macaroon.Macaroon();
    Macaroon = make_shared<Macaroons::NDNMacaroon>(serializedMacaroon);
    valid_names.insert(Macaroon->getLocation());

    for (unsigned i = 1; i <= Macaroon->getNumThirdPartyCaveats(); i++){
      std::string group_keys_distributor;
      ndn::ConstBufferPtr tp_id_sp;
      Macaroon->getThirdPartyCaveat(i, group_keys_distributor, &tp_id_sp);
      valid_names.insert(group_keys_distributor);
    }
    fetchCertificate(e_Macaroon, valid_names, 0);
   }
}
```

Consumer con la función fetchCertificate pedirá los certificados de las claves DSK, para poder cifrar las claves de sesión simétricas que establecerá con AC y GKD para intercambiar datos de forma segura.

Cuando Consumer haya obtenido los certificados de las claves DSK, pasará a procesar la cabecera TPC contenida en el Macaroon. Creará una clave de sesión cifrada con la clave DSK de GKD. Consumer enviará una solicitud Interest a GKD para verificar la condición impuesta en el TPC y firmará el paquete. La firma es necesaria por el contenido importante del nombre y para poder autenticarse en la entidad GKD.

```
void
processThirdPartyCaveats (){
  for (unsigned i = 1; i <= Macaroon->getNumThirdPartyCaveats(); i++){
    std::string group_keys_distributor;
    ndn::ConstBufferPtr tp_id_sp;
    Macaroon->getThirdPartyCaveat(i, group_keys_distributor, &tp_id_sp);

    ndn::Name session_key_name(std::string("/session-key-consumer") +
                          std::string("-") + std::to_string(i));
    ndn::ConstBufferPtr enc_session_key =
```

```
    createSessionKey(group_keys_distributor, session_key_name);

    Name interestName(group_keys_distributor);
    interestName.append(ndn::name::Component(enc_session_key))
      .append(ndn::name::Component(*tp_id_sp))
      .append("Doctors")
      .append("V1");

    Interest interest = createInterest(interestName.toUri(), true);

    m_face.expressInterest(interest,
            bind(&Consumer::onThirdPartyData, this, _2, session_key_name),
            bind(&Consumer::onTimeout, this, _1, 1));
  }
}
```

Cuando Consumer reciba el paquete Data de la solicitud del DM y valide la firma, se comprobará el último componente del nombre para ver el resultado de la solicitud. Si el componente es notAuthorized, Consumer no forma parte del grupo del que ha recibido el Macaroon, con lo que no tiene acceso a la clave de grupo, y no podrá descifrar la clave de datos cifrada, ni los datos cifrados. Si el componente es authorized, extraerá el DM cifrado en el contenido del paquete Data, descifrará el DM con la clave de sesión establecida con GKD. Incluirá el DM en el Macaroon y ejecutará la función validateMacaroonAndDischarge.

```
void
onThirdPartyData(const Data& data, ndn::Name& session_key_name){
  if (data.getName()[-2].toUri() == "authenticated") {
    ndn::ConstBufferPtr decrypted_content =
    m_secTpmFile.decryptInTpm(data.getContent().value(),
            data.getContent().value_size(),
            session_key_name, /*symmetric*/ true);

    std::string dm = std::string(decrypted_content->buf(),
                      decrypted_content->buf() + decrypted_content->size());

    Macaroon->addDischargeAndPrepare(dm);
    validateMacaroonAndDischarge();
  }else{
    std::cout << "NOT authenticated!" << std::endl;
  }
}
```

La función validateMacaroonAndDischarge, extraerá la clave de grupo y la almacenará. Enviará un Interest a la entidad AC, con la orden updateGroupKey que contendrá la siguiente información en el nombre: clave de sesión cifrada con la clave DSK, el Macaroon, el DM cifrado con la clave de sesión, el nombre del producer en el que se establecerá la clave de grupo, grupo y versión del grupo. Consumer esperará recibir el paquete Data con el resultado de la validación y verificación del Macaroon.

Cuando Consumer reciba el paquete Data y valide la firma utilizará la función responseValidatedMacaroonAndDischarge para comprobar el nombre. Si el último componente del nombre es authorized, comprobará si tiene la clave de datos cifrada. Si Consumer no dispone de ella, la pedirá a Producer, y una vez que reciba el paquete Data con la clave de datos cifrada tendrá ya lo necesario para descifrar tanto la clave de datos como los propios datos. Si ya posee la clave de datos cifrada, pasará a descifrarla, y por último a descifrar los datos.

```
void
responseValidatedMacaroonAndDischargue(const Interest& interest,
                                       const Data& data) {
  if (data.getName()[-1].toUri() == "authorized") {
    if(enc_key_data == ""){
      requestKeyData();
    }else{
          setKeyData();
      decryptingData();
    }
  }else{
    std::cout << "Macaroon not validated" << std::endl;
  }
}
```

4.7.2. Ejecución de escenarios de prueba

Se realizan varios escenarios de prueba para comprobar el funcionamiento del sistema de acceso a recursos:

- Escenario 1: Un único Consumer.

  Solo se ejecuta un único Consumer en el sistema de acceso a recursos. Al ejecutarse por primera vez, el consumidor se encarga de solicitar los datos y la clave de datos. Al no tener Producer la clave de grupo no podrá enviarla. Consumer se encarga de establecer la clave de grupo en Producer y volver a solicitar la clave de datos para poder almacenarla y descifrar los datos solicitados con anterioridad. Cuando Consumer se ejecuta por segunda vez se descifran los datos más rápido, debido a que ya está establecida la clave de grupo en Producer y no hace falta solicitar la clave de datos por segunda vez.

- Escenario 2: N consumidores.

  Se ejecutan varios consumidores solicitando el mismo recurso. El primer consumidor establecerá la clave de grupo necesaria para que Producer cifre la clave de datos. El acceso al recurso por parte del resto de consumidores se hará de manera más rápida, una vez establecida la clave de grupo. Todos los Consumidores pueden descifrar los datos solicitados.

Capítulo 5
Conclusiones

5.1. Conclusiones

En este capítulo se resumirá el trabajo realizado para conseguir cumplir los subobjetivos marcados en el capítulo 2. La finalidad de los subobjetivos era dividir el trabajo para la facilitar el desarrollo del objetivo principal: crear un sistema de acceso a recursos a través de Macaroons en una red NDN.

Subobjetivo 1: Estudio y compresión de la arquitectura NDN

Estudiar y asimilar los conceptos básicos, así como los conceptos más complejos de una arquitectura de red NDN ha sido una parte fundamental de mi trabajo de fin de grado. Toda la información necesaria para el estudio de la arquitectura de red NDN la he obtenido de la página http://named.data.net . Las tutorías han sido de gran ayuda para la asimilación y compresión de ciertos conceptos de NDN, así como del funcionamiento de una red NDN.

Subobjetivo 2: Estudio y compresión de los Macaroons

El estudio de los Macaroons se realizó a través de varios documentos proporcionados por mi tutora, uno de los principales fue el artículo de investigación sobre Macaroons[12] donde se explica de manera detallada los Macaroons. Para asimilar y comprender completamente la tecnología Macaroons me he apoyado en ejemplos proporcionados por mi tutora así como de las tutorías necesarias para solucionar las dudas que podía tener en la materia.

Subobjetivo 3: Estudio y compresión de la librerías necesarias para el desarrollo de aplicaciones NDN

El estudio que he realizado de las librerías de NDN ha sido a través de los ejemplos básicos de intercambio de paquetes Interest y Data aportados por la

página http://named-data.net. Para comprender la firma y validación de paquetes a través de certificados he estudiado los ejemplos aportados por mi tutora y que se pueden encontrar en https://github.com/evaCastro/ndn-signed-data . En todos lo ejemplos se pueden apreciar las funciones necesarias para el envío, recepción y validación de paquetes. En algunos casos la información proporcionada por los ejemplos no ha sido suficiente, por lo que ha sido necesaria la utilización de información que proporciona la página de NDN donde se explican las clases y sus métodos, permitiéndome conocer otros métodos diferentes a los utilizados en los ejemplos.

Subobjetivo 4: Estudio y compresión de la librerías necesarias para el desarrollo de aplicaciones con Macaroons

Para el desarrollo de la aplicación ha sido necesario comprender las librerías libNDNMacaroon y libMacaroon. LibMacaroon me ha ayudado a comprender la estructura del Macaroon y cada una de sus partes, así como la creación, modificación y verificación de los Macaroons. LibNDNMacaroon utiliza la librería libMacaroon y la adapta para el uso en la arquitectura NDN. La librería libNDNMacaroon ha sido necesaria comprenderla tanto para su modificación como para poder enviar y recibir los Macaroons.

Subobjetivo 5: Diseño de un modelo de autorización descentralizado con Macaroons en un red NDN

El diseño está basado en un ejemplo proporcionado por mi tutora. El sistema se diseñó pensando en las funciones que tendría que llevar a cabo cada entidad, además de la comunicación que tenía que existir entre todas las entidades participantes en el sistema de autorización descentralizado. El sistema está compuesto por la entidad Producer que es la encargada de suministrar los datos cifrados y las claves de datos también cifradas con claves de grupo. Otra de las entidades es AC que es la encargada de generar los Macaroons necesarios para que los consumidores puedan solicitar la clave de grupo, además de que AC es la única entidad capaz de establecer la clave de grupo con Producer. La última entidad es GKD que es la entidad encargada de autentificar los consumidores a través del TPC que envían los consumidores para solicitar el DM. Con las tareas específicas de cada entidad un consumidor es capaz de obtener los datos cifrados, la clave de datos cifrada a Producer, solicitar el Macaroon a AC y poder autentificarse en GKD para poder obtener la clave de grupo que le ayude a descifrar la clave de datos cifrada y a continuación los datos cifrados.

Subobjetivo 6: Desarrollo de una aplicación que permita aplicar el modelo diseñado

El consumidor es la aplicación que permite probar el sistema de acceso a los recursos que publica el productor.

El consumidor ha permitido solicitar el Macaroon que contiene las cabeceras FPC y TPC. El Macaroon debe adjuntar el DM para que tenga validez y permita obtener la clave de grupo. El DM se obtiene de la tercera parte que es GKD, la que permite verificar la identidad del consumidor y ver si pertenece al grupo que indica la condición impuesta en el TPC.

5.2. Problemas encontrados

En esta sección se explicarán los problemas surgidos durante la realización del trabajo de fin de grado. Algunos de los problemas que aparecieron fueron:

- Planificación temporal: en la figura 2.1 se puede observar la estimación de tiempo esperada para la realización de cada subobjetivo. Esta estimación no se pudo cumplir obligándome a dedicar más tiempo a cada subobjetivo debido a la complejidad de alguno de ellos.

- Las versiones de las librerías utilizadas en este trabajo de fin de grado han ido evolucionando y cambiando sus APIS. Ha sido necesario restringir el código de este trabajo a versiones concretas de ndn-cxx y libMacaroons.

- La falta de documentación sobre la librería libMacaroon. Ha sido necesario el estudio del código de la librería de manera más minuciosa para poder comprender el uso de ciertos métodos.

- El objetivo inicial de las claves DSK era cifrar cierta información dentro del sistema. El cifrado de estos datos con la clave DSK no ha sido posible por las limitaciones impuestas por la propia librería sobre el tamaño de los datos a cifrar con claves RSA. Para solucionar el problema se establecen claves de sesión simétricas que permiten cifrar los datos enviados. La clave DSK se utiliza para cifrar la clave de sesión entre ambas entidades.

5.3. Trabajos futuros del modelo de autorización descentralizado con Macaroons en una red NDN

En esta sección se propondrán futuras mejoras para el sistema de acceso a recursos diseñado. Siendo algunas posibles mejoras:

- El modelo que he desarrollado está limitado a entidades que son miembros de los grupos ya creados, sin que estas entidades permitan el acceso a recursos específicos a otras entidades que no pertenezcan a su grupo. Una de las posibles mejoras que se pueden aplicar al modelo desarrollado es la atenuación del Macaroon por parte de los consumidores para poder dar acceso a recursos específicos a otras entidades ajenas. El nuevo consumidor que recibe el Macaroon atenuado tendría que validarse con GKD para obtener una clave específica para descifrar el recurso.

- Otra posibles mejoras son la salida de miembros de los grupos. Esta mejora obliga a GKD a notificarle a AC que existen cambios en el grupo. GKD modificaría el DM para cambiar la clave de grupo y AC informaría a los productores de que existe un cambio de clave de grupo. Los productores tendrían que crear nuevas claves de datos. Las nuevas claves de datos serían cifradas con las nuevas claves de grupo. Para fortalecer la seguridad se puede establecer un tiempo de vida corto al Macaroon, obligando a cambiar la clave de grupo cada poco tiempo.

- Producer está diseñado para publicar únicamente información de un grupo. Una mejora sería hacerlo para publicar los datos de más de un grupo en Producer.

Anexo

El ejemplo desarrollado para probar el sistema de delegación de accesos a recursos a través de Macaroons sobre una red NDN se encuentra en el siguiente repositorio de GitHub:

https://github.com/djuanes87/PFG-NDNMacaroons

Bibliografía

[1] Named Data Networking:
https://named-data.net/wp-content/uploads/2014/04/tr-ndn-0019-ndn.pdf

[2] RFC1180 - Un Tutorial de TCP/IP:
https://www.rfc-es.org/rfc/rfc1180-es.txt

[3] RFC791 - Internet Protocol:
https://www.rfc-es.org/rfc/rfc0791-es.txt

[4] RFC2460 - Internet Protocol, Version 6 (IPv6):
https://www.rfc-es.org/rfc/rfc0791-es.txt

[5] Networking Named Content
http://conferences.sigcomm.org/co-next/2009/papers/Jacobson.pdf

[6] Consumer-Producer API for Named Data Networking:
https://named-data.net/wp-content/uploads/2014/02/TR17-consumer-producer-API.pdf

[7] NFD Developers Guide:
https://named-data.net/wp-content/uploads/2016/10/ndn-0021-7-nfd-developer-guide.pdf

[8] NDN Technical Memo: Naming Conventions :
https://named-data.net/wp-content/uploads/2014/08/ndn-tr-22-ndn-memo
-naming-conventions.pdf

[9] Segurity library NDN:
http://named-data.net/doc/ndn-cxx/current/tutorials/security-library.html

[10] Public Key Management in Named Data Networking:
https://named-data.net/wp-content/uploads/2015/04/ndn-0029-1-public-
key-management-ndn.pdf

[11] API documentation:
http://named-data.net/doc/ndn-cxx/current/doxygen/annotated.html

[12] Macaroons:
http://research.google.com/pubs/pub41892.html

[13] Protocol Buffers:
https://developers.google.com/protocol-buffers/docs/overview