TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Sistema de autenticación de doble factor basado en Criptografía de Curva Elíptica

Autor
Sergio García Prados

Director
Antonio Francisco Díaz García

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 9 de julio de 2021

Sistema de autenticación de doble factor basado en Criptografía de Curva Elíptica

Sergio García Prados

Palabras clave: Criptografía, Sistemas operativos, C, Pluggable Authentication Module, Autenticación, Message Queuing Telemetry Transport, Seguridad, Criptografía de Curva Elíptica, Algoritmo de Firma Digital de Curva Elíptica

Resumen

Cada vez está más en boca el término "seguridad". No es de extrañar que desde que comenzó la era digital, los retos tecnológicos son cada vez más complejos. No obstante, dado el grado de conocimiento y herramienta de las que disponemos, estos son bastante factibles y llamativos para cualquiera. Es por eso que el uso de elementos tecnológicos en nuestro día a día se concibe como algo innato en nuestras vidas.

La tecnología nos trae numerosos beneficios. Nada más que con echar la vista atrás y ver como la sociedad ha llegado a conseguir retos tan importantes gracias a la ayuda de esta es suficiente. Sin embargo, la tecnología tiene su lado adverso, la seguridad. Por muy llamativa y útil que pueda ser una tecnología, si no se lleva a cabo unos cumplimientos de seguridad bien definidos y estrictos durante su desarrollo, el usuario final puede no llegar a usarlo y por tanto quedar en vano el trabajo realizado o peor aun, que sea usado muchos usuarios y tengas graves vulnerabilidades que afecten a la integridad de sus datos.

Con la llegada del *Big Data*, cada vez más tenemos acceso a cualquier información en cualquier momento. Las compañías hacen un arduo esfuerzo en proteger nuestros datos y que estos no caigan en las manos equivocadas. Pero como bien es de saber, ningún sistema, por muy complejo y sofisticado que pueda llegar a ser, es 100 % seguro. Cada día salen nuevas vulnerabilidades y complejos ataques para corromper estos sistemas y es por ello que la inversión de las compañías en ciberseguridad incrementa cada año. Para que esta inversión sea eficiente, es necesario apostar por la investigación en este campo y así garantizar un futuro más "seguro".

El objetivo de este trabajo es el estudio de la seguridad en entornos definidos (HPC y Cloud) donde el volumen de datos que se almacena es elevado y por consiguiente puede ser información más sensible, garantizando la confidencialidad, integridad y disponibilidad del sistema.

Se propone una solución [1] de un modelo de autenticación multifactor, MFA, basada en un sistema electrónico (MCU basado en un ESP-32) garantizando un alto rendimiento y eficiencia debido al consumo de estos

componentes electrónicos, una arquitectura distribuida ofreciendo una mayor escalabilidad y con un alta granularidad en cuanto a seguridad se refiere.

Este trabajo se centra principalmente en la incorporación de un modulo de seguridad a este sistema de autenticación propuesto de forma que pueda ser usado por cualquier otro dispositivo en entornos variados garantizando una solución portable y eficiente.

Double authentication factor system based on Elliptic Curve Criptography

Sergio García Prados

Keywords: Criptography, Operative System, C, Pluggable Authentication Module, Authentication, Message Queuing Telemetry Transport, Security, Elliptic Curve Criptography, Elliptic Curve Digital Signature Algorithm

Abstract

The term "security" is becoming more and more popular. It is not surprising that since the digital era began, technological challenges have become increasingly complex. However, given the degree of knowledge and tools at our disposal, they are quite feasible and appealing to anyone. That is why the use of technological elements in our daily lives is conceived as something innate in our lives.

Technology brings us numerous benefits. Just looking back and seeing how society has achieved such important challenges thanks to its help is enough. However, technology has its downside: security. No matter how attractive and useful a technology may be, if it does not comply with well-defined and strict security standards during its development, the end user may not use it and therefore the work done may be in vain, or worse still, it may be used by many users and have serious vulnerabilities that affect the integrity of their data.

With the advent of *Big Data*, we increasingly have access to any information at any time. Companies make an arduous effort to protect our data from falling into the wrong hands. But as we all know, no system, no matter how complex and sophisticated it may be, is 100 % secure. Every day new vulnerabilities and complex attacks to corrupt these systems emerge and that is why companies' investment in cybersecurity increases every year. In order for this investment to be efficient, it is necessary to invest in research in this field and thus guarantee a more "secure" future.

The objective of this work is the study of security in defined environments (HPC and Cloud) where the volume of data stored is high and therefore may be more sensitive information, ensuring the confidentiality, integrity and availability of the system.

A solution [1] of a multifactor authentication MFA model is provided, based on an electronic system (MCU based on ESP-32) guaranteeing high performance and efficiency due to the consumption of these electronic components, a distributed architecture offering greater scalability and with a high granularity in terms of security.

This work is mainly focused on the incorporation of a security module to this proposed authentication system so that it can be used by any other device in varied environments guaranteeing a portable and efficient solution.

Yo, **Sergio García Prados**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77148519X, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Sergio García Prados

Granada a 9 de julio de 2021

D. **Antonio Francisco Díaz García**, Profesor del Departamento de Arquitectura y Tecnoloía de Computadores de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Sistema de autenticación de doble factor basado en Criptografía de Curva Elíptica***, ha sido realizado bajo su supervisión por **Sergio García Prados**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 9 de julio de 2021.

El director:

Antonio Francisco Díaz García

Agradecimientos

A mi director de Trabajo Fin de Grado, Antonio Francisco Díaz García por su coordinación y conocimiento acerca de la materia que me ha ayudado a desarrollar este trabajo a pesar de los inconvenientes.

Tambien me gustaría agradecer este trabajo al apoyo de mi madre y hermanos que han confiado en mi en todo momento y me han dado soporte en los momentos mas dificiles de mi recorrido académico. Pero sobre todo a mi padre, que ha sido mi apoyo incondicional en estos años de carrera y del que seguro que estaría orgulloso de ver lo lejos que he llegado.

¡Gracias!

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

El concepto de "Seguridad de la Información" se acuñó por primera vez en un artículo del Instituo Nacional de Estándares y Teconología (NIST) en 1997 [2] aunque la seguridad en el campo de la tecnología siempre ha estado en boca de todos y ha sido un apsecto a tener en cuenta.

La ciberseguridad se empezó a tener en cuenta con el "*boom*" de la era digital. No obstante, habría que remontarse unos siglos atras para saber cuando se empezó a aplicar métodos seguros a la información. Concretamente cuando Julio César vivía, se empezó a usar algoritmos de cifrado simples que se aplicaban sobre mensajes que tenían que ser enviados y así si alguien lo interceptada, no podía saber su contenido. A esta práctica se conoce como criptografía. La criptografía ha estado latente en muchos hitos de nuestra historia, como por ejemplo con la aparición Alang Turing y su ingenio para descifrar las comunicaciones entre integrantes del ejército alemán en código morse.

La criptografía, según la RAE es el "arte de escribir con clave secreta o de un modo enigmático". Esta herramienta ha sido la base de la seguriadad de la información ya que sustenta las tres propiedades de todo sistema seguro (CIA): confidencialidad (*confidenciality*), integridad (*integrity*) y disponibilidad (*availability*).

Tal y como se puede apreciar en [3], el número de ciberataques que suceden a diaría es muy elevado. Es por ello que se cualquier empresa que de soporte a clientes almacenando información sensible use herramientas de seguridad que estén al día de los distintas vulnerabilidades que se descubren [4].

1.1. Motivación

La idea de este tarbajo nace de a raiz de la tesis doctoral [5] publicada en 2020 por la Universidad de Granada titulada "*Mecanismos de seguridad para Big Data basados en circuitos criptográficos*" y elaborada por *Ilia Blockin*.

Esta tesis sugiere soluciones basada en sistemas electrónicos que ofrecen una solución eficiente y flexible para aumentar la seguridad en el acceso a servicios y sistemas que pueden procesar gran cantidad de información.

El planteamiento se detalla en unos de las mejoras propuestas de dicha tesis: "*continuar mejorando los sistemas propuestos agregando compatibilidad con otros métodos de autenticación como el Módulo de autenticación conectable de Linux (PAM) u otros protocolos de autenticación*"

Personalmente, me he decantado por la elección de este tema ya que tengo especial interés en el campo de la ciberseguridad y por el reto de conocer en profundidad las bases de la criptografía.

1.2. Objetivos

Para elaborar la lista de objetivos, es necesario conocer los requisitos del sistema donde se pretende implantar. Este entorno viene descrito de forma detallada en [1].

1.2.1. Diseño del sistema propuesto

Se propone un sistema con un conjunto de elementos físicos y virtualizados:

- *Usurio*: quien accede al sistema
- *Servicios*: ofrecido por programas a través de conexión TCP
- *Servidores*: ordenadores donde los servicios son instalados
- *Clientes*: ordenadores desde donde los servicios son accedidos
- *eToken*: dispositivos que autentica la petición del usuario

Se define una lista de control (ACL) con reglas para garantizar una autorización deseada. Cada elemento viene identificado por un Identificador Único Universal (UUID) de 128 bits y un par de claves pública y privada que permite al sistema a autenticar las peticiones. El sistema es híbrido, es decir que usan la clave pública combinado con un modelo centralizado para autorizar el acceso. Hay dos elementos más:

- *Servidor de configuración*: permite modificar la configuración de cada elemento remotamente
- *Servidor de autenticación*: autoriza la petición de acceso a servicios basado en las ACL
- *Gateway*: redirige la conexión de red entre clinte y servidores
- *Broker MQTT*: servidor de intercambio de mensajes

1.2.2. Listado de objetivos

El presente trabajo conlleva el cumplimiento de los siguientes objetivos:

1. Crear un módulo PAM para el sistema de autenticación propuesto en [1]
2. Implementar el módulo PAM para el servicio SSH
3. Usar el protocolo MQTT
4. Seguir esquema de autenticación *challenge-response* [6]
5. Cifrar el *challenge* mediante un algoritmo de cifrado unidireccional robusto como por ejemplo SHA512 (SHA)
6. Usar Criptografía de Curva Elíptica o ECC tanto para la firma del *challenge* como para la verificación del mismo usando el Algoritmo de Firma Digital de Curva Elíptica (ECDSA)
7. Encriptar las comunicaciones entre los elementos del sistema propuesto usando TLS versión 1.2

1.3. Estructura del trabajo

El presente trabajo se divide en las siguientes capítulos: en el Capítulo 2 se detalla la "situación actual de la tecnología". Se habla de otros proyectos que existen actualmente y que realizan funcionalidades iguales o parecidas a las que se propone en este proyecto. Se valora los puntos fuertes y ámbitos en los que se puede aplicar. En el Capítulo 3, se habla de la seguridad en el apartado de la autenticación en sistemas, la seguridad aplicada a entornos IoT, el protocolo MQTT usado, la criptografía de curva elíptica y el algoritmo ECDSA y por último PAM y su arquitectura. Al final se hace alusión de las herramientas usadas para el desarrollo de este trabajo. En el Capítulo 4 se hace un análisis exhaustivo del funcionamiento del programa indicando sus distintas fases. En el Capítulo 5 lista algunas caracterísiticas de seguridad y cómo las lleva a cabo. En el Capítulo 6 se hace un resumen de lo que ha costado económicamente el proyecto y su desarrollo temporal con un diagrama de Gantt. Por último, en el Capítulo 7 se hace un resumen de objetivo de este proyecto, los resultado obtenidos, problemas afrontados y futuras mejoras.

Capítulo 2

Estado del arte

Vamos a enfocar los trabajos relacionados con respecto a la seguridad que ofrece la Criptografía de Curva Elíptica en dispositivos electrónicos para proporcionar un método de autenticación de doble factor. Los trabajos citados a continuación son comparados con el propuesto en [1]. *Braeken* propone en [7] un protocolo de autenticación para dispositivos IoT basado en una función física no clonable PUF. Ambos sistemas se caracterizan porque usan un esquema de acuerdo de claves en el que cualquier nodo está registrado bajo la supervisión de un tercer elemento de confianza TTP y el uso de certificados. Además, en ambos sistemas se usa el mecanismo de autenticación basado en reto-respuesta (*challenge-response*). En este, una entidad se autentica enviando un valor dependiente de un valor secreto y un valor desafío cambiante [8]. Si la respuesta es correcta, se da por légitimo al cliente y se autentica. El sistema prouesto en [1] a diferencia de [7], usa el protocolo MQTT para la comunicación entre los dispositivos de tal forma que lo hace más escalable.

Gao [9] propone un sistema similar que usa un token de autenticación basado en el formato JSON, JWT, junto al servicio SSH y desarrolla, al igual que lo que se propone en el presente trabajo, un módulo PAM de tipo desafio-respuesta. Sin embargo, a diferencia de [9], este trabajo no usa un formato concreto para responder al desafio. De manera similar a [7], tmapoco implementa el protocolo MQTT para la comunicación entre los nodos que intervienen.

Existen otros sistemas como por ejemplo el propuesto por *Fayad* en [10] que ratifica la robustez que tiene ECC sobre otros mecanismos de autenticacion como las contraseñas de un solo uso (OTP) basadas en mensajes *Hash* (HMAC) o basadas en tiempo (TOTP) para entornos con dispositivos IoT.

Actualmente, existen múltiples métodos de autenticación bien definidos que cumplen con los parametros requeridos: algo que sabes (contraseña), algo que tienes (token) y algo que eres (biometría). Para que sea multifactor, al menos dos parametros tienen que ser requeridos. Google creó un módulo

PAM [11] para garantizar un factor de serguridad en la autenticación usando su producto *Google Authenticator*. Por otra parte, existen productos como llaves físicas de autenticación que verifican tu identidad usando protocolos de seguridad robustos. Es el caso por ejemplo de Yubikey [12]. A diferencia del sistema propuesto, la verificación por llave de seguridad requiere del dispositivo físico. También hay productos software en el mercado que permiten el acceso seguro a sistemas sin la necesidad de tener que proveer de la metodología clásica par usario-contraseña. El inicio de sesión único o SSO [13] es un método de autenticación que permite a los usuarios autenticarse de forma segura en múltiples aplicaciones y servicios web usando un único conjunto de credenciales.

Capítulo 3

Análsis del problema

3.1. Seguridad en la autenticación

La autenticación del usuario es el punto de entrada a diferentes redes o instalaciones informáticas en las que se presta un conjunto de servicios a los usuarios o se puede realizar un conjunto de tareas.

Una vez autenticado, el usuario puede acceder, por ejemplo a la Intranet de una empresa, a consolas, bases de datos, edificios vehículos, etc. La usabilidad de los mecanismos de autenticación se investiga cada vez con más detenimiento y dado que los estos son concebidos, implementados, puestos en práctica y corrompidos por teceras personas, hay que tener en cuenta los factores humanos en su diseño.

Actualmente existen múltiples métodos para autenticar a un usuario contra un sistema, siendo el más común es el par de claves usuario y contraseña, pero no es el único. El uso de certificados está cada vez más extendido en Infraestructuras de Clave Pública (PKI).

Para garantizar que un sistema es necesario que se garantizen tres aspectos [14]:

- *Confidencialidad*: prevenir la divulgación no autorizada de la información
- *Integridad*: prevenir modificaciones no autorizadas de la información
- *Disponibilidad*: prevenir interrupciones no autorizadas de los recursos informáticos

Usar un método de autenticación no implica que el sistema sea completamente seguro. De echo, la autenticación se convierte en un proceso más robusto y fiable aplicando múltiples métodos, también llamado MFA. Este mecanismo propone tres tipos de factores que permiten a un usuario vincularlo con las credenciales establecidas [15]:

1. *Factor conocimiento*: algo que el usuario conoce (contraseña)
2. *Factor pertenencia*: algo que el usuario tiene (token)
3. *Factor biometrico*: algo que el usuario es (huella dactilar)

La autenticación basada en múltiples factores provee un nivel de seguridad elevado y facilita una protección continua de dispositivos y otros servicios críticos ante accesos no autorizados usando dos o más factores.

Además de la robustez de la seguridad durante el proceso de autenticación, la usabilidad se convierte a su vez en una cuestión estratégica en el establecimiento de métodos de autenticación de usuarios.

La usabilidad puede definirse como "la medida en que un producto puede ser utilizado por determinados usuarios para alcanzar determinados objetivos con eficacia, eficiencia y satisfacción en un contexto de uso específico". La usabilidad de la seguridad se ocupa del estudio de cómo debe tratarse la información de seguridad en la interfaz de usuario y de cómo deben ser los mecanismos de seguridad y los sistemas de autenticación deben ser fáciles de usar [16].

Este sistema [1] propone un mecanismo de autenticación usable, de múltiples factores y escalable gracias principalmente al uso de dispositivos electrónicos con circuitos integrados como es Arduino así como del protocolo MQTT.

3.1.1. Seguridad en IoT

El Internet de las cosas (IoT) es un término relativamente nuevo. Se puede definir como una red abierta y completa de objetos inteligentes que tienen la capacidad de auto-organizarse, compartir información, datos y recursos, reaccionar y actuar ante situaciones y cambios en el entorno [17].

El IoT ha facilitado la interconectividad entre dispositivos ayudando a conectar sensores, vehículos, hospitles, industrias y consumidores a través de Internet. Las arquitecturas en IoT son cada vez más complejas, descentralizadas y fluidas dado el incremento de dispositivos que se comunuican entre sí y es por ello que la seguridad juega un papel fundamental en él. Los dispositivos IoT deben ser seguros y no pueden ser manipulados por terceras personas no autorizadas.

3.1.2. MQTT

Es un protocolo de comunicación que funciona a nivel de la capa de aplicación. Es usado principalmente en entornos con dispositivos IoT por ser un protocolo con un consumo de bando de ancha y batería mínimo [18].

A diferencia del protoclo de comunicación HTTP que funciona mediante petición-respuesta, MQTT está basado es publicación-respuesta. Este modelo de protocolo requiere de un broker mensajero. Existen múltiples tipos de

brokers como Mosquitto [19], HiveMqtt [20], Mosca [21], cloudMQTT [22], MQTT.Js [23], etc. Para el trabajo propuesto, se usa Mosquitto.

Funcionamiento del protocolo MQTT

El cliente MQTT se conecta el broker MQTT subscribiéndose a un tópico. Un tópico es un identificador al cual los nodos se subscriben y por el que publican mensajes. Si algun mensaje es publicado por un tópico, este llega a todos los nodos que estés suscritos al mismo. Se representa con una cadena de caracteres separada por el símbolo / indicando el elemento, dentro de una jerarquía, al que se subscribe. Un ejemplo de un tópico sería $cocina/sensor1/temperatura$ de tal forma que el cliente obtiene la información de temperatura que le llega el sensor1 en la cocina. MQTT se basa en el protoclo TCP/IP para la transmisión de datos. El cliente siempre interactua con el broker. No obstante, puede haber múltiples brokers que intercambien datos segun los tópicos de subscripción.

Seguridad en MQTT

Para garantizar un nivel de seguridad elevado en cuanto a la transmisión de datos se refiere, este protocolo usa por encima TLS para garantizarlo. Este protolo crea un canal seguro entre el cliente y el broker, parecido a como funciona en los navegadores web. Para ello es necesario un certificado del servidor de confianza.

Ventajas de MQTT sobre HTTP

En HTTPS, no se mantiene una conexión continua. El cliente tiene que mandar petición de sondeo (*poll request*) donde este solicita información al servidor acerca de la nueva petición de conexión TCP así como la negociación del tipo de encriptación SSL/TLS.

A diferencia de HTTPS, MQTT establece una única conexión inicial. MQTT mantiene un flujo de mensajes fijo (*keep alive*) entre aplicaciones sobre la conexión TCP de tal forma que las aplicaciones pueden detectar cuando una conexión está rota.

3.1.3. Criptografía de Curva Elíptica

La Criptografía de Curva Elíptica (ECC) pertenece a las tres familias de algoritmos de clave pública de gran relevancia (factorización de enteros, logaritmos discretos y curva elíptica). Este último nació entorno a mitad de los años 80. ECC provee del mismo nivel de seguridad que RSA o sistemas logarítmicos discretos con operaciones considerablemente pequeñas. ECC está basado en el problema de logaritmo discreto (DLP) [24]. ECC tiene beneficios en cuanto a su capacidad computacional ya que requiere de menos

operaciones computacionales y tiene un consumo de ancho de banda muy inferior debido a la pequeña longitud de las claves y firmas con respecto otros como RSA. Esto lo hace ídoneo en arquitecturas que intervienen dispositivos electrónicos de bajo consumo (Arduinos, Raspberry Pi, etc). No obstante, las operaciones RSA que implican una clave pública más pequeña son más rápidas que ECC.

El objetivo de esta sección es explicar las bases de este algoritmo sin entrar en detalles matemáticos. Para mayor documentación se sugiere leer el libro [25] del cual se ha sacado la información.

Esta sección se distribuye de la siguiente manera: definición del concepto de curva elíptica, problema de logaritmo discreto aplicado a las curvas elípticas, algoritmo ECDSA y comparativa con RSA.

Definición de curva elíptica

De las ecuaciones polinómicas de una circunferencia y una elipse se pueden sacar diferentes tipos de curvas. Las curva elíptica es un tipo especial de ecuación polinómica. En criptografía se trabaja sobre un conjunto de números finitos, más popularmente campos primo, donde todas las operaciones aritméticas se desarrollan sobre módulo p, siendo este un número primo.

La definición de curva elíptica es la siguiente:

Definition 3.1.1 (Curva Elíptica)**.** La curva elíptica sobre $\mathbb{Z}_p$, $p > 3$, es el conjunto de pares de punto $(x,y) \in \mathbb{Z}_p$ que cumplen

$$y^2 = x^3 + ax + b \; mod \; p \tag{3.1}$$

junto a un punto infinito imaginario O, donde

$$a, b \in \mathbb{Z}_p \tag{3.2}$$

y la condición

$$4a^3 + 27b^2 \neq 0 \; mod \; p \tag{3.3}$$

Ejemplo de una curva elíptica:

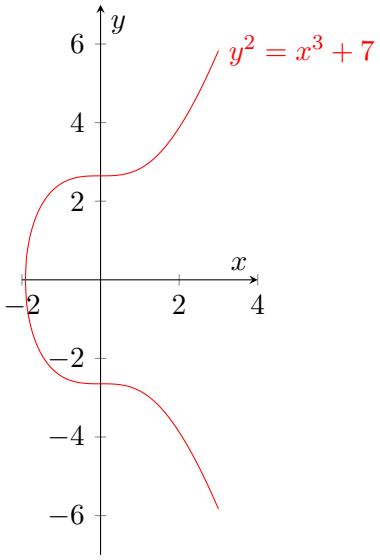

Figura 3.1: Representación gráfica de una curva elíptica

Aspecto fundamentales a tener en cuenta de la curva:

- Es una curva no-singular, es decir que la curva no se intersecta a si mismo siempre y cuando el discriminante de la curva $-16(4a^3+27b^2) \neq 0$
- La curva es simétrica con respecto al eje de abscisas ya que si despejamos la y de la ecuación obtenemos dos soluciones $\pm\sqrt{x^3 + ax + b}\ mod\ p$
- Hay solo una intersección en el eje de abscisas. Se debe a que si solucionamos la ecuación para $y = 0$, exite una solución real (intersección con el eje de abscisas) y dos soluciones complejas (no mostradas en la gráfica). Existen curvas elípticas con tres soluciones reales.

ECDSA

La curva elíptica tiene ventajas sobre RSA y otros esquemas DL. Los ECC con una clave con un tamaño que oscila entre los 160 y 256 bits proveen de una seguridad similar a otros algoritmos criptográficos de entre 1024 y 3072 bits. Esta propiedad resulta en un tiempo de procesamiento menor así como firmas más pequeñas. Por esa razón, ECDSA fue estandarizado en Estados Unidos por la ANSI en 1998. El estándar ECDSA es definido para curvas elípticas sobre campos de número primo $\mathbb{Z}_p$ y campos de Galois $GF(2^m)$. Este algorimo se compone de las siguientes fases:

1. Generación de claves. Estas deben ser de al menos 160 para un nivel de seguridad elevado

2. Generación de firma. Usa la clave privada para generar una pareja de valores enteros (r, s). Usa

3. Verificación de firma. Usa la clave pública junto al par (r, s) y un valor concreto.

Seguridad sobre ECDSA

Suponiendo que los parámetros de la curva elíptica son escogidos correctamente, el principal ataque analíico sobre ECDSA intentaría resolver el problema de logaritmo discreto de curva elíptica. Si un atacante fuera capaz de llevarlo acabo, podría resolver la clave privada y/o la clave efímera. No obstante, el mejor ataque conocido contra ECDSA tiene una complejidad proporcional a la raiz cuadrada del tamaño del grupo sobre el cual el DL es definido. El parámtero de ECDSA y su correspondiente nivel de seguridad están definidos en la tabla 3.1

Tamaño número primo q	*Tamaño hash*	*Nivel de seguridad*
192	192	96
224	224	112
256	256	128
384	384	192
512	512	256

Tabla 3.1: Tamaño de bits y nivel de seguridad de ECDSA

El algoritmo ECDSA requiere de claves mas pequeñas para proporcionar la misma robustez que otras como por ejemplo RSA. En la tabla 3.2 se detalla la comparación entre claves de ambos algoritmos de encriptación:

Tamaño clave RSA	*Tamaño clave ECDSA*	*Curva ESP_IDF*
1024 bits	160-223 bits	secp192r1, sec192k1
2048 bits	224-255 bits	secp224r1, sec224k1
3072 bits	256-383 bits	secp256r1, secp256k1, bp256r1
7680 bits	384-511 bits	secp384r1, bp384r1
15360 bits	512 $\geq$ bits	sepc512r1, bp512r1

Tabla 3.2: Comparación de tamaño de claves entre RSA y ECDSA

Para este trabajo se usa la curva de tipo *secp521r1* dada su gran robustez.

Ota gran diferencia entre ECDSA y RSA reside en la funcionalidad ya que este primero solo permite firma mientras que el último permite tanto firma como encriptación.

Tambien existe otra variante de algoritmo de curva elíptica, Elliptic-Curve Diffie-Hellman, basado en el algoritmo Diffie-Hellman usando curva elíptica y permitiendo el intercambio de un valor de forma segura que pueda ser usado luego en encriptación simétrica.

3.1.4. Pluggabe Authentication Module

Tal y como se comentó en la sección 1.1, la motivación de este trabajo nace de la propuesta de mejora en [5] basada en implementar en el sistema propuesto la compatibilidad con otros métodos de autenticación como los módulos PAM.

La empresa tecnológica *SunSoft* propuso en 1996 [26] un mecanismo de autenticación compatible con multiples tipos de sistemas dando capacidad para administrar no solo la autenticación sino también las sesiones y las contraseñas

Los mecanismos y protocolos de autenticación como por ejemplo SSH, Rlogin o FTP tienen como objetivo ser independientes de los mecanismos de autenticación específicos utilizados por las computadoras. No obstante, es importante que se aplique un marco que conectase todos esos mecanismos. Para ello se requiere que las aplicaciones usen una API estándar que interactúe con los servicios de autenticación. Si este mecanismo de autenticación al sistema se mantuviera independiente del usado por el computador, el administrador del sistema podría instalar módulos de autenticación adecuados sin requerir cambios en las aplicaciones.

Lo ideal en todo sistema sería aplicar un mecanismo de autenticación complejo simplemente recordando una contraseña. No obstante, los sistemas son cada vez mas heterogéneos y complejos y por ellos requieren de varios mecanismos de autenticación (problema de inicio de sesión integrado o unificado).

El objetivo reside en la integración modular de las tecnologías de autenticación de red con el inicio de sesión y otros servicios.

Las propiedades que este mecanismo debe seguir son las siguientes:

- El administrador del sistema debe poder elegir el mecanismo de autenticación por defecto
- Debe ser posible configurar la autenticación del usuario para cada aplicacióon
- Debe soportar requisitos de visuallización de las aplicaciones
- Debe ser posible configurar múltiples protocolos de autenticación
- El administrador del sistema debe poder configurar el sistema de tal forma que el usuario pueda autenticarse usando múltiples protocolos de autenticación sin tener que reescribir la contraseña

- No debe ser reconfigurado cuando el mecanismo que funcione por debajo cambie
- La arquitectura debe proveer un modelo modular de autenticación
- Debe soportar los requisitos de autenticación del sistema sobre el que opere
- La API debe ser independiente del Sistema Operativo

Los elementos principales del *framework* PAM son la API (librería de autenticación) considerada el *front-end* y el módulo de autenticación específico, el *back-end*, ambos conectados a través del SPI. El proceso consta de los siguientes pasos:

1. La aplicación escribe a la API de PAM
2. Se carga el módulo de autenticación apropiado según especifique el archivo de configuración *pam.conf*
3. La petición es enviada al módulo de autenticación correspondiente para llevar a cabo la operación concreta
4. PAM devuelve la respuesta desde el módulo de autenticación a la aplicación

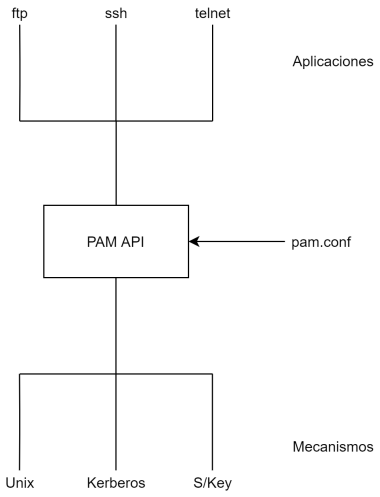

Figura 3.2: Arquitectura básica PAM

PAM unifica autenticación y control de acceso al sistema, y permite añadir módulos de autenticación a través de interfaces bien definidas. Configuración en la autenticación es un componente junto a administración de

cuenta, sesión y contraseñas. Para este trabajo, solo se va a usar la parte de autenticación ya que el resto no es necesaria. Cada una de estas áreas funcionales trabajan como módulos separados.

Dado la extensión y temática del trabajo, no se pretende dar detalles acerca del funcionamiento de la API de PAM. Simplemente anotar que para el desarrollo de este proyecto se usó las siguientes funciones de la API:

- *pam_authenticate()* [27]: función para autenticar a un usuario
- *pam_get_user()* [28]: función para obtener el nombre del usuario específico que intenta autenticarse

El archivo de configuración PAM (*pam.conf*) es la base de gestión de los módulos. Tal y como se ha mencionado antes, cuando una aplicación solicita autenticarse usando algún mecanismo que funcione con PAM, la API comprueba los módulos a ejecutar en este archivo así como la política que siguen.

El archivo de configuración A.1 se ha usado en el broker MQTT de este trabajo. Define los siguientes parámetros [29]:

- *log_dest*: ruta absoluta del archivo de logs
- *log_type*: tipo de mensajes a registrar
- *log_timestamp*: añadir valor de marca de tiempo
- *include_dir*: ruta absoluta del directorio de archivos de configuración externos
- *listener*: puerto de escucha
- *cafile*: ruta absoluta del certificado de la CA
- *certfile*: ruta absoluta del certificado del broker MQTT
- *keyfile*: ruta absoluta de la clave privada del broker MQTT
- *allow_anonymous*: permitir que clientes se puedan conectar sin proporcionar claves (usuario)

El cliente *mosquitto* escucha por defecto por el puerto 1883 para comunicaciones no seguras y por el 8883 para seguras. El primero solo se usa para comunicaciones internas (*localhost*).

La CA es una entidad que administra certificados digitales, los cuales son usados para vincular una entidad a una clave pública. Para el presente tranajo, a diferencia de [1], se ha usado el broker MQTT como CA. No obstante, la CA debe correr en un servidor independiente debido a su papel fundamental en la seguridad de toda infraestructura y para facilitar la gestión de nodos que se incorporen a la arquitectura del sistema así como revocar aquellos certificados de clientes que no tengan más acceso.

3.2. Análisis de herramientas

Para el presente trabajo se ha desarrollado el software usando el lenguaje de programación C dada la facilidad y documentación disponible a la hora de desarrollar módulos PAM en dicho lenguaje. Se ha usado *mosquitto* [19] como cliente MQTT dado su extensa documentación, por ser un software de código abierto y estar escrito en C. Con respecto al entorno de pruebas, se ha usado Vagrant [30] como orquestador de máquinas virtuales para desplegar el entorno de prueba.

Capítulo 4

Diseño de la solución

4.1. Arquitectura del sistema propuesto

El sistema propuesto está orientado a ser distribuido y escalable. Además, permite que cualquier aplicación use si esquema de autenticación. El echo de que use un servidor de autenticación centralizado simplifica la gestión de reglas de acceso y reduce complejidad en la revocación de certificados.

En la figura 4.1 se muestra cómo es la interconexión entre los distintos elementos. A este diseño hay que añadir dos elementos adicionales: el broker MQTT y el servidor de configuración, tal y como se muestra en 4.2.

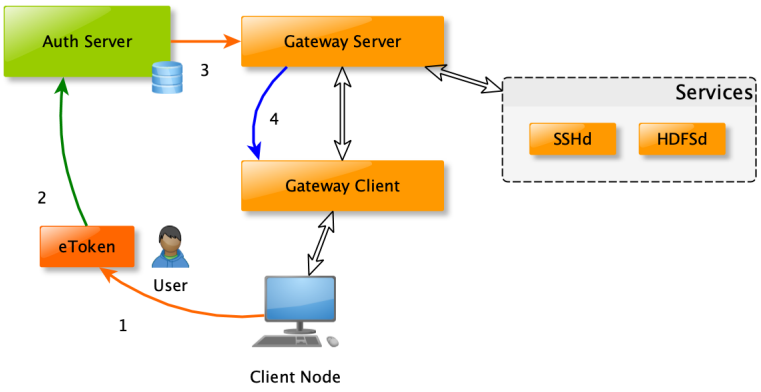

Figura 4.1: Diseño del sistema propuesto en [1]

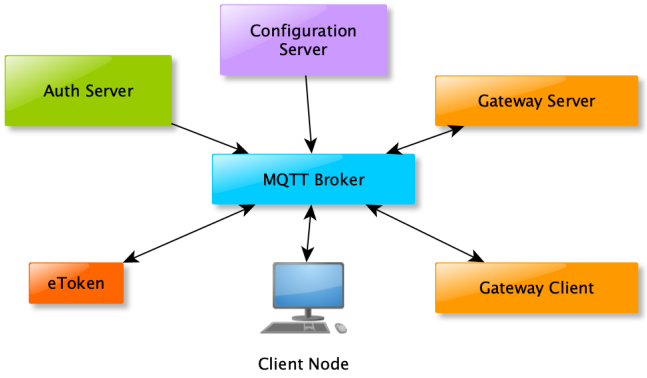

Figura 4.2: Topología MQTT

4.2. Fases de la solución

La solución propuesta en la sección1.1 se ha desarrollado siguiendo las siguientes fases: una primera etapa de preparación de entorno seguro, una segunda etapa de identificación, y una final de autenticación. Para cada una de las fases se adjunta una diagrama de secuencias [1] [2]

4.2.1. Fase TLS

En esta fase, el broker MQTT debe crear un certificado firmado por una CA para verificar su identidad. Para comunicarse por TLS con el broker MQTT, se ha llevado a cabo los siguientes pasos:

1. Crear la clave privada de la CA
2. Crer el certificado CA usando la clave privada del paso 1 para firmarla
3. Crear la clave privada del broker MQTT
4. Crear la solicitud de firma de certificados (CSR) para el broker MQTT usando la clave privada del paso 3
5. Usar la clave y certificado CA para firmar el certificado del broker MQTT
6. Enviar el certificado del paso 5 al broker MQTT
7. Distribuir el certificado CA a los cliente que se quieran comunicar a través del broker MQTT

[1] Los valores entre < y > indican un valor en concreto, no el valor definido entre ambos símbolos

[2] Los tópicos tienen la estructura emisor/receptor/item

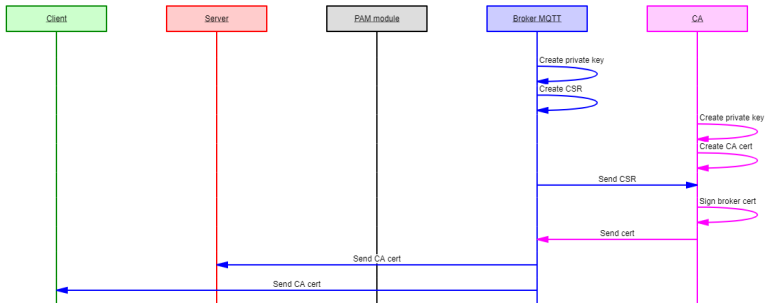

Figura 4.3: Fase TLS

Como último paso, el archivo de configuración del cliente *mosquitto* en el broker MQTT tiene que indicar la clave y certificados necesarios tal y como figura en A.1.

Si un cliente quiere usar el broker MQTT para publicar un mensaje o subscribrse a un tópico, necesita "mostrar" el certificado CA al broker MQTT. No es la única forma de identificación vía TLS. También existe la opción de que cada cliente cree su propio certificado firmados por la CA.

4.2.2. Fase de identificación

En esta fase, el cliente se tiene que registrar en el servidor para poder usar su servicio. Para ello, el cliente crea un identificador único UUID y un par de claves pública y privada. Estos se guardan en un directorio en concreto. Se ha escogido *.anubis* en la carpeta *home* del usuario. Las claves se guardan con el UUID como nombre del archivo ($< uuid > .pem$ y $< uuid > .key$) y finalmente se envía la clave pública al servidor. Se puede usar el protocolo SCP. Dado que no se quiere usar un método de autenticación distinto al propuesto por este trabajo, se podría crear una clave pública temporal del servidor y subirla a algún servidor de claves públicas. El cliente por tanto podría usarla para mandar su clave pública a su directorio *.anubis* de forma segura y una vez envíada, eliminarla. Una vez que el servidor tiene la clave pública del cliente, este lo registra en el archivo de configuración de usuario A.2. Este indica el UUID concreto para un usuario en el sistema. Se usa en caso de que el usuario tenga varias claves públicas y por tanto el servidor sepa que clave usar para la autenticación.

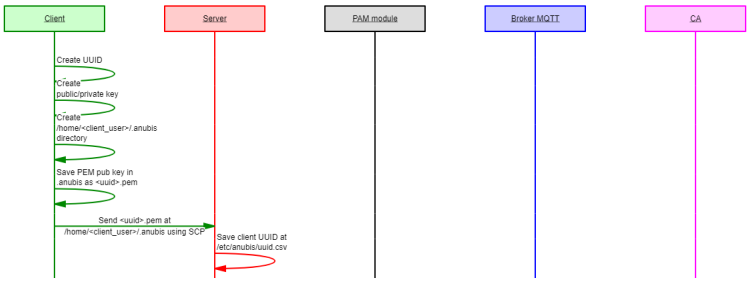

Figura 4.4: Fase de identificación

4.2.3. Fase de autenticación

En esta fase reside el proces de autenticación del cliente contra el servidor una vez que se ha establecido un canal seguro y registrado el cliente en el mismo.

El cliente se subscribe al tópico $pam/ < uuid > /challenge$ por el cual recibirá el desafío y envía una petición para abrir una sesión por SSH al servidor.

Al llegar la petición SSH al servidor, este comprueba que almacena el nombre del usuario que se quiere autenticar y comprueba su UUID en el archivo de configuración de usuarios A.2 ($/etc/anubis/uuid.csv$). Este es un archivo de valores separados por coma (CSV), el usuario y su UUID asignado. Una vez el servidor conoce el UUID, de plantean dos posibilidades:

1. Que el usuario no necesite autenticarse de la forma propuesta
2. Que el usuario tenga que autenticarse

Cada condición se da según la política definida en el archivo de configuración de anubis A.3. Existen dos tipos de políticas:

1. *relax*: no es necesario aplicar la autenticación
2. *strict*: se aplica la autenticación

La política *relax* es útil en casos en los que por ejemplo el usuario provenga de una red de confianza como puede ser una Universidad. En ese caso, el módulo PAM propuesto devolvería un *PAM_IGNORE* pasando el siguiente módulo. En caso de la política *strict*, es necesario ejecutar el proceso de autenticación propuesto y que devuelva un *PAM_SUCCESS*.

Una vez que se compruebe el UUID del usuario, si la política es de tipo *strict*, el servidor se subscribe a dos tópicos:

- $< uuid > /pam/r$
- $< uuid > /pam/s$

Por esos tópicos, el cliente publicará el par de valores $[r, s]$ del algoritmo ECDSA definido en 3.1.3 en formato hexadecimal.

Seguidamente, crea el desafío y lo publica al tópico $pam/ < uuid > /challenge$. El desafio es una cadena de 64 caracteres aleatoria compuesta de números y letras. Al llegar el mensaje al broker MQTT, este lo reenvía a todos los nodos que estén suscritos a dicho tópico. Dado que solo hay un UUID por cliente, el mensaje solo le llega al cliente determinado por el UUID.

El cliente crea el hash del desafío usando el algoritmo SHA-512 dado su robustez con respecto a otros de menor tamaño como puede ser SHA-256. Una vez que tiene el hash, lo firma usando su clave privada creada en 4.2.2 y envía ambos valores $[r, s]$ por los tópicos $< uuid > /pam/r$ y $uuid > /pam/s$ respectivamente siendo estos retransmitidos por el broker MQTT al servidor.

El servidor recibe ambos valores $[r, s]$ para crear la firma de curva elíptica y crea el hash del desafío. A continuación verifica el hash usando la firma generada anteriormente y la clave pública de tal forma que:

- Si el cliente es quien dice ser, la verificación es correcta ya que solo el cliente verídico tiene la clave privada
- Si el desafío ha sufrido alguna modificación por la intervención de una tercera persona, la verificación saldrá incorrecta

Una verificación correcta devuelve *PAM_SUCCESS* mientras que una errónea devuelve *PAM_AUTH_ERR*, siendo estas variables globales de la librería de PAM.

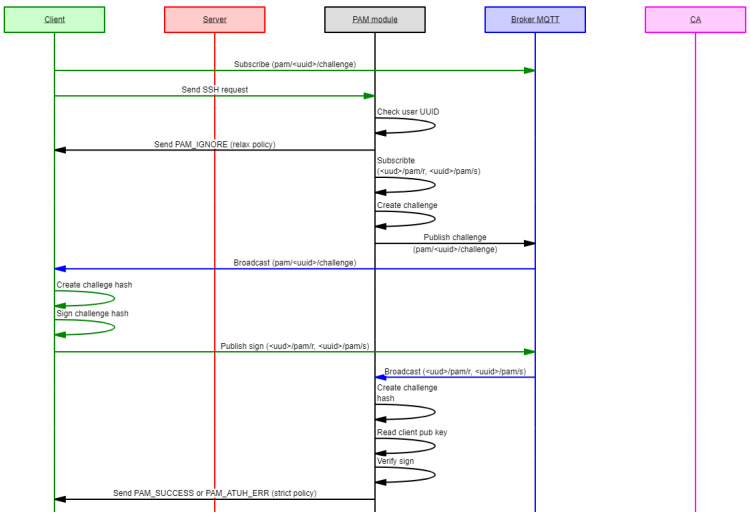

Figura 4.5: Fase de autenticación

En la siguiente imagen se mustra la topología global del sistema propuesto:

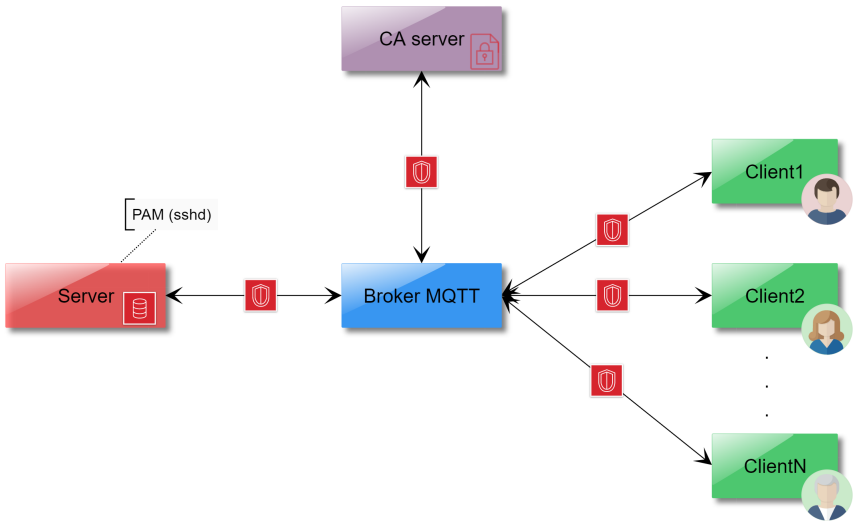

Figura 4.6: Topología del diseño

4.3. Código fuente

El código fuente no se adjunta por simplicidad y limpieza en la memoria pero se encuentra subido a la plataforma de GitHub [31].

4.4. Entorno de virtualización

4.4.1. Vagrant

Tal y como se ha mencionado en 3.2, en vez de usar un ESP-32 como eToken de autenticación y un servidor físico que recrearía un escenario real, he decido virtualizar el entorno usando máquinas virtuales gracias al programa *open-source* de Oracle *Vagrant*.

Vagrant es una herramienta destinada a la creación y configuración de entornos de desarrollo. Orginariamente fue desarrollado para que trabajase con entornos que corriesen bajo el hipervisor de *VirtualBox* [32], siendo este un software de virtualización desarrollado por la propia Oracle.

Actualmente $Vagrant$ funciona con otros hpervisores como el de Windows $Hyper-V$, el famoso $VMware$ así como entornos en la nube tales como $DigitalOcean$ o $AmazonWebServices$. Está desarrollado en Ruby pero se puede usar en proyectos de diversos lenguajes de programación.

Gracias a su facilidad de despliegue y rapidez a la hora de escalar, me ha permitido crear entornos complejos para probar el sistema propuesto.

4.4.2. Despliegue

Vagrant require de un archivo de tipo YAML para su configuración que se llama $Vagrantfile$. Para este proyecto se ha escrito el siguiente archivo de configuración A.7 compuesto de dos parte:

1. El método add_ssh_key añade la clave pública a la máquina virtual creada
2. La parte de configuración de las máquinas virtuales: broker MQTT, cliente y servidor

Para desplegar una máquina virtual en un Vagrantfile se tiene que especificar una serie de parámetros obligatorios:

- Dirección IP privada a usar. Puede ser estática o dinámica (DHCP)
- El *Vagrant Box* [33]. Esta es la imagen del sistema operativo a usar
- El proveedor (VirtualBox)
- Número y cantidad de memoria CPU
- Nombre del hostname

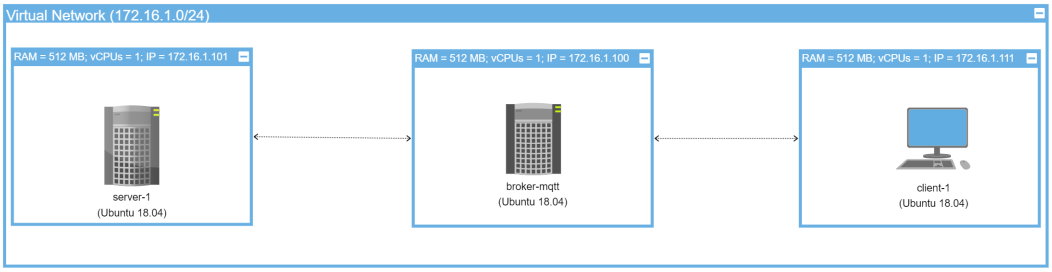

Figura 4.7: Topología del entorno virtualizado con Vagrant

Capítulo 5

Análisis de seguridad

Una vez detallado el diseño e implementación del sistema de autenticación propuesto, es necesario conocer la caracterísiticas de seguridad que implementa.

5.1. Vulnerabilidades afrontadas

Farooq propone en [34] un sistema similar al propuesto en este trabajo pero centrado en sistemas electrónicos de tipo contador inteligentes. En él, se hace un estudio de los distintas vulnerabilidades a ciberataques que el sistema evita que se sean explotados y los cuales se van a analizar contra este sistema propuesto.

5.1.1. MITM

El ataque MITM conocido como "Ataque de Hombre en el Medio" es muy común en todo sistema que use una red pública como puede ser Internet para establecer una comunicación entre dos o más integrantes. Para evitar este tipo de ataque, el sistema propuesto usa el algoritmo ECDSA para firmar el desafío usando la clave privada del cliente. Para que una tercera persona se haga pasar por este cliente, esta primera tendría que conocer dicha clave.

5.1.2. Integridad del desafío

El hash del desafío se calcula en ambas parte, cliente y servidor, sin llegar a enviarse. Por ello, si alguien qu estuviera escuchando y modificase el desafío, el cliente fimaría un hash distinto al servidor y por tanto no se verificaría.

5.1.3. Confidencialidad del mensaje

Como ya se ha mencionado en 4.2.1, la comunicación con el broker MQTT se hace vía TLS. Esto permite que los mensajes vaya cifrados por una clave. Para usar el broker MQTT y comunicarse con el servidor o viceversa, es necesario presentar un certificado. Esto garantiza que sólo usuarios legítimos se comumiquen con el servidor.

5.2. Análisis de resultados

En la figura A.5 se ve la salida del script del lado del cliente. Concretamente, los logs del cliente MQTT. Entre medias aparecen dos mensajes: el primero indica que está suscrito el tópico *pam/68263723-e928-4f71-8339-c609478f0a1a/challenge* y el segundo que ha recibido el desafío *ITOeM0joCRNR5dm.hWS5O7BaxvE8UdE7SMoPKoQck5WhhYu1di2KrBrxGsG6o76* del servidor.

Por otro lado, la figura A.4 muestra la salida de la petición SSH del cliente al servidor. El primer mensaje aivsa de que ha encontrado el UUID del cliente y su valor concreto. A continuación muestra los logs del cliente MQTT con respecto a la conexión con el broker MQTT y las subscripciones a ambos tópicos, *68263723-e928-4f71-8339-c609478f0a1a/pam/r* y *68263723-e928-4f71-8339-c609478f0a1a/pam/s*, correspondientes a los valores en hexadecimal de la firma digital de curva elíptica. Posteriormente publica el desafío creado en *pam/68263723-e928-4f71-8339-c609478f0a1a/challenge* y por último los mensajes de recepción. Al final aparece un mensaje de si la firma ha sido verificada o no y por tanto si el valor PAM devuelto es *PAM_SUCCESS* o *PAM_AUTH_ERR*.

En el lado del servidor, simplemente hay que añadir al archivo de configuración PAM de SSH A.6 la siguiente directiva: *auth required mqtt-pam.so broker.mqtt.com 8883 /etc/mosquitto/ca_certificates/ca.crt*, donde:

- *auth* indica el tipo de módulo PAM a usar (autenticación)
- *required* indica la política de ejecucion. En este caso, para que se ejecuten el resto de módulos PAM es necesario que el valor devuelto sea *PAM_SUCCESS*
- *mqtt-pam.so broker.mqtt.com 8883 /etc/mosquitto/ca_certificates/ca.crt* indica el módulo y los parámetros. Al no especificar la ruta absoluta del ejecutable, PAM busca por defecto en */lib/security/*

Capítulo 6

Presupuesto

6.1. Componentes hardware y software

Para la realización de este proyecto no se ha usado dispositvos hardware dedicados como podría ser un Arduino o Raspberry. Simplemente se ha virtualizado el entorno de pruebas mediante software de vitualización (VirtualBox) y por tanto no ha tenido coste económico ninguno. En cuanto al software, tampoco se han usado programas con licencias ya que tanto el cliente MQTT (mosquitto), la API de SSL y el entorno de desarrollo (Visual Studio Code) son de código libre.

6.2. Diagrama de Gantt

		Nombre	Duracion	Inicio	Terminado
1		Lectura tesis y PAM	42,875 days	22/10/20 8:00	21/12/20 16:00
2		Vacaciones Navidad	9 days	22/12/20 8:00	1/01/21 17:00
3		Creación de entorno de pr...	21 days?	4/01/21 8:00	1/02/21 17:00
4		Desarrollo	39 days?	2/02/21 8:00	26/03/21 17:00
5		Vacaciones de Semana Santa	6 days?	29/03/21 8:00	5/04/21 17:00
6		Desarrollo	40 days?	6/04/21 8:00	31/05/21 17:00
7		Documentación	289 days?	1/06/21 8:00	8/07/22 17:00

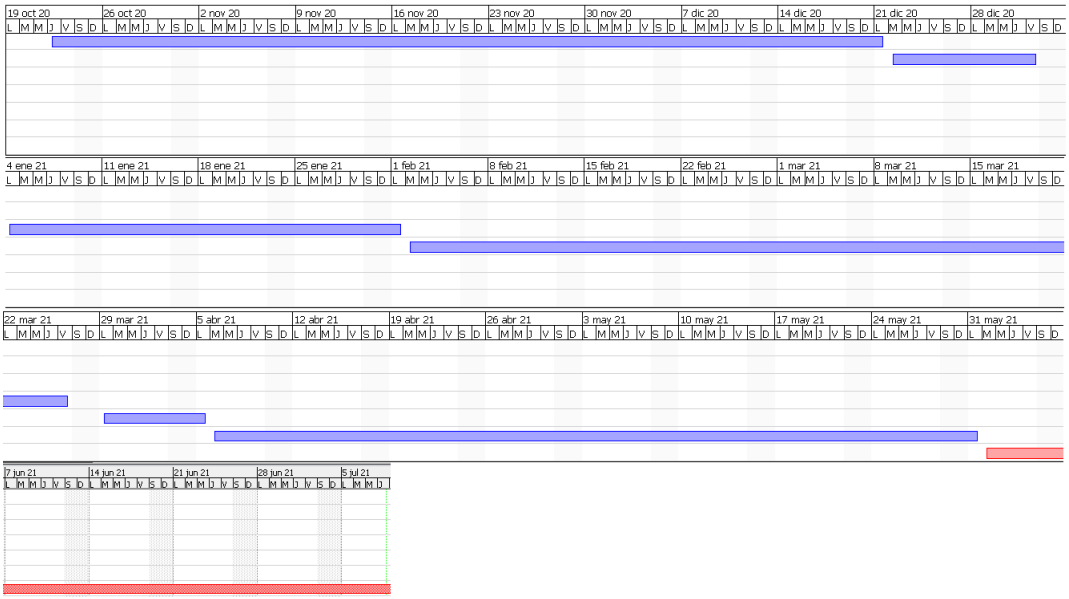

Figura 6.1: Diagrama de Gantt

Capítulo 7

Conclusión

La propuesta de mejora en [5] ha sido el punto de partida de este proyecto basado en el desarrollo de un módulo PAM que integre un sistema de autenticación para el entorno de multiautenticación propuesto en [1]. Este módulo esta pensado no solo para casos específicos como HPC o *cloud* sino para cualquiera debido a la amplia granularidad y disponibilidad que ofrecen estos módulos. El desarrollo de este proyecto me ha permitido ganar conocimiento en el campo de la seguridad informática, concretamente en criptografía e infraestructuras de clave pública, así como desarrollo en C de un módulo PAM el cual nunca había experimentado. Además, he aprendido a leer detalladamente artículos científicos y conocer a fondo las bases de un proyecto científico de alto nivel.

7.1. Problemas afrontados

Durante el desarrollo de este proyecto me he topado con diversos problemas. Al principio, me costó mucho entender el funcionamiento de PAM. Tuve que investigar en profunidad para conocer bien su funcionamiento y así poder escribir mi propio módulo.

Con respecto a la API de *mosquitto*, a pesar de que la documentación [29] está bien estructurada, tuve inconvenientes en el desarrollo de los scripts en C por la reserva de memoria que este lenguaje require.

7.2. Trabajo futuro

Como trabajo futuros se propone lo siguiente:

1. Aplicar para la política *relaxed* una condición dependiendo de la dirección IP de destino de la que provenga la petición SSH de tal forma que si es de una institución, se habilite sin llegar a autenticarse.

2. Llevar a cabo pruebas de estrés con varios accesos simultáneos desde mútliples direcciones IP al servidor para comprobar el rendimiento del protocolo MQTT y del servidor

3. Añadir una protección extra al script del servidor añadiéndole una política con SELinux

4. Tal y como se especifica en [1], usar una lista de control de acceso o ACL para indicar que puede hacer cada usuario y a qué nivel

Apéndice A

Archivos de configuración

Listing A.1: Archivo de configuración PAM

```
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example
pid_file /run/mosquitto/mosquitto.pid

log_dest file /var/log/mosquitto/mosquitto.log
log_type all
log_timestamp true

include_dir /etc/mosquitto/conf.d

listener 1883 localhost
listener 8883

cafile /etc/mosquitto/ca_certificates/ca.crt
certfile /etc/mosquitto/certs/broker-mqtt.crt
keyfile /etc/mosquitto/certs/broker-mqtt.key

allow_anonymous true
```

Listing A.2: Archivo de configuración de usuarios en /etc/anubis/uuid.csv

```
username,uuid
client-1,68263723-e928-4f71-8339-c609478f0a1a
```

Listing A.3: Archivo de configuración anubis en /etc/anubis/anubis.conf

```
# ------------------------#
# /etc/anubis/anubis.conf
# ------------------------#
#
# NOTE
# ----
#
# Configuration file for MQTT-PAM module used to authenticate via
    SSH
```

```
# Use only a space between key and value
#
# Permitted values:
# access_type [relax, strict]
# ip_address 150.214.*.*
# ------------------------#
#
# Format:
# key value

access_type relax
```

Listing A.4: Petición SSH cliente al servidor

```
vagrant@client-1:~$ ssh client-1@172.16.1.101
client-1@172.16.1.101's password:
Found UUID in user client-1: 68263723-e928-4f71-8339-c609478f0a1a
Client server_2941 sending CONNECT
Client server_2941 sending SUBSCRIBE (Mid: 1, Topic: 68263723-e928
    -4f71-8339-c609478f0a1a/pam/r, QoS: 0, Options: 0x00)
Client server_2941 sending SUBSCRIBE (Mid: 1, Topic: 68263723-e928
    -4f71-8339-c609478f0a1a/pam/s, QoS: 0, Options: 0x00)
Client server_2941 sending PUBLISH (d0, q0, r0, m2, 'pam/68263723-
    e928-4f71-8339-c609478f0a1a/challenge', ... (64 bytes))
Client server_2941 received CONNACK (0)
Client server_2941 received SUBACK
Client server_2941 received PUBLISH (d0, q0, r0, m0, '68263723-
    e928-4f71-8339-c609478f0a1a/pam/r', ... (130 bytes))
Client server_2941 received PUBLISH (d0, q0, r0, m0, '68263723-
    e928-4f71-8339-c609478f0a1a/pam/s', ... (130 bytes))
Successfully verified
Exiting...
Client server_2941 sending DISCONNECT
PAM OK
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-74-generic x86_64)

* Documentation:  https://help.ubuntu.com
* Management:     https://landscape.canonical.com
* Support:        https://ubuntu.com/advantage

System information as of Mon Jun 21 21:19:03 UTC 2021

System load:  0.0               Processes:               109
Usage of /:   7.0% of 38.71GB   Users logged in:         1
Memory usage: 41%               IPv4 address for enp0s3: 10.0.2.15
Swap usage:   0%                IPv4 address for enp0s8:
    172.16.1.101

* Super-optimized for small spaces - read how we shrank the memory
footprint of MicroK8s to make it the smallest full K8s around.

https://ubuntu.com/blog/microk8s-memory-optimisation

38 updates can be applied immediately.
To see these additional updates run: apt list --upgradable


Last login: Mon Jun 21 21:11:24 2021 from 172.16.1.111
client-1@server-1:~$
```

Listing A.5: Salida script cliente

```
vagrant@client-1:~/tfg/bin$ ./client broker.mqtt.com 8883
    68263723-e928-4f71-8339-c609478f0a1a /etc/mosquitto/
    ca_certificates/ca.crt
Client 68263723-e928-4f71-8339-c609478f0a1a sending CONNECT
Client 68263723-e928-4f71-8339-c609478f0a1a sending SUBSCRIBE (Mid
    : 1, Topic: pam/68263723-e928-4f71-8339-c609478f0a1a/challenge
    , QoS: 0, Options: 0x00)
Listening to pam/68263723-e928-4f71-8339-c609478f0a1a/challenge
    topic...
Client 68263723-e928-4f71-8339-c609478f0a1a received CONNACK (0)
Client 68263723-e928-4f71-8339-c609478f0a1a received SUBACK
Client 68263723-e928-4f71-8339-c609478f0a1a received PUBLISH (d0,
    q0, r0, m0, 'pam/68263723-e928-4f71-8339-c609478f0a1a/
    challenge', ... (64 bytes))
Received challenge: ITOeM0joCRNR5dm.
    hWS5O7BaxvE8UdE7SMoPKoQck5WhhYu1di2KrBrxGsG6o76
Client 68263723-e928-4f71-8339-c609478f0a1a sending PUBLISH (d0,
    q0, r0, m2, '68263723-e928-4f71-8339-c609478f0a1a/pam/r', ...
    (130 bytes))
Client 68263723-e928-4f71-8339-c609478f0a1a sending PUBLISH (d0,
    q0, r0, m3, '68263723-e928-4f71-8339-c609478f0a1a/pam/s', ...
    (130 bytes))
Exiting...
Client 68263723-e928-4f71-8339-c609478f0a1a sending DISCONNECT
```

Listing A.6: Archivo de configuración PAM para sshd

```
# PAM configuration for the Secure Shell service

# MQTT PAM module
auth required mqtt-pam.so broker.mqtt.com 8883 /etc/mosquitto/
    ca_certificates/ca.crt

# Standard Un*x authentication.
@include common-auth

# Disallow non-root logins when /etc/nologin exists.
account    required     pam_nologin.so

# Uncomment and edit /etc/security/access.conf if you need to set
    complex
# access limits that are hard to express in sshd_config.
# account  required     pam_access.so

# Standard Un*x authorization.
@include common-account

# SELinux needs to be the first session rule.  This ensures that
    any
# lingering context has been cleared.  Without this it is possible
     that a
# module could execute code in the wrong domain.
session [success=ok ignore=ignore module_unknown=ignore default=
    bad]        pam_selinux.so close

# Set the loginuid process attribute.
session    required     pam_loginuid.so

# Create a new session keyring.
```

```
session    optional     pam_keyinit.so force revoke

# Standard Un*x session setup and teardown.
@include common-session

# Print the message of the day upon successful login.
# This includes a dynamically generated part from /run/motd.
    dynamic
# and a static (admin-editable) part from /etc/motd.
session    optional     pam_motd.so  motd=/run/motd.dynamic
session    optional     pam_motd.so noupdate

# Print the status of the user's mailbox upon successful login.
session    optional     pam_mail.so standard noenv # [1]

# Set up user limits from /etc/security/limits.conf.
session    required     pam_limits.so

# Read environment variables from /etc/environment and
# /etc/security/pam_env.conf.
session    required     pam_env.so # [1]
# In Debian 4.0 (etch), locale-related environment variables were
    moved to
# /etc/default/locale, so read that as well.
session    required     pam_env.so user_readenv=1 envfile=/etc/
    default/locale

# SELinux needs to intervene at login time to ensure that the
    process starts
# in the proper default security context.  Only sessions which are
     intended
# to run in the user's context should be run after this.
session [success=ok ignore=ignore module_unknown=ignore default=
    bad]        pam_selinux.so open

# Standard Un*x password updating.
@include common-password
```

Listing A.7: Archivo de configuración Vagrantfile

```
1   n_servers = 1
2   n_clients = 1
3
4   server_net = "172.16.1.10"
5   client_net = "172.16.1.11"
6   broker_mqtt_net = "172.16.1.100"
7
8   def add_ssh_key(config)
9       ssh_pub_key = File.readlines("#{Dir.home}/.ssh/id_rsa.pub").
            first.strip
10      config.vm.provision "shell" do |s|
11        s.inline = <<-SHELL
12          echo #{ssh_pub_key} >> /home/vagrant/.ssh/authorized_keys
13          mkdir -p /root/.ssh/
14          chmod 700 /root/.ssh/
15          echo #{ssh_pub_key} >> /root/.ssh/authorized_keys
16        SHELL
17      end
18  end
19
```

```
20      Vagrant.configure("2") do |config|
21          config.vm.synced_folder ".", "/home/vagrant/tfg"
22
23          (1..n_servers).each do |i|
24              config.vm.define "server-#{i}" do |node|
25                  node.vm.network :private_network, ip: "#{server_net}#{
                        i}"
26                  node.vm.box = "ubuntu/focal64"
27                  node.vm.provider "virtualbox" do     |pmv|
28                      pmv.memory = 512
29                      pmv.cpus   = 1
30                  end
31                  node.vm.hostname = "server-#{i}"
32                  add_ssh_key(config)
33              end
34          end
35
36          (1..n_clients).each do |i|
37              config.vm.define "client-#{i}" do |node|
38                  node.vm.network :private_network, ip: "#{client_net}#{
                        i}"
39                  node.vm.box = "ubuntu/focal64"
40                  node.vm.provider "virtualbox" do |pmv|
41                      pmv.memory = 512
42                      pmv.cpus   = 1
43                  end
44                  node.vm.hostname = "client-#{i}"
45                  add_ssh_key(config)
46              end
47
48          end
49
50          config.vm.define "broker-mqtt" do |node|
51              node.vm.network :private_network, ip: "#{broker_mqtt_net}"
52              node.vm.box = "ubuntu/focal64"
53              node.vm.provider "virtualbox" do |pmv|
54                  pmv.memory = 512
55                  pmv.cpus   = 1
56              end
57              node.vm.hostname = "broker-mqtt"
58              add_ssh_key(config)
59          end
60      end
```

Bibliografía

[1] Antonio Francisco Díaz García, Ilia Blokhin, Mancia Anguita López, Julio Ortega Lopera, and Juan José Escobar Pérez. Multiprotocol authentication device for hpc and cloud environments based on elliptic curve cryptography, 7 2020.

[2] AJ Neumann, N Statland, and RD Webb. *Post Processing Audit Tools and Techniques*. 1977.

[3] Digital Attack Map. `https://www.digitalattackmap.com`.

[4] CVE - CVE. `https://cve.mitre.org/`.

[5] Ilia Blokhin. Mecanismos de seguridad para big data basados en circuitos criptográficos, 2020.

[6] C Newman, A Menon-Sen, A Melnikov, and N Williams. Salted challenge response authentication mechanism (scram) sasl and gss-api mechanisms. *Internet Requests for Comments, RFC Editor, RFC*, 5802, 2010.

[7] An Braeken. Puf based authentication protocol for iot. *Symmetry*, 10(8):352, 2018.

[8] Henk CA Van Tilborg and Sushil Jajodia. *Encyclopedia of cryptography and security*. Springer Science & Business Media, 2014.

[9] You Alex Gao, Jim Basney, and Alex Withers. Scitokens ssh: Token-based authentication for remote login to scientific computing environments. In *Practice and Experience in Advanced Research Computing*, pages 465–468. 2020.

[10] Achraf Fayad. *Secure authentication protocol for Internet of Things*. PhD thesis, Institut Polytechnique de Paris, 2020.

[11] google/google-authenticator-libpam. `https://github.com/google/google-authenticator-libpam`, June 2021.

[12] YubiKey. `https://www.yubico.com/la-yubikey/?lang=es`.

[13] Single sign-on: What is it & how does it work? https://www.onelogin.com/learn/how-single-sign-on-works.

[14] Bases de la seguridad informática | Seguridad Informática. http://descargas.pntic.mec.es/mentor/visitas/demoSeguridadInformatica/bases_de_la_seguridad_informtica.html.

[15] Aleksandr Ometov, Sergey Bezzateev, Niko Mäkitalo, Sergey Andreev, Tommi Mikkonen, and Yevgeni Koucheryavy. Multi-factor authentication: A survey. *Cryptography*, 2(1):1, 2018.

[16] Christina Braz and Jean-Marc Robert. Security and usability: the case of the user authentication methods. In *Proceedings of the 18th Conference on l'Interaction Homme-Machine*, pages 199–203, 2006.

[17] Somayya Madakam, Vihar Lake, Vihar Lake, Vihar Lake, et al. Internet of things (iot): A literature review. *Journal of Computer and Communications*, 3(05):164, 2015.

[18] KALAIVANAN SUGUMAR. Mqtt-a lightweight communication protocol relative study. *Authorea Preprints*, 2020.

[19] Eclipse Mosquitto. https://mosquitto.org/, January 2018.

[20] HiveMQ - Enterprise ready MQTT to move your IoT data. https://www.hivemq.com/.

[21] moscajs/mosca. https://github.com/moscajs/mosca, June 2021.

[22] CloudMQTT - Hosted message broker for the Internet of Things. https://www.cloudmqtt.com/.

[23] MQTT.js. https://github.com/mqttjs.

[24] Kevin S McCurley. The discrete logarithm problem. In *Proc. of Symp. in Applied Math*, volume 42, pages 49–74. USA, 1990.

[25] Christof Paar and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[26] Vipin Samar. Unified login with pluggable authentication modules (pam). In *Proceedings of the 3rd ACM conference on Computer and communications security*, pages 1–10, 1996.

[27] pam_sm_authenticate(3) - Linux manual page. https://man7.org/linux/man-pages/man3/pam_sm_authenticate.3.html.

[28] pam_get_user(3) - Linux manual page. https://man7.org/linux/man-pages/man3/pam_get_user.3.html.

[29] mosquitto.conf man page. `https://mosquitto.org/man/mosquitto-conf-5.html`, June 2021.

[30] Vagrant by HashiCorp. `https://www.vagrantup.com/`.

[31] Sergio García. sergiogp98/mqtt-pam. `https://github.com/sergiogp98/mqtt-pam`, June 2021.

[32] Oracle VM VirtualBox. `https://www.virtualbox.org/`.

[33] Vagrant Boxes. `https://app.vagrantup.com`.

[34] Shaik Mullapathi Farooq, SM Suhail Hussain, and Taha Selim Ustun. Elliptic curve digital signature algorithm (ecdsa) certificate based authentication scheme for advanced metering infrastructure. In *2019 Innovations in Power and Advanced Computing Technologies (i-PACT)*, volume 1, pages 1–6. IEEE, 2019.

Acrónimos

ACL Access Control List

ANSI American National Standards Institute

API Application Program Interface

CA Cerificate Authority

CSR Certificate Signing Request

DLP Discrete Logarithm Problem

ECC Elliptic Curve Criptography

ECDSA Elliptic Curve Digital Signature Algorithm

FTP File Transport Protocol

HMAC Hash Message Authentication Code

HPC High Performance Computing

HTTP Hyper Text Transfer Protocol

IoT Internet of Things

JWT JSON Web Token

MFA Multiple Factor Authentication

MITM Man In The Middle

MQTT Message Queueing Transport Telemetry

NIST National Institute of Standards and Technology

OTP One Time Password

PAM Pluggabe Authentication Module

PKI Public Key Infrastructure

PUF Physical Unclonable Functions

RAE Real Academia Española

Rlogin Remote Login

RSA Rivest-Shamir-Adleman

SCP Secure Copy Protocol

SHA Secure Hash Algorithm

SPI Service Provider Interface

SSH Secure Shell

SSO Single Sign-On

TLS Transport Layer Security

TOTP Time-based One Time Password

TTP Trusted Third Party

UUID Universally Unique IDentifier

YAML YAML Ain't Markup Language