ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

CAN BUS ENCRIPTADO

Autor: Daniel Levy Moreno

Director: Prof. Wayne T Padgett

Madrid

Junio 2018

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

CAN BUS Encriptado

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2017/18 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Daniel Levy Moreno Fecha: ……/ ……/ ……

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Francisco Javier Herraiz Martínez Fecha: ……/ ……/ ……

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D._____

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: _____, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente).*

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.

b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.
d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a …….. de ……... de …….

ACEPTA

Fdo……

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

BUS CAN ENCRIPTADO

Autor: Daniel Levy Moreno

Director: Prof. Wayne T Padgett

Madrid

Junio 2018

Agradecimientos

Firstly, I want to thank Rose-Hulman´s ECE department for giving me the amazing opportunity to work on this project. It has been an incredible journey and learning experience in which I got to truly develop my engineering skills. I also want to thank the menbers of senior design team 16, Paul Earhart, Haoran Geng, Dustin Kline and Larry Kozlowski; for providing invaluable assistance throughout the projects development. Lastly, I want to thank Rolls-Royce cybersecurity division, who were the original proposers of the project and its raison d'etre.

The work presented in this memoir would have been impossible without them.

CAN BUS ENCRIPTADO

Autor: Levy Moreno, Daniel
Director: Prof. Wayne T. Padgett
Entidad Colaboradora: Rolls-Royce Holdings plc

RESUMEN DEL PROYECTO

En este proyecto se propone el uso de técnicas criptográficas embebidas en el protocolo Controller Area Network (CAN), comúnmente utilizado en la industria del automóvil como protocolo de comunicación para el diagnóstico a bordo de control de emisiones. Se explorarán los motivos de la implementación de dichas técnicas desde una perspectiva de ciberseguridad, se discutirá su grado de practicidad y por último se analizaran los resultados obtenidos en la implementación elegida para el proyecto.

Palabras clave: CAN, Criptografía, Ciberseguridad.

1. Introducción

El protocolo CAN es actualmente y ha sido el protocolo de comunicación estándar de la industria del automóvil desde los años 90. Este es dado uso en el bus CAN, un bus de comunicaciones en serie para aplicaciones de control en tiempo real, cuya instalación es obligatoria en todos los vehículos desde 2008. El protocolo CAN también se utiliza para el diagnóstico a bordo de control de emisiones. A través del puerto OBD-II, cuya instalación es obligatoria en los vehículos desde 1996 [1], es posible observar las comunicaciones en tiempo real de los diferentes componentes del automóvil, así como interactuar con ellos mediante el envío de tramas a través del mismo puerto.

Dada la estandarización masiva de la tecnología CAN, su practicidad en la industria es incuestionable. Sin embargo, como se demostrará más adelante, el bus CAN posee serios riesgos de seguridad que podrían tener consecuencias desastrosas si no se abordan, especialmente teniendo en cuenta los recientes esfuerzos en investigación y desarrollo de nuevas tecnologías en la industria automotriz, como el automóvil autómata [2] que acentuará los riesgos existentes. Por lo tanto, la implementación de técnicas criptográficas en el bus CAN surge como una medida preventiva a estos riesgos, cuyos desafíos y limitaciones serán

explorados, tratando de responder la pregunta "¿Es la criptografía una opción práctica dentro del protocolo CAN?".

Este proyecto podría considerarse una carta abierta a los líderes de la industria automotriz, con la esperanza de que se preste más atención a los riesgos de ciberseguridad discutidos en este documento, con la esperanza de que pueda motivar más investigación hacia mecanismos de seguridad que sean capaces de reducirlos. La solución criptográfica presentada en este documento es solo una entre innumerables posibilidades. Es el deseo de todos los participantes de que sirva al menos como inspiración de otras soluciones propuestas en proyectos de similar temática.

2. Definición del proyecto

Antes de analizar la solución criptográfica, es clave entender el problema que se está abordando. A través del puerto OBD-II, una entidad separada de los miembros de confianza de la red de bus CAN no solo es capaz de acceder a todas las comunicaciones enviadas a través del bus, sino que también puede enviar paquetes firmados con cualquiera de sus identidades. De esta forma, utilizando ingeniería inversa [3], el bus CAN es vulnerable a un ataque de "spoofing" [4].

Aunque este ataque está lejos de ser el único al que el bus CAN es vulnerable [5], es el foco principal de la solución implementada. La razón principal de este riesgo es la falta de verificación de la firma ID de un mensaje CAN dado. Por lo tanto, un mensaje que posee la firma de un determinado miembro en el campo ID de trama de CAN siempre se interpreta como procedente de dicho miembro. Es por ello que la criptografía aplicada a CAN pretende verificar las identidades de los comunicantes tras recibir un mensaje cualquiera.

En este proyecto, no solo se investigará la viabilidad práctica de las técnicas criptográficas aplicadas al protocolo CAN, teniendo en cuenta las limitaciones de CAN y el análisis de los efectos sobre el rendimiento del sistema, sino que se prestará especial atención a la solución desarrollada para garantizar que es seguro además de un mecanismo de seguridad eficaz contra un ataque de "spoofing".

3. Descripción del sistema/modelo/herramienta

Un bus CAN está compuesto por "Nodos", por la "Engine Control Unit" (ECU) y dos cables entrelazados, CAN-H y CAN-L, que forman las líneas conectoras del bus, juntándose en una resistencia de 120Ω. [6] Un "Nodo" está compuesto por una unidad de control, normalmente un sensor, que envía su información atreves de un "CAN Transceiver", que procesa los datos recibidos por la unidad de control y los convierte en señales eléctricas que envía a través de las líneas de bus CAN. Adicionalmente, un "CAN Transceiver" también es capaz de procesar las señales de las líneas del bus y traducirlas de forma que puedan ser procesadas por su respectiva unidad de control.

La "Engine Control Unit" también posee un "CAN transciever" y una unidad de control, los cuales reciben y analizan los datos enviados por el resto de nodos, además de una serie de actuadores que, dependiendo de los datos recibidos, son accionados con el fin de conseguir el rendimiento óptimo del motor [7]. Los cables "CAN-H" y "CAN-L", encargados de transmitir las comunicaciones del bus en forma de señales eléctricas, mandan la misma secuencia de datos, pero con amplitudes opuestas, lo que dificulta la corrupción de datos debida al ruido.

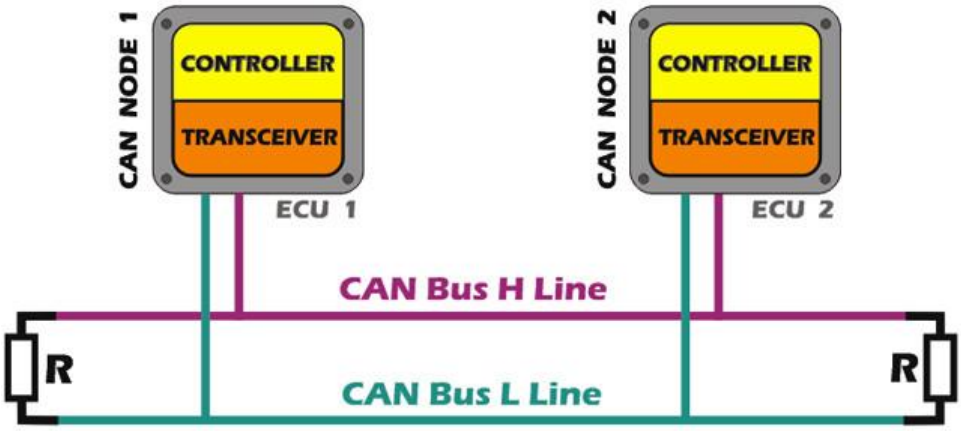

Figura 1 - Esquema del bus CAN [6]

Los campos del protocolo CAN están definidos por ISO 11898-1. Existen dos variantes, "Standard CAN", que utiliza 11 bits en el campo ID, y "Extended CAN" que utiliza 29 bits. Otros campos comunes a las dos variantes son 1 bit para indicar el comienzo del mensaje, 6

bits de control con 4 bits dedicados a la longitud de cargo útil, hasta 64 bits de carga útil, 16 bits de CRC, 2 bits de ACK y 7 bits para indicar el final de trama. [8]

Figura 2 - Esquema de los campos del protocolo CAN [6]

Todos los "Nodos" conectados a la red CAN reciben todas las tramas transmitidas por el bus CAN y dependiendo del valor ID, deciden aceptar o ignorar el mensaje. Si varios nodos quieren transmitir un mensaje al mismo tiempo, se da prioridad al mensaje con ID más bajo y el resto de los nodos deben esperar hasta que la red este libre para comenzar a transmitir.

4. Resultados

Primeramente, como parte elemental de la implementación, se necesitaba un modelo análogo al sistema bus CAN mediante el cual se pudiera hacer pruebas de riesgo de ciberseguridad en un espacio seguro y controlado. Este sistema, alias "sistema análogo", debería reflejar tanto en la arquitectura hardware como en la arquitectura software la estructura y funcionamiento de un sistema bus CAN genérico. Su arquitectura, una vez fue definida en la fase de diseño, está compuesta por tres Arduino Nano [9] actuando como diferentes "Nodos" y un BeagleBone Black Rev C [10] actuando como "ECU". Los Arduino utilizan módulos CAN MCP2515 TJA1050 [11] como "transciever", a los que se conectan por SPI, y el BeagleBone Black un TI-SN65HVD231D que hace de driver para la regulación de la capa física [12]. Todos los "trancievers" están conectados con dos cables entrelazados a mano, actuando los papeles de "CAN-H" y "CAN-L".

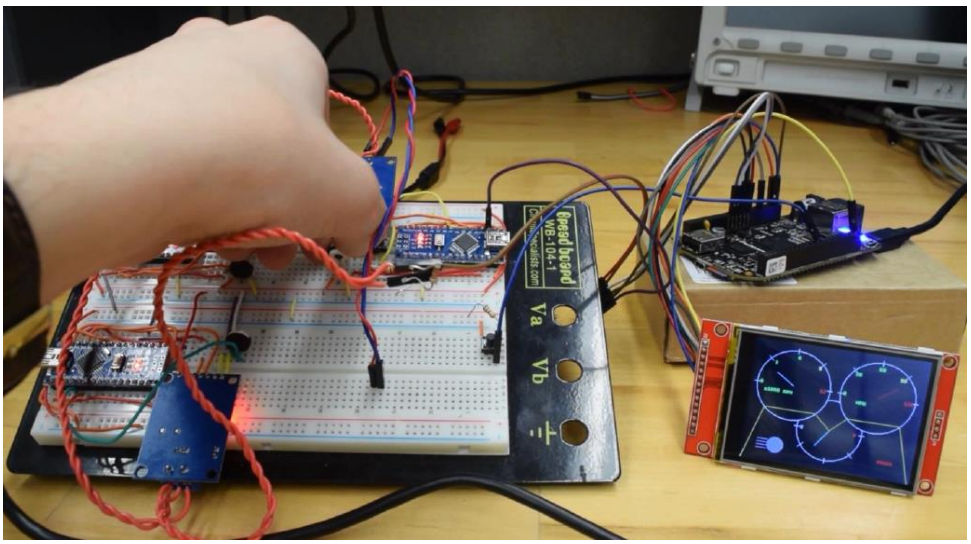

Figura 3 - Sistema análogo

Para la capa software se utilizaron varias librerías "Open Source". Para los "Arduino Nano", se utilizó CAN Bus Shield [13] que ofrece una interfaz de comunicación con su "transciever" CAN MCP2515 TJA1050. Para el BeagleBone Black, tras habilitar su capa CAN [14], se utilizó la librería socketCAN [15] [16] que permite enviar tramas CAN con la estructura socket.

Una vez construido el sistema análogo, se inició la fase de diseño del mecanismo de defensa criptográfico contra el ataque de ID "spoofing". Este consiste en unir un valor secreto, la llave, solamente conocido entre el emisor y receptor, a la firma del campo ID. Esta llave es utilizada para encriptar los mensajes entre estas dos entidades y un agente que desconozca su valor es incapaz por lo tanto de falsificar sus identidades. Adicionalmente, se consigue proteger la confidencialidad de los mensajes mandados, añadiendo gran dificultad al proceso de ingeniería inversa por el cual se puede obtener información adicional sobre el efecto sobre el motor de los distintos mensajes.

Sin embargo, todos los beneficios conseguidos por la implementación son invalidados si por cualquier motivo el valor de la llave secreta se hace público, ya que volveríamos al estado original de la cuestión. Por ello, no solo es importante que la técnica criptográfica utilizada para convertir el texto plano a texto cifrado sea segura, sino que además es absolutamente crítico que el acuerdo de llave entre comunicantes sea realizado de manera confidencial. En

este último punto es donde se ha hecho mayor énfasis y ha consumido los mayores recursos y tiempo de trabajo.

Teniendo en cuenta las limitaciones en memoria y computación impuestas por el sistema análogo, se optó por un diseño que no utilizara criptografía asimétrica [17] para la negociación de valor de llave entre "Nodo" y "ECU", dada a su alto coste computacional. Es por ello que tomando inspiración en el "Challange Handshake Authetication Protocol" (CHAP) utilizado en la "Virtual Private Networks" (VPN), se decidió diseñar una adaptación de este algoritmo orientado a CAN, al que se le dio el alias de "CAN-CHAP". Este algoritmo da una plataforma a un "Nodo" y a la "ECU" de verificar mutuamente sus identidades, a través de desafíos matemáticos y respuestas pre-calculadas. Una vez sus identidades son verificadas, las dos entidades pueden obtener el valor de la llave secreta combinando los valores de los mensajes compartidos y valores secretos que solo ellos dos conocen. [18]

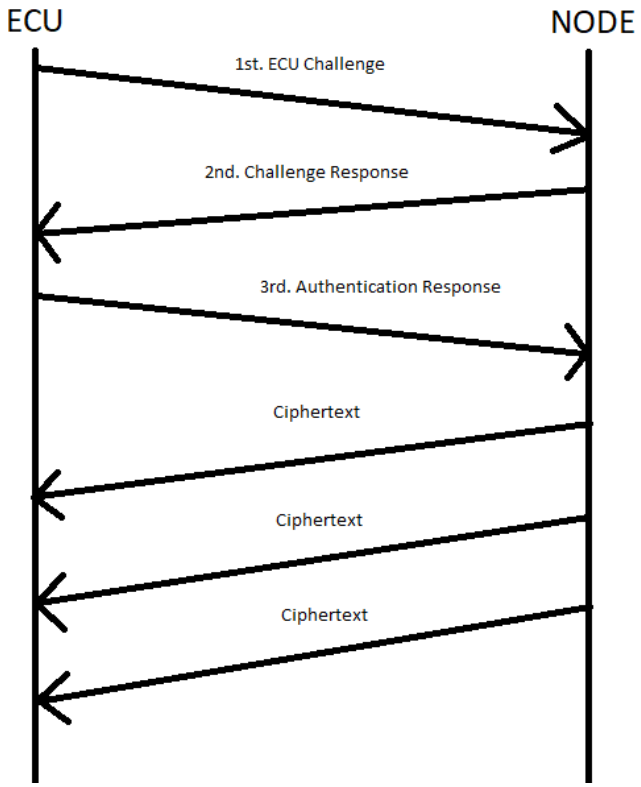

Figura 4 – Diagrama funcional de CAN-CHAP

La rapidez de computación de los componentes que forman parte de "CAN-CHAP" no fue el único factor que se tuvo en cuenta durante la fase de diseño, ya que la limitada carga útil de CAN, de únicamente 64 bits, forzaría el uso de múltiples mensajes por emisor si se utilizarán algoritmos que obtuvieran resultados mayores que la carga útil, lo cual tendría un efecto notable sobre el rendimiento del sistema. Para la fase de verificación se utilizó el "Keyed-Hash" [19] "sipHash" [20], ya que fue diseñado para resultados de exactamente 64 bits. Para crear el valor de la llave, a través de la técnica conocida como "key stretching" [21], el algoritmo "blake2s" [22] fue utilizado, ya que su implementación en Arduino Nano era significativamente más rápida que la de los hashes más tradicionales de la familia "SHA" [23].

Por último, tras la fase de verificación, los mensajes transmitidos serían encriptados con él algoritmo simétrico "ChaCha20" [24], no solo por ser la implementación más rápida entre los candidatos probados en el Arduino Nano, sino que además por la libertad que ofrece de limitar la longitud de su texto cifrado resultante, que se restringió a 64 bits. Tanto "blake2s" como "ChaCha20" tienen una implementación optimizada para la plataforma Arduino en la librería "open-source" oficial criptográfica de Arduino [25], que fue utilizada para CAN-CHAP.

La implementación de "CAN-CHAP", realizada sobre un modelo de 1 "nodo" y una "ECU" por simplicidad, resulto en un tiempo medio de 9900 ± 50µs para la mutua verificación de identidad y negociación del valor secreto. La tabla 1 detalla los valores obtenidos por los distintos componentes y fases de "CAN-CHAP". Durante el capítulo 5 de esta memoria se explicará más detalladamente los principios de diseño de "CAN-CHAP", sus limitaciones.

Step	Arduino Nano	Beagle Bone Black
Random Number Generation (Normal Distribution Value Selection)	-	60 ± 2µs
Hashing (SipHash)	650 ± 20µs	63 ± 1µs
Encryption (ChaCha20)	60 ± 6µs / 900 ± 6µs *	66 ± 1µs
Key Extension (Blake2s)	700 ± 20µs	65 ± 2µs

Tabla 1 - Tiempos obtenidos.

5. Conclusiones

Pese a la rápida negociación de llaves obtenida en CAN-CHAP, su mayor factor de riesgo es su vulnerabilidad a ataques de "replay" [26]. Es por ello por lo que es esencial asegurar la entropía del método por el que se obtenga el número arbitrario que comienza la fase de negociación y del cual dependen el resto de los mensajes enviados entre "nodo" y "ECU".

La carga útil de 64 bits es el mayor factor limitante en cuanto al diseño de la implementación. Al haber optado por eficiencia, restringiendo la longitud del texto plano a la de la carga útil, la solución obtenida es considerablemente más vulnerable a ataques de "cumpleaños" [27]. Otra consideración a tener en cuenta es el efecto al rendimiento de cabeceras extra al rendimiento. Para CAN-CHAP, se utilizó 1 byte de contador para asegurar la sincronización entre "Nodo" y "ECU", lo que resulto en un 12.5% de pérdida adicional del rendimiento.

Aunque no se hizo una implementación que utilizará criptografía asimétrica, se investigaron componentes para un posible diseño denominado la solución "hardware". Esta consistiría en utilizar microcontroladores especializados en el uso de técnicas criptográficas para la fase de negociación de llaves, utilizando entre otras, criptografía asimétrica. Un componente que demostró ser muy prometedor fue el "ESP32" [28], dado a la facilidad de programación y amplio soporte por la comunidad online.

Sin embargo, dado a las restricciones de tiempo a las que el proyecto estaba sometido durante su desarrollo, no hubo la posibilidad de realmente explorar profundamente esta posibilidad. Es por ello que, si otros proyectos deciden perseguir una temática parecida a la de este, sería de gran interés desarrollar al completo la solución "Hardware" y contrastarla con los resultados obtenidos a través de CAN-CHAP.

6. Referencias

[1] Augeri, Fernando. *Protocolo de Comunicación CAN*, Magazine, Septiembre 2010, Source: http://www.cise.com/portal/notas-tecnicas/item/166-protocolo-de-comunicaci%C3%B3n-can.html

[2] ABC MOTOR. *Las marcas que lideran la carrera para desarrollar coches autónomos, y las más retrasadas*, Magazine, junio 2018. Source:

http://www.abc.es/motor/reportajes/abci-marcas-cabeza-hacia-coche-autonomo-y-mas-retrasadas-201801230137_noticia.html

[3] Currie, Roderick. T. *Hacking the CAN Bus: Basic Manipulation of a Modern Automobile Through CAN Bus Reverse Engineering*, SANS Institute paper, Mayo 2017. Source: https://www.sans.org/reading-room/whitepapers/threats/hacking-bus-basic-manipulation-modern-automobile-through-bus-reverse-engineering-37825

[4] DuPaul, Neil. *Spoofing Attack: IP, DNS & ARP*, Web Blog, Source: https://www.veracode.com/security/spoofing-attack

[5] Evenchick, Eric. *Hopping on the CAN Bus: Automotive Security and the CANard Toolkit*, BlackHat Asia presentation material, Source: http://www.blackhat.com/docs/asia-15/materials/asia-15-Evenchick-Hopping-On-The-Can-Bus.pdf

[6] Mucevski, Kiril. *Automotive CAN Bus System Explained*, linkedIn article, Source: https://www.linkedin.com/pulse/automotive-can-bus-system-explained-kiril-mucevski/

[7] Pro Car Mechanics, *How an Engine Control Unit Affects Performance*. Web Blog, Source: http://procarmechanics.com/how-an-engine-control-unit-affects-performance/

[8] Dahikar, Shrikant, *Basic of CAN (Controller Area Network) for vehicle ECU communication*, linkedIn article, Source: https://www.linkedin.com/pulse/basic-can-controller-area-network-vehicle-ecu-shrikant-dahikar/

[9] ARDUINO, *ARDUINO NANO*, Web Store, Source: https://store.arduino.cc/usa/arduino-nano

[10] adafuit, *BeagleBone Black Rev C*, Web Store, Source: https://www.adafruit.com/product/1876

[11] dx, *Módulo de bus CAN MCP2515 TJA1050 Módulo receptor SPI*, Web Store, Source: http://www.dx.com/es/p/mcp2515-can-bus-module-tja1050-receiver-spi-module-blue-434988#.Wy57-FUzaDI

[12] mangoboard, *CAN Transciever Module*, Web Store, Source: http://www.mangoboard.com/main/view.aspidx=424&pageNo=1&cate1=9&cate2=165&cate3=

[13] github, *CAN Bus Shield*, Web Reposotary, Source: https://github.com/Seeed-Studio/CAN_BUS_Shield

[14] Webdemeyer, Thomas. *BeagleBone CANbus & Python*, Blog, Source: http://www.thomas-wedemeyer.de/beaglebone-canbus-python.html

[15] github, *CAN utils*, Web Repository, Source: https://github.com/linux-can/can-utils

[16] linklayer, SocketCan, Web Wiki, Source: https://wiki.linklayer.com/index.php/SocketCAN

[17] De Luz, Sergio. *Criptografía: Algoritmos de cifrado de clave asimétrica*. Web Blog, Source: https://www.redeszone.net/2010/11/16/criptografia-algoritmos-de-cifrado-de-clave-asimetrica/

[18] Rouse, Margaret. *CHAP (Challenge Handshake Authetication Protocol)*. Web Wiki, Source: https://searchsecurity.techtarget.com/definition/CHAP-Challenge-Handshake-Authentication-Protocol

[19] Krawczyk, H, Bellare, M, Canetti. HMAC: Keyed-Hashing for Message. Public Memo, Source : https://tools.ietf.org/html/rfc2104

[20] Aumasson, Jean-Philippe, J. Bernstein, Daniel. *SipHash: a fast short-input PRF*. Web Blog, Source: https://131002.net/siphash/

[21] Spacey, John. *What is Key Streching?*, Web Blog, Source: https://simplicable.com/new/key-stretching-definition

[22] Aumasson, Jean-Philippe, Neves, Samuel, Wilcox-O´Hearn, Zooko, Winnerlein, Christian. *Blake2 – fast secure hashing*, Web Blog, Source: https://blake2.net/

[23] Landman, Nathan. Williams, Chistopher. Ross, Elli. *Secure Hash Algorthms*. Web Blog, Source: https://brilliant.org/wiki/secure-hashing-algorithms/

[24] Y. Nir, Check Point, A.Langley, ChaCha20 and Poly1305 for IETF Protocols, Scienctific Surver, Source: https://tools.ietf.org/html/rfc7539

[25] github, *Arduino Cryptography Library*, Web Repository, Source: https://rweather.github.io/arduinolibs/crypto.html

[26] KK, *Replay-Attack*, Web Wiki, Source: http://www.crypto-it.net/eng/attacks/replay.html

[27] Miessler, Daniel, The Birthday Attack, Web Blog, Source: https://danielmiessler.com/study/birthday_attack/

[28] Espressif, *ESP32 Overview*, Web Market, Source: https://www.espressif.com/en/products/hardware/esp32/overview

ENCRIPTED CAN BUS

Author: Levy Moreno, Daniel.
Supervisor: Prof. Wayne T. Padgett
Collaborating Entity: Rolls-Royce Holdings plc

ABSTRACT

This project proposes the implementation of cryptographic techniques embedded within the Controller Area Network (CAN) protocol, commonly used in the automobile industry as a communication protocol for the on-board diagnostics of emissions control. The necessity for these techniques will be explored from a cybersecurity perspective, their practicality will be discussed and lastly the results of this project´s chosen implementation will be analyzed.

Keywords: CAN, Criptography, Cibersecurity

1. Introduction

The CAN protocol is currently and has been the standard communication protocol of the automotive industry since the 90s. It is used in the CAN bus, a serial communications bus for real-time control applications, whose installation is mandatory in all vehicles since 2008. The CAN protocol is also used for on-board diagnostics in emission control. Through the OBD-II port, whose installation is mandatory in vehicles since 1996 [1], it is possible to observe the real-time communications of the different components of the car, as well interact with them using the same port.

Given the massive standardization of CAN technology, its practicality in the industry is unquestionable. However, as it will be shown later on, the CAN bus possesses serious security risks that could result in disastrous consequences if they are not addressed, especially considering the efforts being done in research and development of new technologies in the automotive industry, such as the self-driven car [2], which will introduce new risk factors. Thus, the implementation of cryptographic techniques in the CAN bus emerges as a preventive measure against these risks, whose challenges and limitations will

be explored, trying to answer the question "Is cryptography a practical option in the CAN protocol?".

This project could be considered an open letter to the heads of the automotive industry, hoping that attention is raised to the cybersecurity risks discussed in this paper, perhaps motivating more research into competent safety mechanisms that could reduce them. The cryptographic solution presented in this paper is only one among innumerable possibilities. It is the desire of all the participants that it serves at least as inspiration for other solutions proposed in projects of similar theme.

2. Project definition

Before discussing the cryptographic solution, it is key to understand the problem that is being addressed. Through the OBD-II port, an entity separate to the trusted members of the CAN bus network is not only capable of accessing all communications sent through the bus but is also able to send packets signed with any of their identities. This way, using reverse engineering [3], the CAN bus is vulnerable to a "spoofing" attack [4].

Although this attack is far from the only one the CAN bus is vulnerable to [5], it is the sole focus of the implemented solution. The main cause for this risk is the lack of verification of the ID signature of a given CAN message. Thus, a message that has the signature of a certain member in the CAN´s frame ID field is always interpreted as coming from said member. Cryptography applied to CAN intends to verify the identities of the communicators once their transmissions have been received.

In this project, not only will the practical viability of the cryptographic techniques applied to the CAN protocol be researched, considering CAN´s limitations and analyzing the effects on system performance, but also that special attention will be given to the developed solution to ensure that is safe and, therefore, is an effective security mechanism against a "spoofing" attack.

3. Model/System/tool description

A CAN bus is made of "Nodes", an "Engine Control Unit" (ECU) and two interconnected cables, CAN-H and CAN-L, which form the connected bus lines, which join into a 120Ω resistance. [6] A "Node" is composed of a control unit, usually a sensor, that sends its information through a "CAN Transceiver", which processes the data received by the control unit and the transforms them into electrical signals sent through the CAN bus lines. Additionally, a "CAN Transceiver" is also capable of receiving the signals from the bus lines and translate them to be processed by its respective control unit.

The "Engine Control Unit" also has a "CAN transceiver" and a control unit, which receive and analyze the data sent by the rest of the nodes, in addition to a series of actuators that, depending on the data received, are operated with in order to maintain the optimal performance of the motor [7].The "CAN-H" and "CAN-L" cables, responsible for transmitting bus communications in the form of electrical signals, send the same sequence of data, but with opposite amplitudes, which makes data corruption due to noise difficult.

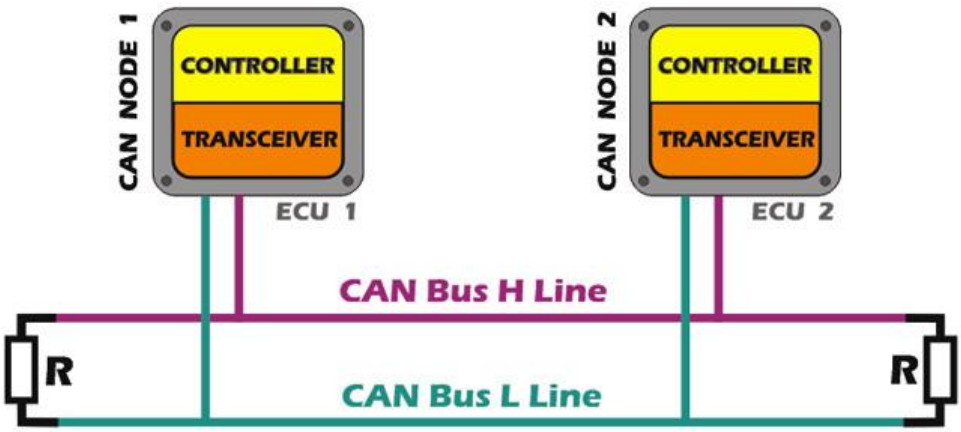

Illustration 1 - CAN bus scheme [6]

The CAN protocol fields are defined by ISO 11898-1. There are two variants, "Standard CAN", which uses 11 bits in the ID field, and "Extended CAN" that uses 29 bits. Other fields common to the two variants are 1 bit to indicate the start of the message, 6 control bits with 4 bits dedicated to the payload length, up to 64 payload bits, 16 CRC bits, 2 ACK bits and 7-bit sequence to indicate the end of frame. [8]

Illustration 2 - CAN Protocol overhead scheme [6]

All "Nodes" connected to the CAN network receive all the frames transmitted through the CAN bus and depending on the ID value, they can decide to accept or ignore the message. If several nodes want to transmit a message at the same time, priority is given to the message with the lowest ID and the rest of the nodes must wait until the network is free to start transmitting.

4. Results

Firstly, as an elementary part of the implementation, a model analogous to the CAN bus system was needed, through which cybersecurity risk tests could be done in a safe and controlled space. This model, alias "analogous system", had to reflect the structure and operation of a generic CAN bus system both in the hardware architecture and in the software architecture. Its structure, once defined in the design phase, is composed of three Arduino Nano [9] acting as different "Nodes" and a BeagleBone Black Rev C [10] deployed as an "ECU". The Arduino use CAN modules MCP2515 TJA1050 [11] as "transceiver", to which they connect by SPI, and the BeagleBone Black a TI-SN65HVD231D as driver for the regulation of the physical layer [12]. All the "transceivers" connect with two cables interlaced by hand, acting as "CAN-H" and "CAN-L".

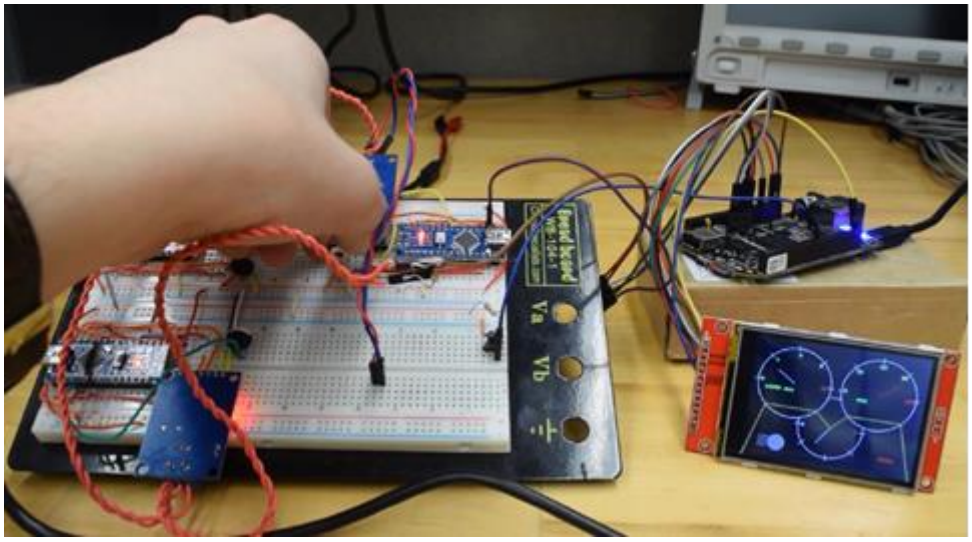

Photo 1 - Analogous System

Several "Open Source" libraries were used for the software layer. For the "Arduino Nano", CAN Bus Shield [13] was used, which offers a communication interface with its "transciever" CAN MCP2515 TJA1050. For the BeagleBone Black, after enabling its CAN layer [14], the socketCAN library [15] [16] was used to send CAN frames through the socket structure.

Once the analog system was built, the design phase of the cryptographic defense mechanism against the "spoofing" ID attack was initiated. This consists of joining a secret value, the

key, only known between the sender and receiver, to the signature of the ID field. This key is used to encrypt the messages between these two entities and an agent who does not know its value is therefore unable to falsify their identities. Additionally, the confidentiality of the messages sent is maintained, adding great difficulty to the reverse engineering process by which additional information can be obtained about the effect on the engine of the different messages.

However, all the benefits obtained by the implementation are invalidated if for any reason the value of the secret key is made public, since we would return to the original state of the matter. Therefore, it is not only important that the cryptographic technique used to convert plain text to encrypted text is safe, but it is also critical that the key agreement between communicators is made confidentially. Is in this last point where the greater focus has been placed and has consumed the greatest amount of resources and time.

Considering the limitations in memory and computation imposed by the analog system, we chose a design that did not use asymmetric cryptography [17] for the negotiation of key value between "Node" and "ECU", given its high computational cost. That is why, taking inspiration from the "Challenge Handshake Authentication Protocol" (CHAP) used in the "Virtual Private Networks" (VPN), it was decided to design an adaptation of this algorithm oriented to CAN, which was given the alias of " CAN-CHAP. " This algorithm gives a platform to a "Node" and to the "ECU" to mutually verify their identities, through mathematical challenges and pre-calculated responses. Once their identities are verified, the two entities can obtain the value of the secret key by combining the values of the shared messages and secret values that only the two know. [18]

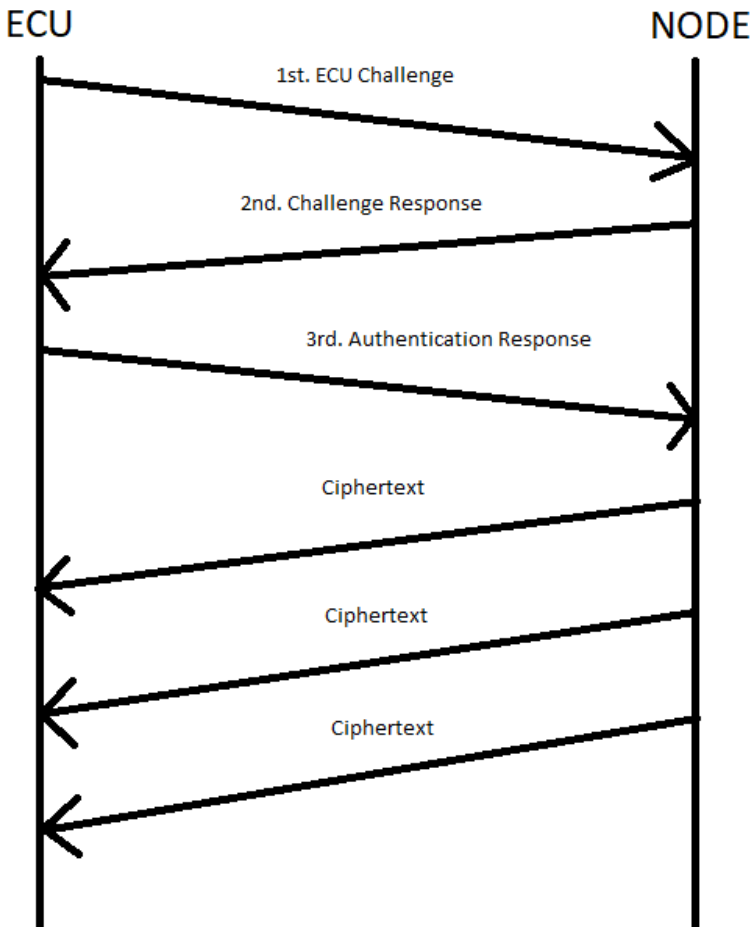

Figura 4 – Diagrama funcional de CAN-CHAP

The speed of computation of the components that are part of "CAN-CHAP" was not the only factor that was considered during the design phase, since the limited CAN payload, of only 64 bits, would force the use of multiple messages per emitter if algorithms that obtain results greater than the payload were used, which would have a noticeable effect on the performance of the system. For the verification phase the "Keyed-Hash" [19] "sip Hash" [20] was used, since it was designed for results of exactly 64 bits. To create the value of the key, through the technique known as "key stretching" [21], the algorithm "blake2s" [22] was used, since its implementation in Arduino Nano was significantly faster than that of the hashes more traditional family "SHA" [23].

Finally, after the verification phase, the transmitted messages would be encrypted with the symmetric algorithm [24] "ChaCha20", not only because it is the fastest implementation

among the candidates tested on the Arduino Nano, but also because of the freedom it offers to limit the length of its resulting ciphertext, which was restricted to 64 bits. Both "blake2s" and "ChaCha20" have an optimized implementation for the Arduino platform in the official "open-source" cryptographic library of Arduino [25], which was used for CAN-CHAP.

The implementation of "CAN-CHAP", carried out on a model of 1 "node" and an "ECU" for simplicity, resulted in an average time of 9900 ± 50µs for the mutual identity verification and negotiation of the secret value. Table 1 details the values obtained by the different components and phases of "CAN-CHAP". During chapter 5 of this report, the design principles of "CAN-CHAP" and its limitations will be explained in more detail.

Step	Arduino Nano	Beagle Bone Black
Random Number Generation (Normal Distribution Value Selection)	-	60 ± 2µs
Hashing (SipHash)	650 ± 20µs	63 ± 1µs
Encryption (ChaCha20)	60 ± 6µs / 900 ± 6µs *	66 ± 1µs
Key Extension (Blake2s)	700 ± 20µs	65 ± 2µs

Tabla 1 - Tiempos obtenidos.

5. Conclusion

Despite the fast negotiation of keys obtained in CAN-CHAP, its biggest risk factor is its vulnerability to "replay" attacks [26]. That is why it is essential to ensure the entropy of the method by which the arbitrary number that begins the negotiation phase is obtained and on which the rest of the messages sent between "node" and "ECU" depend.

The 64-bit payload is the biggest limiting factor in the design of the implementation. Having opted for efficiency, restricting the length of the plain text to that of the payload, the solution obtained is considerably more vulnerable to "birthday" attacks [27]. Another consideration to consider is the effect on performance of extra headers to performance. For CAN-CHAP, 1 counter byte was used to ensure synchronization between "Node" and "ECU", which resulted in a 12.5% additional loss of overall performance.

Although an implementation that would use asymmetric cryptography was not made, components were investigated for a possible design called the "hardware" solution. This

would consist of using microcontrollers specialized in the use of cryptographic techniques for the key negotiation phase, using, among others, asymmetric cryptography. One component that proved very promising was the "ESP32" [28], given the ease of programming and broad support by the online community.

However, given the time constraints to which the project was subjected during its development, there was no cahnce of deeply exploring this possibility. That is why, if other projects decide to pursue a theme similar to this one, it would be of great interest to develop the complete "Hardware" solution and contrast it with the results obtained through CAN-CHAP.

6. References

[1] Augeri, Fernando. *Protocolo de Comunicación CAN*, Magazine, Septiembre 2010, Source: http://www.cise.com/portal/notas-tecnicas/item/166-protocolo-de-comunicaci%C3%B3n-can.html

[2] ABC MOTOR. *Las marcas que lideran la carrera para desarrollar coches autónomos, y las más retrasadas*, Magazine, Junio 2018. Source: http://www.abc.es/motor/reportajes/abci-marcas-cabeza-hacia-coche-autonomo-y-mas-retrasadas-201801230137_noticia.html

[3] Currie, Roderick. T. *Hacking the CAN Bus: Basic Manipulation of a Modern Automobile Through CAN Bus Reverse Engineering*, SANS Institute paper, Mayo 2017. Source: https://www.sans.org/reading-room/whitepapers/threats/hacking-bus-basic-manipulation-modern-automobile-through-bus-reverse-engineering-37825

[4] DuPaul, Neil. *Spoofing Attack: IP, DNS & ARP*, Web Blog, Source: https://www.veracode.com/security/spoofing-attack

[5] Evenchick, Eric. *Hopping on the CAN Bus: Automotive Security and the CANard Toolkit*, BlackHat Asia presentation material, Source: http://www.blackhat.com/docs/asia-15/materials/asia-15-Evenchick-Hopping-On-The-Can-Bus.pdf

[6] Mucevski, Kiril. *Automotive CAN Bus System Explained*, linkedIn article, Source: https://www.linkedin.com/pulse/automotive-can-bus-system-explained-kiril-mucevski/

[7] Pro Car Mechanics, *How an Engine Control Unit Affects Performance*. Web Blog, Source: http://procarmechanics.com/how-an-engine-control-unit-affects-performance/

[8] Dahikar, Shrikant, *Basic of CAN (Controller Area Network) for vehicle ECU communication*, linkedIn article, Source: https://www.linkedin.com/pulse/basic-can-controller-area-network-vehicle-ecu-shrikant-dahikar/

[9] ARDUINO, *ARDUINO NANO*, Web Store, Source: https://store.arduino.cc/usa/arduino-nano

[10] adafuit, *BeagleBone Black Rev C*, Web Store, Source: https://www.adafruit.com/product/1876

[11] dx, *Módulo de bus CAN MCP2515 TJA1050 Módulo receptor SPI*, Web Store, Source: http://www.dx.com/es/p/mcp2515-can-bus-module-tja1050-receiver-spi-module-blue-434988#.Wy57-FUzaDI

[12] mangoboard, *CAN Transciever Module*, Web Store, Source: http://www.mangoboard.com/main/view.aspidx=424&pageNo=1&cate1=9&cate2=165&cate3=

[13] github, *CAN Bus Shield*, Web Repository, Source: https://github.com/Seeed-Studio/CAN_BUS_Shield

[14] Webdemeyer, Thomas. *BeagleBone CANbus & Python*, Blog, Source: http://www.thomas-wedemeyer.de/beaglebone-canbus-python.html

[15] github, *CAN utils*, Web Repository, Source: https://github.com/linux-can/can-utils

[16] linklayer, SocketCan, Web Wiki, Source: https://wiki.linklayer.com/index.php/SocketCAN

[17] De Luz, Sergio. *Criptografía: Algoritmos de cifrado de clave asimétrica*. Web Blog, Source: https://www.redeszone.net/2010/11/16/criptografia-algoritmos-de-cifrado-de-clave-asimetrica/

[18] Rouse, Margaret. *CHAP (Challenge Handshake Authetication Protocol)*. Web Wiki, Source: https://searchsecurity.techtarget.com/definition/CHAP-Challenge-Handshake-Authentication-Protocol

[19] Krawczyk, H, Bellare, M, Canetti. HMAC: Keyed-Hashing for Message. Public Memo, Source: https://tools.ietf.org/html/rfc2104

[20] Aumasson, Jean-Philippe, J. Bernstein, Daniel. *SipHash: a fast short-input PRF*. Web Blog, Source: https://131002.net/siphash/

[21] Spacey, John. *What is Key Streching?*, Web Blog, Source: https://simplicable.com/new/key-stretching-definition

[22] Aumasson, Jean-Philippe, Neves, Samuel, Wilcox-O´Hearn, Zooko, Winnerlein, Christian. *Blake2 – fast secure hashing*, Web Blog, Source: https://blake2.net/

[23] Landman, Nathan. Williams, Chistopher. Ross, *Elli. Secure Hash Algorithms*. Web Blog, Source: https://brilliant.org/wiki/secure-hashing-algorithms/

[24] Y. Nir, Check Point, A.Langley, ChaCha20 and Poly1305 for IETF Protocols, Scienctific Surver, Source: https://tools.ietf.org/html/rfc7539

[25] github, *Arduino Cryptography Library*, Web Repository, Source: https://rweather.github.io/arduinolibs/crypto.html

[26] KK, *Replay-Attack*, Web Wiki, Source: http://www.crypto-it.net/eng/attacks/replay.html

[27] Miessler, Daniel, The Birthday Attack, Web Blog, Source: https://danielmiessler.com/study/birthday_attack/

[28] Espressif, *ESP32 Overview*, Web Market, Source: https://www.espressif.com/en/products/hardware/esp32/overview

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

El bus CAN (Controller Area Network) es en la actualidad el protocolo estándar utilizado por la industria del automóvil desde los años noventa. La rápida adopción y uso masivo del protocolo CAN demuestra su alto nivel práctico en la industria. Sin embargo, avances en la tecnología digital crean nuevos factores de riesgo en la ciberseguridad de los automóviles, los cuales deben ser considerados al decidir que protocolo utilizar en el vehículo. Es por ello que Rolls-Royce, asignó al equipo 16 de "Senior Design" de Rose-Hulman Institute of Technology, investigar los riesgos de seguridad del protocolo CAN y seguidamente, aplicar mecanismos de defensa capaces de minimizarlos en la medida de lo posible.

Para investigar los posibles riesgos, parte del equipo elaboró un sistema de ataque cuyo objetivo era causar daños en la integridad, accesibilidad o confidencialidad del sistema, esencialmente actuando el papel un agente malicioso que intenta hackear el bus CAN del automóvil. El sistema se conecta a través del puerto On-board Diagnostics (OBD-II) [1], fácilmente accesible por cualquier dispositivo que posea una interfaz CAN y cuya implementación es obligatoria en todos los automóviles manufacturados en EEUU desde 1996. Diversos ataques fueron diseñados e implementados. Entre ellos, el ataque “Imitación de ID” ("ID Spoofing") y las repercusiones que conllevaba, fueron los progenitores de este proyecto.

Este ataque consiste en escuchar las comunicaciones que están dando lugar en el bus y seleccionar un ID presente en la red. Seguidamente, utilizando el sistema de ataque, se mandan mensajes a través del bus con el ID seleccionado. Estos mensajes son interpretados por la unidad ECU como mensajes originados por el agente con el ID seleccionado. De esta manera, se puede usurpar la identidad de cualquier ID presente en la red.

Este ataque es especialmente peligroso por varios motivos. En primer lugar, es muy difícil de detectar, la unidad ECU no tiene ningún modo de realmente verificar la identidad de cada ID y solamente los agentes cuyos IDs han sido usurpados son capaces detectar el ataque una

vez ya ha ocurrido. En segundo lugar, es un ataque directo y letal contra la integridad del sistema. Un automóvil está compuesto de numerosos componentes críticos para su correcto funcionamiento. Información no verificada, maliciosamente otorgada a la unidad ECU podría producir daños irreparables al automóvil.

1. MOTIVACIÓN DEL PROYECTO

Este proyecto desea investigar la posibilidad de crear una red CAN encriptada, con el objetivo de otorgar al bus CAN una medida preventiva contra el ataque de "Imitación de ID". El proyecto busca un compromiso razonable entre seguridad criptográfica y rapidez de procesamiento. Al final del proyecto, debería existir una respuesta definitiva a la pregunta, "¿Es una implementación de una red CAN encriptado no solo posible sino practica?".

La respuesta a esta pregunta no es solo relevante a para la industria del automóvil sino también para otras industrias que utilizan protocolos similarmente limitados en carga útil. Avances en técnicas criptográficas, utilizadas al nivel del microcontrolador, podrían asegurar la confidencialidad e integridad de la información obtenida por estos. Esto es especialmente relevante considerando el cada vez más cercano estado tecnológico del "Internet Of Things" ("IoT") [2], en el cual será critico encontrar técnicas de bajo coste computacional y de mínimo efecto al rendimiento para verificar la información obtenida por estos dispositivos, antes de utilizar dicha información.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

1. CRIPTOLOGÍA

La criptología es la disciplina que se dedica al estudio de la escritura secreta, es decir, estudia los mensajes que, procesados de cierta manera, se convierten en difíciles o imposibles de leer por entidades no autorizadas. La criptología comprende varias disciplinas, pero relevante a este proyecto es en primer lugar la criptografía [3], que ocupa del estudio de los algoritmos, protocolos y sistemas que se utilizan para proteger la información y dotar de seguridad a las comunicaciones y a las entidades que se comunican y en segundo lugar el criptoanálisis [4], que se ocupa de conseguir capturar el significado de mensajes construidos mediante criptografía sin tener autorización para ello.

Sendas disciplinas requieren el uso extensivo de complicadas operaciones matemáticas por los individuos que las practiquen. Sin embargo, gracias a la invención de la computadora, estas operaciones pueden ser automatizadas y realizadas en tiempos inalcanzables para un ser humano. Esto ha permitido la rápida evolución de sendas disciplinas durante las últimas décadas. Cabe destacar que los algoritmos criptográficos no solo se han vuelto más complejos, pero también más eficientes, salvando coste de procesamiento en cuanto viene a implementación. Para este proyecto, se consideraron las siguientes técnicas criptográficas:

1.1. CRIPTOGRAFÍA SIMÉTRICA

También llamada criptografía de clave secreta o criptografía de una clave es un método criptográfico en el cual se usa una misma clave para cifrar y descifrar mensajes en el emisor y el receptor. [5]. Las dos partes que se comunican han de ponerse de acuerdo de antemano sobre la clave a usar. Una vez que ambas partes tienen acceso a esta clave, la remitente cifra un mensaje usando la clave, lo envía al destinatario, y éste lo descifra con la misma clave.

Estos algoritmos suelen ser computacionalmente ligeros, lo que lo hacen ideales considerando las limitaciones técnicas impuestas por los componentes utilizados. Sin embargo, estos algoritmos solo son considerados realmente seguros si:

1. La clave es conocida únicamente por el emisor y el receptor. Esto puede parecer bastante obvio, sin embargo, es extremadamente importante enfatizar la importancia de la secrecía de la llave. La seguridad conseguida por la implementación de cualquier técnica criptográfica está en gran medida determinada por el nivel de opacidad que se consigue sobre el valor de la clave. En otras palabras, un sistema criptográfico que falla en ocultar el valor de la clave no es un sistema criptográfico, es un sistema sin uso.
2. La clave es únicamente utilizada una vez por cada mensaje. Como anteriormente mencionado, la criptografía simétrica es computacionalmente ligera, lo que la hace mucho más vulnerable a ataques de criptoanálisis. Para aumentar exponencialmente la complejidad de estos algoritmos, es práctica común cambiar la llave después de cada uso. Un simple contador que aumenta con cada uso y se añade al valor original de la llave antes de su utilización es suficiente para enmascarar el valor de la misma contra un ataque de criptoanálisis. Sin embargo, esto es si y solo si el valor original de cada llave es realmente aleatorio cada vez que ocurre un acuerdo entre el emisor y el receptor.

Muchos estudios que investigan técnicas criptográficas con microprocesadores 8 bit, similares al del Arduino Nano, centran sus esfuerzos en crear código de programación lo más eficaz posible de algoritmos ampliamente adoptados por la industria de la ciberseguridad, con la esperanza de hacerlos viables en estas plataformas tan limitadas, con mucho éxito. En las librerías Crypto para Arduino [6] existen implementaciones de algoritmos como AES, 3DES, XTEA entre otros, utilizados en diversos sectores de la seguridad de redes, que consiguen tiempos de cifrado y descifrado consistentemente por debajo del milisegundo.

1.2. CRIPTOGRAFÍA ASIMÉTRICA

También llamada criptografía de clave pública o criptografía de dos claves es el método criptográfico que usa un par de claves para el envío de mensajes. [5] Las dos claves pertenecen a la misma persona que ha enviado el mensaje. Una clave es pública y se puede entregar a cualquier persona, la otra clave es privada y el propietario debe guardarla de modo que nadie tenga acceso a ella. Además, los métodos criptográficos garantizan que esa pareja de claves sólo se puede generar una vez, de modo que se puede asumir que no es posible que dos personas hayan obtenido casualmente la misma pareja de claves.

Si una persona que remite un mensaje a un destinatario usa la llave pública de este último para cifrarlo; una vez cifrado, sólo la clave privada del destinatario podrá descifrar el mensaje, ya que es el único que debería conocerla. Por lo tanto, se logra la confidencialidad del envío del mensaje, nadie salvo el destinatario puede descifrarlo. Cualquiera, usando la llave pública del destinatario, puede cifrarle mensajes; los que solo serán descifrados por el destinatario usando su clave privada.

Esta técnica sería ideal para este proyecto sino fuera por el hecho de que son computacionalmente muy costosas, lo que las hacen totalmente imprácticas en su implementación con los componentes del sistema análogo. Cabe mencionar que muchos estudios se han hecho en implementaciones ligeras de estos algoritmos, con cierto nivel de mérito. Sin embargo, estas implementaciones disminuyen el nivel de complejidad de los algoritmos hasta tal punto que los hacen demasiado vulnerables a ataques de criptoanálisis y, por lo tanto, no fueron utilizadas en este proyecto.

1.3. FUNCIÓN HASH

También conocida como función resumen o función digest, es un algoritmo matemático que transforma cualquier bloque arbitrario de datos en una nueva serie de caracteres con una longitud fija [7]. Independientemente de la longitud de los datos de entrada, el valor hash de salida tendrá siempre la misma longitud.

Actualmente, las funciones hash tienen usos muy diversos. En el contexto de este proyecto, el uso más importante es el de la verificación de información. Ilustrare este uso con un ejemplo. Digamos que Alice ha estado mandando varios mensajes cifrados a Bob utilizando criptografía asimétrica. Alice quiere asegurarse de que Bob realmente está descifrando y entendiendo sus mensajes, pero quiere mantener la confidencialidad del medio. Sin embargo, Bob no tiene una clave para cifrar los mensajes, ya que su clave únicamente puede descifrar mensajes. Con el objetivo de respetar estas restricciones, Alice manda en su último mensaje cifrado instrucciones de cómo hacer una función hash y le indica a Bob que le mande a ella un hash del mensaje que acaba de recibir. Bob sigue las instrucciones de Alice y le manda su hash. Alice, que calculo su hash sobre el mensaje con antelación, verifica el valor de este comparándolo con el suyo y al ser iguales, puede asegurar que Bob ha entendido su mensaje.

Con este ejemplo teórico se observa claramente que estas funciones pueden ser utilizadas para verificar la identidad de los comunicantes, sin revelar ninguna información conocida por estas entidades. El único requisito es que la información que desea verificar sea exactamente idéntica para ambas entidades.

2. *BUS CAN*

Un bus CAN está compuesto por "Nodos", por la "Engine Control Unit" (ECU) y dos cables entrelazados, CAN-H y CAN-L, que forman las líneas conectoras del bus, juntándose en una resistencia de 120Ω. [8] Un "Nodo" está compuesto por una unidad de control, normalmente un sensor, que envía su información atreves de un "CAN Transceiver", que procesa los datos recibidos por la unidad de control y los convierte en señales eléctricas que envía a través de las líneas de bus CAN. Adicionalmente, un "CAN Transceiver" también es capaz de procesar las señales de las líneas del bus y traducirlas de forma que puedan ser procesadas por su respectiva unidad de control.

La "Engine Control Unit" también posee un "CAN transciever" y una unidad de control, los cuales reciben y analizan los datos enviados por el resto de los nodos, además de una serie de actuadores que, dependiendo de los datos recibidos, son accionados con el fin de

conseguir el rendimiento óptimo del motor [9]. Los cables "CAN-H" y "CAN-L", encargados de transmitir las comunicaciones del bus en forma de señales eléctricas, mandan la misma secuencia de datos, pero con amplitudes opuestas, lo que dificulta la corrupción de datos debida al ruido.

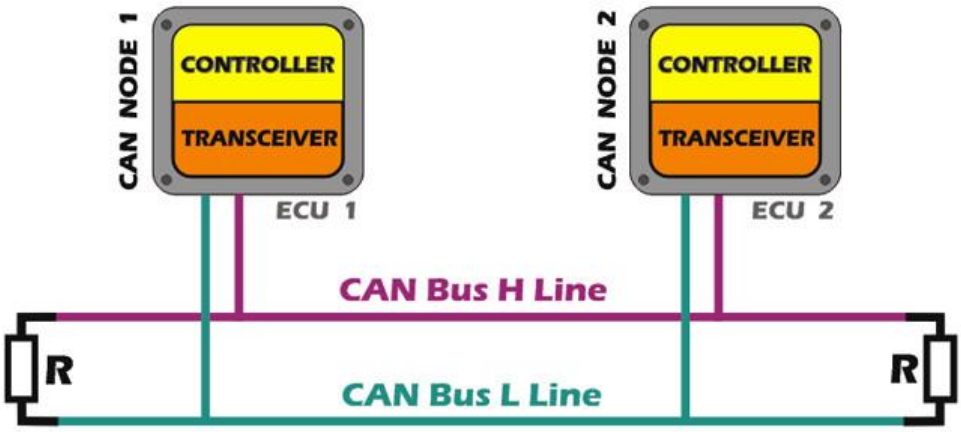

Ilustración 1 - Esquema del bus CAN [8]

Los campos del protocolo CAN están definidos por ISO 11898-1. Existen dos variantes, "Standard CAN", que utiliza 11 bits en el campo ID, y "Extended CAN" que utiliza 29 bits. Otros campos comunes a las dos variantes son 1 bit para indicar el comienzo del mensaje, 6 bits de control con 4 bits dedicados a la longitud de cargo útil, hasta 64 bits de carga útil, 16 bits de CRC, 2 bits de ACK y 7 bits para indicar el final de trama. [10]

Ilustración 2 - Esquema de los campos del protocolo CAN [8]

Todos los "Nodos" conectados a la red CAN reciben todas las tramas transmitidas por el bus CAN y dependiendo del valor ID, deciden aceptar o ignorar el mensaje. Si varios nodos quieren transmitir un mensaje al mismo tiempo, se da prioridad al mensaje con ID más bajo y el resto de los nodos deben esperar hasta que la red este libre para comenzar a transmitir.

Capítulo 3. ESTADO DE LA CUESTIÓN

En este capítulo se deben revisar qué trabajos o soluciones existen en el ámbito de mi proyecto. Ante la idea inicial de desarrollar un proyecto, siempre debemos realizarnos la pregunta: ¿hay algo similar en el mercado? ¿Hay algún trabajo de investigación que haya aportado los resultados que quiero alcanzar? Este apartado debe dar pie al siguiente, Justificación de mi proyecto. ¿Por qué hago el proyecto?

Actualmente, los riesgos de seguridad del protocolo CAN son extensivamente conocidos. Hay innumerables estudios que investigan los posibles "hacks" haciendo uso de dichos riesgos. En particular, el estudio por *Currie, Roderick, Basic Manipulation of a Modern Automovile Through CAN Bus Reverse Engineering* [12]**,** el cual sirvió de inspiración directa del sistema de ataque utilizado en este proyecto, demuestra la inseguridad inherente del protocolo CAN debida a la facilidad de penetración al sistema. Es también destacable que el ataque de "imitación de ID" es solo posible tras un exhaustivo proceso de prueba y error, del cual CAN no posee ningún mecanismo mediante el cual defenderse.

Tras conocerse los riesgos, naturalmente se investigaron mecanismos de seguridad que los remediaran, limitándose a las especificaciones de CAN. El equipo 16 de Senior Design de Rose-Hulman Institute of Technology, los progenitores de este proyecto implementaron el código de sus "nodos" un mecanismo que les hacía capaz de reconocer si los mensajes recibidos contenían su firma ID supuestamente única de cada dispositivo, lo que les indicaría que un intento "imitación de ID" está siendo realizado. Si un "nodo" detectará este ataque, mandaría un mensaje especial a la ECU, informándole del riesgo y permitiendo así que la ECU tomé las medidas necesarias.

Esta medida, aunque simple y efectiva para detectar un ataque, resulta insuficiente como medida preventiva, cualidad que se busca con la solución propuesta en este proyecto. La criptografía en CAN también ha sido explorada por diversos otros estudios. En particular, muchas de las técnicas criptográficas que se pusieron a prueba en este proyecto provinieron

de la lectura del excelente estudio realizado por *Bruton, Jennifer Ann, Securing CAN Bus Communication: An Analysis of Cryptographic Approaches* [13]. Pese a solo ser un extracto de la tesis, fue tremendamente útil para dar dirección al proyecto en su fase de investigación iniciales. Este estudio también discute las ventajas e inconvenientes de las distintas técnicas investigadas, dejando las al lector con los resultados y permitiéndole que saque sus propias conclusiones, reflejando lo abierta que es la cuestión de la criptografía abierta a CAN.

Otro estudio que merece ser mencionado es *Abu Al-Haija, Qasem, Al Tarayrah, Mashhoor, Al-Qadeeb, Hasan, Al-Lwaimi, Abdulmohsen A Tiny RSA Cryptosystem Based On Arduino Microcontroller Useful For Small Scale Networks* [14]. Pese a que la implementación final de este proyecto no utiliza criptografía asimétrica, y nunca se consideró utilizar derivados menos seguros de la misma, fue igualmente importante ya que influyo la dirección del proyecto, guiándolo hacia soluciones alternativas y menos costosas.

Como esfuerzo para rebajar los riesgos de ciberseguridad encontrados en el protocolo CAN, el proyecto EVITA [15] fue fundado por la Unión Europea en 2008 con el apoyo de compañías escrypt, Bosch, Honda, BMW, Continental, EURECOM, Fraunhofer, Fujitsu, infeneon, Mira, Telecom Paris Tech y Trialog entre otras. Notablemente, Bosch EVITA propone el uso de "Hardware Security Modules" (HSM), [16] unidades dedicadas a las costosas operaciones de la criptográficas, poniendo de ejemplo su propio producto "CycurHSM", que sirve de puente entre la ECU y el resto de nodo en la fase de acuerdo de llaves de la criptografía asimétrica. [17]

Este estudio sirvió como inspiración para buscar microcontroladores con componentes dedicados a operaciones criptográficas, para implementarlos de forma similar a un HSM sobre la solución del proyecto.

Capítulo 4. DEFINICIÓN DEL TRABAJO

1. JUSTIFICACIÓN

Este proyecto intenta buscar una solución criptográfica de mínimo efecto al rendimiento, ya que los componentes utilizados tienen capacidad de procesamiento limitada. Pese que la criptografía asimétrica es la solución ideal para este problema en términos de seguridad, es demasiado costosa para ser rentable. Por ello, la solución presentada en este proyecto es un compromiso, una operación ligera pero operacionalmente similar a la negociación de llaves a través de criptografía asimétrica. De este modo, también se consigue una solución segura y fácilmente adoptable por todo tipo de dispositivos, reduciendo el coste frente a otras posibles soluciones.

2. OBJETIVOS

Este proyecto tiene como objetivos:

1. Implementar un diseño de una red CAN encriptado que:
    - Hace imposible que un agente externo usurpe con éxito la identidad de cualquiera de los miembros de la red.
    - Protege la confidencialidad de los mensajes mandados a través del bus.
    - Minimiza el impacto al procesamiento de los mensajes, asegurando que la implementación es práctica.
2. Investigar la posibilidad de utilizar las técnicas criptográficas descritas previamente y sus papeles en el diseño final.
3. Utilizar el hardware del sistema análogo en la medida de lo posible, justificando los cambios realizados si el objetivo deseado es realmente inviable con los componentes que han sido impuestos.

3. METODOLOGÍA

Este proyecto fue otorgado un periodo límite de 20 semanas para su desarrollo. Durante las primeras 5 semanas fueron empleadas exclusivamente a investigación de técnicas criptográficas en microcontroladores 8 bit. Diversos algoritmos fueron testados, sus implementaciones encontradas en librerías open source. [6] Seguidamente, las semanas 6 a 9 fueron empleadas en diseñar e implementar una demostración practica de CAN-CHAP, una adaptación del algoritmo "Challenge Handshake Authentication Protocol" ("CHAP") [18] comúnmente encontrado en las "Virtual Private Networks" ("VPN") [19] al protocolo CAN.

Este modelo, junto a los descubrimientos hechos durante la fase de investigación, fue presentado en la semana 10 a Rolls Royce en su sede de Indianápolis. [20] Pese a que hubo un gran grado de satisfacción con el trabajo presentado, el alcance del proyecto fue expandido, de varias maneras. Primeramente, se intentó desarrollar un modelo más completo de la solución "CAN-CHAP", abarcando más de un "nodo" y una ECU, lo que se denominó "CAN-CHAP Multi-nodo". Sin embargo, debido a resultados muy pobres de la implementación final, se decidió abandonar este ángulo de investigación tras la semana 12.

Seguidamente, para el resto de las semanas se persiguió diseñar la denominada "Solución Hardware", que utilizaría microcontroladores con componentes dedicados para acaparar las costosas operaciones criptográficas. En un primer instante se decidió utilizar Mikroe CEC 1702 [21]. Una vez obtenido este componente y su programador J-TAG [22] su implementación probo muy dificultosa. Soporte dedicado online carente y problemas con la instalación de su bootloader, que no venía instalado de fábrica, paro ningún avance el proyecto durante las próximas 3 semanas.

Entrando en la semana 16, se decidió perseguir otros microcontroladores con capacidades similares, incluyendo el muy prometedor Espressif ESP32 [23]. Sin embargo, dado a la próxima fecha límite del proyecto, no pudo ser testado en gran profundidad y por ello, la solución hardware no pudo ser desarrollada.

4. PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA

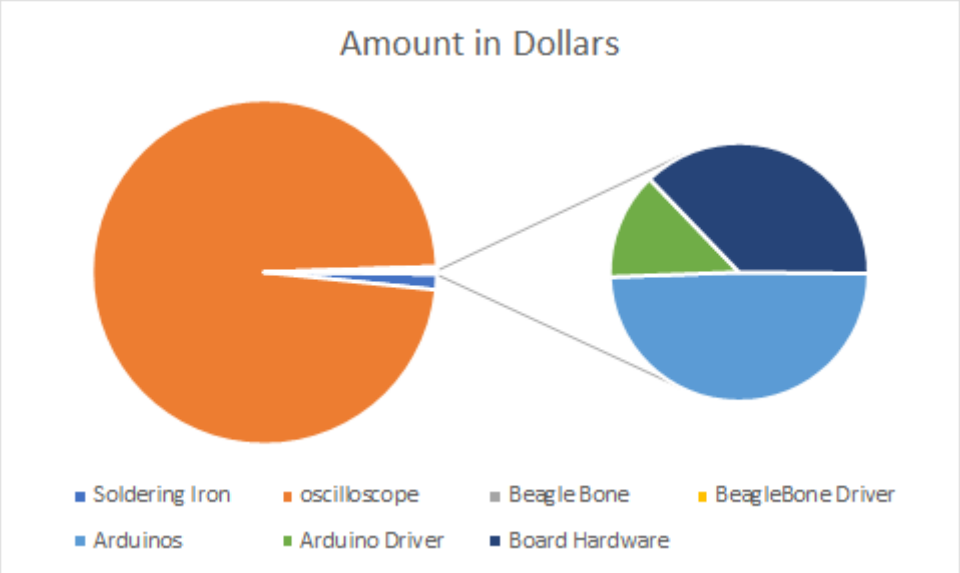

Ilustración 3 - Coste de los materiales usados durante el proyecto.

El gráfico circular que se muestra en la figura X muestra algunos datos llamativos. Específicamente, cómo el precio del osciloscopio utilizado terminó empequeñeciendo el precio de cualquier otro costo del proyecto. Los precios de algunos de los equipos que están disponibles para ser usados por los estudiantes a voluntad ciertamente no son apreciados a veces. El osciloscopio fue una herramienta crucial para el proyecto, ya que era una herramienta invaluable para depurar el bus CAN y verificar que había paquetes CAN limpios y adecuados que se enviaban a través de la red. Esto solo muestra cuánto cuesta el costo inicial del equipo para comenzar un nuevo proyecto.

Sin embargo, este hecho es poco sorprendente si se tiene en cuenta que los componentes obtenidos para la realización, BeagleBone Black Rev C [24] x2, Arduino Nano [11] x3, MCP2515 TJA1050 [25] x3, TI-SN65HVD231D [26] x2, cables y resistencias son todos componentes de bajo coste y en total no salieron por más de $300.

Capítulo 5. SISTEMA/MODELO DESARROLLADO

1. ANÁLISIS DEL SISTEMA

Primeramente, como parte elemental de la implementación, se necesita un modelo análogo al sistema bus CAN mediante el cual se pudiera hacer pruebas de riesgo de ciberseguridad en un espacio seguro y controlado. Este sistema, alias "sistema análogo", debería reflejar tanto en la arquitectura hardware como en la arquitectura software la estructura y funcionamiento de un sistema bus CAN genérico. Sobre este sistema se añadirá la capa criptográfica.

La carga útil del protocolo CAN es extremadamente limitada, de únicamente 64 bits. Si se desea mínimo efecto al rendimiento, es absolutamente crítico que el texto cifrado resultante de las técnicas criptográficas utilizadas se limite a la capacidad la carga útil, ya que, en caso contrario, los mensajes deberán ser mandados en más de una trama. Adicionalmente, el tiempo de procesamiento de las técnicas debe ser ligero para no ralentizar de forma notable el sistema. El límite que fue impuesto en este proyecto fue de 1 ms para el procesamiento de cualquiera de las técnicas utilizadas.

2. DISEÑO

La arquitectura del modelo análogo está compuesta por tres Arduino Nano [9] actuando como diferentes "Nodos" y un BeagleBone Black Rev C [24] actuando como "ECU". Los Arduino utilizan módulos CAN MCP2515 TJA1050 [25] como "transciever", a los que se conectan por SPI, y el BeagleBone Black un TI-SN65HVD231D que hace de driver para la regulación de la capa física [26]. Todos los "trancievers" están conectados con dos cables entrelazados a mano, actuando los papeles de "CAN-H" y "CAN-L".

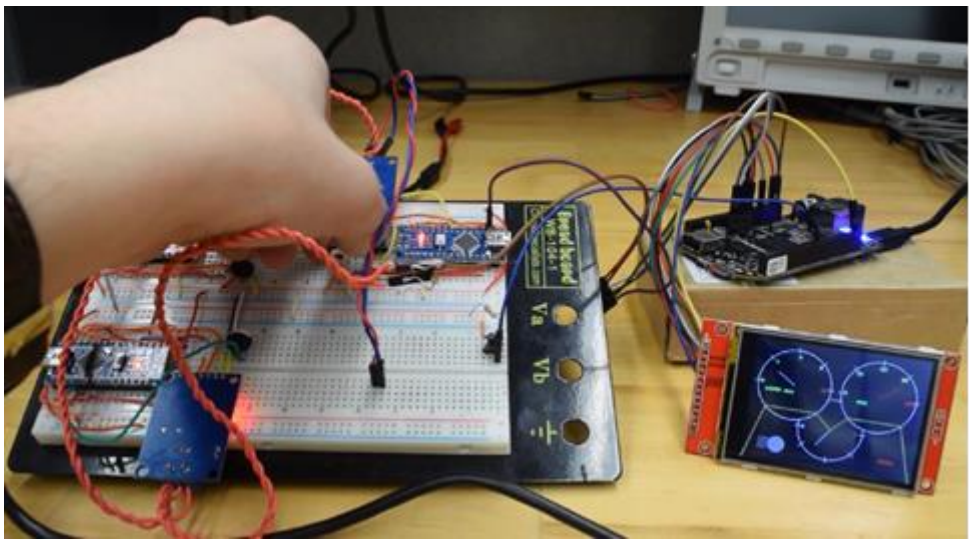

Ilustración 4 - Sistema análogo

Para la capa software se utilizaron varias librerías "Open Source". Para los "Arduino Nano", se utilizó CAN Bus Shield [27] que ofrece una interfaz de comunicación con su "transciever" CAN MCP2515 TJA1050. Para el BeagleBone Black, tras habilitar su capa CAN [28], se utilizó la librería socketCAN [29] [30] que permite enviar tramas CAN con la estructura socket.

Una vez construido el sistema análogo, se inició la fase de diseño del mecanismo de defensa criptográfico contra el ataque de ID "spoofing". Este consiste en unir un valor secreto, la llave, solamente conocido entre el emisor y receptor, a la firma del campo ID. Esta llave es utilizada para encriptar los mensajes entre estas dos entidades y un agente que desconozca su valor es incapaz por lo tanto de falsificar sus identidades. Adicionalmente, se consigue proteger la confidencialidad de los mensajes mandados, añadiendo gran dificultad al proceso de ingeniería inversa por el cual se puede obtener información adicional sobre el efecto sobre el motor de los distintos mensajes.

Sin embargo, todos los beneficios conseguidos por la implementación son invalidados si por cualquier motivo el valor de la llave secreta se hace público, ya que volveríamos al estado original de la cuestión. Por ello, no solo es importante que la técnica criptográfica utilizada para convertir el texto plano a texto cifrado sea segura, sino que además es absolutamente

crítico que el acuerdo de llave entre comunicantes sea realizado de manera confidencial. En este último punto es donde se ha hecho mayor énfasis y ha consumido los mayores recursos y tiempo de trabajo.

3. IMPLEMENTACIÓN

Teniendo en cuenta las limitaciones en memoria y computación impuestas por el sistema análogo, se optó por un diseño que no utilizara criptografía asimétrica [31] para la negociación de valor de llave entre "Nodo" y "ECU", dada a su alto coste computacional. Es por ello que tomando inspiración en el "Challange Handshake Authetication Protocol" (CHAP) utilizado en las "Virtual Private Networks" (VPN), se decidió diseñar una adaptación de este algoritmo orientado a CAN, al que se le dio el alias de "CAN-CHAP". Este algoritmo da una plataforma a un "Nodo" y a la "ECU" de verificar mutuamente sus identidades, a través de desafíos matemáticos y respuestas pre-calculadas. Una vez sus identidades son verificadas, las dos entidades pueden obtener el valor de la llave secreta combinando los valores de los mensajes compartidos y valores secretos que solo ellos dos conocen. [32]

Para poder crear CAN-CHAP, hubo que analizar CHAP y abstrayéndose de las particularidades técnicas, entender que hace a este algoritmo tan potente y cuáles son los elementos clave que su adopción debe mantener. El recurso el cual verifica las identidades del cliente y servidor en CHAP es conocimiento secreto previamente compartido, en otras palabras, algo que solo ellos dos saben. Este conocimiento no solo son datos, sino también una cierta operación matemática que, tras ser utilizada en conjunto con los datos públicos y privados, dan un resultado que ambas entidades pueden verificar. Yendo de lo abstracto a lo concreto, se puede observar que esta descripción se amolda muy bien al uso de "HMAC" [33] o "Keyed Hash", ya que tiene un valor secreto, la llave que en debería ser conocida únicamente por el cliente y el servidor, además de la operación matemática mutuamente verificable.

Sin embargo, utilizar "HMAC" es altamente costoso computacionalmente, además de que los resultados de dichas operaciones sobrepasan la carga útil de CAN, siendo de 256 bits o 512 bits habitualmente. Es por ello que se utilizó el "Keyed-Hash" "Sip Hash" [34], ya que fue diseñado para resultados de exactamente 64 bits. Este potente algoritmo, relativamente reciente y diseñado para como de medida de defensa contra ataques "D-DOS" [35], asegura que conociendo el texto plano y el resultante "hash", es imposible conocer el valor de la llave utilizada de ningún modo más eficiente que la fuerza bruta. Esto garantiza que el uso de la llave del "Sip Hash" como información secreta es seguro. Este algoritmo además es optimizado para aplicaciones de recursos computacionales limitados.

Una vez obtenida la verificación mutua, obtener la clave final es solamente un proceso de utilizar la información secreta y publica para crear la llave, con la seguridad de que los dos agentes lo harán de la misma manera. Sin embargo, para ofrecer aún más seguridad, se decidió utilizar, la técnica conocida como "key stretching" [36]. Esta simplemente consiste en hacer el valor de la llave un poco más largo y entrópico. El algoritmo utilizado para ello fue "blake2s" [37], por ningún otro motivo que su implementación en Arduino Nano era significativamente más rápida que la de los hashes más tradicionales de la familia "SHA" [38]. Esta llave sería utilizada por el algoritmo "ChaCha20" [39] para encriptar los mensajes transmitidos por los "nodos" y descifrados por la "ECU". El uso de este algoritmo no es solo debido tener la implementación más rápida entre los candidatos probados en el Arduino Nano, sino que además por la libertad que ofrece de limitar la longitud de su texto cifrado resultante, que se restringió a 64 bits.

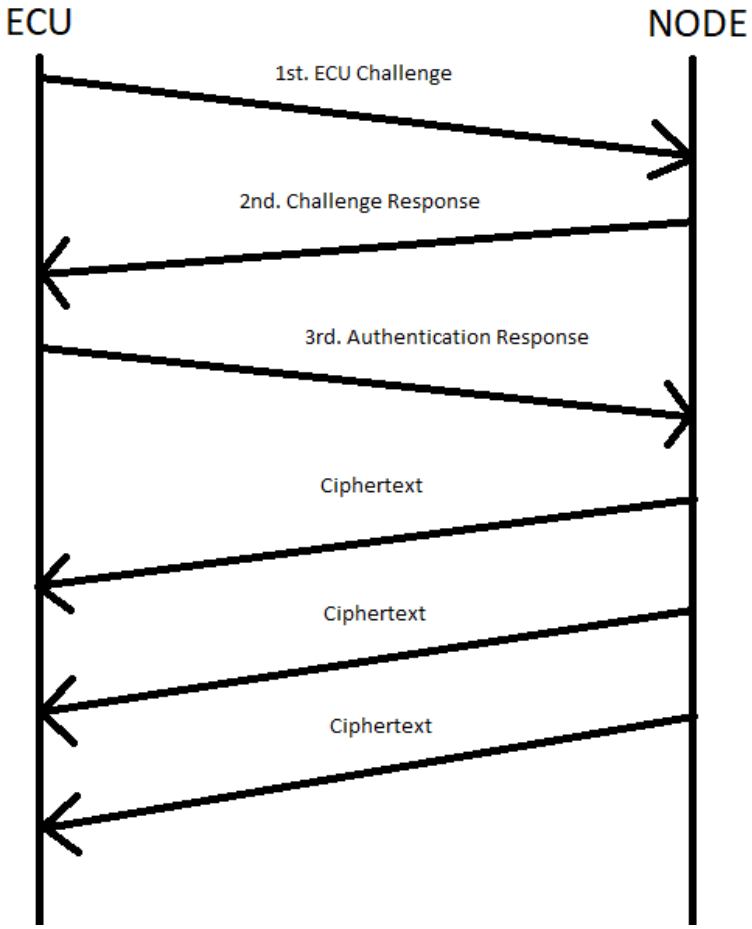

Ilustración 5 - Diagrama funcional de CAN-CHAP

La figura X muestra el modelo que se siguió para la implementación de CAN-CHAP. Esta sigue los siguientes pasos:

1. La "ECU" envía un número de 64 bit aleatorio a través de CAN BUS para ser recibido por los nodos de la red.
2. Un "Nodo" recibe un mensaje:
    a. Calcula un hash de 64 bits del número aleatorio, el cual será referido como s1.
    b. Calcula un hash de 64 bits utilizando s1 y su firma ID, el cual será referido como s2.
    c. Envía s1 a través del bus CAN para ser recibido por la ECU.

3. La "ECU" recibe una respuesta
    a. Calcula y verifica el valor s1. Si es correcto...
    b. Calcula s2 y lo envía a través del bus.
    c. Utiliza un hash del número aleatorio s1 y s2 para calcular la clave que utilizará para descifrar los mensajes recibidos con la ID del mensaje. La ECU está lista para recibir texto de cifrado procedente específicamente del ID de la respuesta recibida.
    d. Por otro lado, si el valor recibido no es s1, sigue a la escucha hasta haber una respuesta que tenga ese valor o hasta que termine su tiempo de escucha termine, en cuyo caso, la "ECU" vuelve al paso 1.
4. Un "Nodo" recibe un mensaje
    a. Verifica que el valor recibido sea s2. Si es correcto...
    b. Utiliza un hash del número aleatorio, s1 y s2 para calcular la nueva clave y comienza a enviar texto de cifrado
    c. Por otro lado, si el valor recibido no coincide con s2, el "nodo" vuelve a escuchar a otro número aleatorio procedente de la "ECU", en cuyo caso volviendo al paso 2.

Esta implementación también aprovecha una cualidad peculiar del protocolo CAN, la arbitración de ID, por la cual, en caso de que haya una colisión, la trama de ID más bajo tiene preferencia. [10] Es por ello por lo que el valor del campo ID utilizado por un "nodo" se asigna en función de cómo de crítica es la información que comunica a la "ECU". La implementación es por lo tanto coherente con este principio, ya que cuando la "ECU" mande su número arbitrario desafío, los "nodos" de ID más bajos serán los primeros en verificarse.

En el Anexo A se puede observar el código utilizado en el lado del BeagleBone Black y el del Arduino Uno para construir el modelo de CAN-CHAP de 1 nodo y una "ECU" que fue presentado en la sede de Rolls-Royce de Indianápolis el 19 de febrero de 2018. Este consta de una máquina de estados que sigue los pasos previamente mencionados. La “ECU” cambia su “ID” para reflejar el estado en el que se encuentra, principalmente, enviando un desafío o

verificando un “Nodo”. Los “Nodos” utilizan estos cambios de “ID” para cambiar su estado y de ese modo sincronizar su máquina de estados con la máquina de estados de la “ECU”.

Capítulo 6. ANÁLISIS DE RESULTADOS

Step	Arduino Nano	Beagle Bone Black
Random Number Generation (Normal Distribution Value Selection)	-	60 ± 2µs
Hashing (SipHash)	650 ± 20µs	63 ± 1µs
Encryption (ChaCha20)	60 ± 6µs / 900 ± 6µs *	66 ± 1µs
Key Extension (Blake2s)	700 ± 20µs	65 ± 2µs

Tabla 1 - Tiempos obtenidos.

La implementación de "CAN-CHAP", realizada sobre un modelo de 1 "nodo" y una "ECU" por simplicidad, resulto en un tiempo medio de 9900 ± 50µs para la mutua verificación de identidad y negociación del valor secreto. La tabla 1 detalla los valores obtenidos por los distintos componentes y fases de "CAN-CHAP". Como se puede observar en la TABLA 1, para el algoritmo “ChaCha20”, existen dos entradas para los tiempos obtenidos en Arduino Nano. Está discrepancia ocurre por funcionamiento único de este algoritmo, el cual se explicará a continuación.

“ChaCha20” es lo que es conocido como un “Stream Cipher” [40]. Estos algoritmos se basan en calcular una secuencia de bits, conocida como “bit-stream”, con la llave y vector de inicialización, que son datos aleatorios para hacer la llave más segura, y seguidamente, hacer una operación XOR con el texto plano sobre esa secuencia de bits para obtener el texto cifrado. Ya que dos operaciones XOR consecutivas se cancelan, si un receptor del texto cifrado ha calculado la misma secuencia de bits que el emisor, descifrar el mensaje es solo cuestión de realizar la operación XOR sobre el texto cifrado para obtener el texto plano.

En la librería open source de ChaCha20 utilizada para este proyecto el “bit-stream” es de 512 bits o 64 bytes. Ya que los mensajes a cifrar son de 64 bits o 8 bytes, solo se usan 8 bytes del “bit-stream” por cada mensaje cifrado, descartándolos una vez utilizados. Una vez utilizados los 64 bytes, se calcula un “bit-stream” nuevo, cambiando el valor de la llave, normalmente sumándole un “counter” de mensajes. En el Arduino Nano, la operación XOR llevaba 60 ± 6 µs. Una vez utilizado el “bit stream” completo, calcular el nuevo “bit-stream”

y realizar la operación XOR llevaba 900 ± 6µs, lo cual ocurría una vez cada 8 mensajes. Este efecto era inapreciable en el BeagleBone Black, debido a su mayor capacidad de procesamiento.

Por un lado, esto hace la "ChaCha20" el algoritmo criptográfico simétrico más rápido en promedio disponible en el Arduino Nano, por un margen muy significativo, ya que el resto de los algoritmos comprobados no bajaban de 700 µs. Sin embargo, tiempos de procesamiento irregulares pueden tener consecuencias muy negativas si el mensaje que se desea transmitir es de naturaleza crítica, ya que el sistema no tiene control sobre en qué punto del "bit stream" se enviará el mensaje. Es por ello por lo que algoritmos con tiempos más estables pudieran ser deseables.

El mayor factor de riesgo de "CAN-CHAP" es su vulnerabilidad a ataques de "replay" [41] Es por ello por lo que es esencial asegurar la entropía del método por el que se obtenga el número arbitrario que comienza la fase de negociación y del cual dependen el resto de los mensajes enviados entre "nodo" y "ECU". El método utilizado en la solución por el BeagleBone Black es un método matemático pseudo-arbitrario. Dado a que la implementación solo era una demonstración práctica, no se busco un método más seguro, como los generadores "TRNG" [42]. Sin embargo, en una versión comercial este sería un punto clave de trabajo, para prevenir los previamente mencionados "Replay-Attack".

Para la demonstración, no se mostró ningún mecanismo para detectar un ataque de "imitación de ID", pese a que dicho fue diseñado. Este consiste en sacrificar 1 byte de la carga útil y utilizarlo como un contador de mensajes, cuyo valor esta sincronizado entre "ECU" y el "Nodo". Si en algún momento existe un discrepancia entre el valor del contador de un mensaje recibido por la "ECU" y el valor guardado en memoria dentro de la "ECU" asociado al "ID" del mensaje, entonces la "ECU" tacha a esa "ID" de las "IDs" verificadas y ignorada todo mensaje procedente de la misma.

Pese a los puntos negativos o mejorables discutidos hasta el momento, se debe reconocer el mayor mérito conseguido con "CAN-CHAP", la rápida verificación de las identidades además de la negociación de las llaves, en menos de 10 ms. Teniendo en cuenta las

restricciones impuestas por el Hardware del Arduino Nano, si se hubiera utilizado la más estándar criptografía asimétrica, estaríamos observando tiempos de varios cientos de milisegundos. Si se tienen en cuenta las varias docenas de "Nodos" de una red real, utilizando "CAN-CHAP" se podría conseguir la verificación de todos los nodos y comienzo de la comunicación en ese mismo tiempo, mientras que una red con criptografía asimétrica limitada al hardware del proyecto se aproximaría más al orden de los segundos.

Por último, durante la fase de desarrollo de este proyecto se intento crear una demostración de un modelo multi-nodo con una "ECU", utilizando un "Thread" en el BeagleBone Black que escucha continuamente el socket abierto, separado de la máquina de estados, y una zona de memoria compartida con la máquina de estados donde está tendría acceso a las comunicaciones recibidas por los nodos. Sin embargo, un módelo temprano obtuvo tiempos de 30ms a 50ms para verificar y poner de acuerdo la llave con un único nodo. La inestabilidad de los tiempos más su incremento por factores probablemente relacionados con las limitaciones del BeagleBone Black desanimaron continuar más la investigación de este modelo.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

El mayor logro conseguido es un sistema que análogamente consigue lo mismo que una negociación de llaves con criptografía asimétrica. Este era uno de los grandes objetivos del proyecto y ha sido cubierto. Sin embargo, un modelo paralelo que negociará con criptografía asimétrica los valores de las llaves entre "ECU" y "Nodo", con componentes criptográficos dedicados, hubiera servido de una gran base para contrastar resultados y realmente legitimar "CAN-CHAP". Es más, se podría haber utilizado un modelo similar a "CAN-CHAP" con esos componentes, es decir, no haciendo uso de criptografía asimétrica, de este modo teniendo una verdadera perspectiva compresiva sobre la practicidad de "CAN-CHAP" o modelos similares sobre el que sacar conclusiones.

Como fue mencionado en el Capítulo 6. , la demostración actual posee algunos problemas. Para tiempos más estables para la criptografía asimétrica, otro candidato que fue considerado fue el algoritmo "Spritz" [43], que posee un funcionamiento similar a "ChaCha20", sin embargo, calcula un "bit-stream" nuevo por cada mensaje cifrado, por lo que su tiempo de cifrado es unos más lentos pero estables 800µs. Para asegurar la entropía del sistema, de modo que el número pseudoaleatorio utilizado como "desafío" es lo mas aleatorio posible, existen varias posibles soluciones.

Primeramente, se podría conectar al BeagleBone Black un generador de TRNG [42] el cual aseguraría la entropía del número desafío. Otra posibilidad es utilizar el texto cifrado transmitido por los "Nodos". Pese a la limitada carga útil del protocolo CAN en él contexto criptográfico, es raro que en aplicaciones industriales o automovilísticas reales se utilice el paquete entero. Es por ello que parte del mensaje cifrado "sobra". Sin embargo, si se utiliza la parte de la trama que "sobra", resultante de la operación criptográfica que es técnicamente "Pseudo-arbitraria", podría servir para alimentar la entropía del generador del número arbitrario.

Un objetivo que quedo sin satisfacer del proyecto fue crear una red CAN real encriptada. Dado los pobres resultados obtenidos por la demonstración multi-nodo, es más que posible que una "ECU" con más capacidad de procesamiento sea necesario. La mejor recomendación es algún microcontrolador con más de un núcleo de procesamiento, que aliviaría los requisitos necesarios para tener "threading" eficiente. Una vez creada esta red real, una mejor visión de la efectividad de CAN-CHAP podría ser realizada, especialmente si con esta red se realizaran experimentos para implementar criptografía asimétrica para la negociación de llaves, de modo que los resultados obtenidos podrían ser contrastados.

Por último, se terminará esta memoria como se empezó, implorando a los que tengan interés, ya sean representantes de la industria u otros investigadores a que busquen soluciones a los graves problemas de ciberseguridad del protocolo CAN. Es la esperanza de todos los miembros de este proyecto que el problema discutido y la solución otorgada sirva como inspiración para nuevas soluciones a este problema. Como se puedo observar en este capítulo, este proyecto no ha cubierto todas las posibilidades dentro de la solución que se deseo perseguir, y como se mencionó al principio de esta memoria, esta solución es solo una entre muchísimas. Mientras el protocolo CAN sea utilizado, es la misión de todos los ingenieros que lo utilizan asegurar la seguridad de su uso, por ello, se implora que estas posibilidades sean exploradas en futuros proyectos.

Capítulo 8. BIBLIOGRAFÍA

[1] Ruta 401, *"Conector OBD del coche: ¿para qué sirve y dónde encontrarlo?"* Web Blog, Source: https://blog.reparacion-vehiculos.es/conector-obd-coche

[2] Rouse, Margaret. Rosencrance, Linda. Shea, Sharon. Wigmore, Ivy. *"internet of Things (IoT)",* tech blog, Source:https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT

[3] Gutiérrez, Pedro. *"¿Qué es y como surge la criptografía?: un repaso por su historia"* En Genbeta Dev, Source: https://www.genbetadev.com/seguridad-informatica/que-es-y-como-surge-la-criptografia-un-repaso-por-su-historia

[4] Itzcoatl Salazar Monroy, Juan Alberto. *"Criptografía y criptoanálisis: la dialéctica de la seguridad".* Universidad Nacional Autónoma de México. Source: https://revista.seguridad.unam.mx/numero-17/criptografí-y-criptoanálisis-la-dialéctica-de-la-seguridad

[5] Diaz, José Luis. *"Tipos de criptografía: criptografía simétrica, criptografía asimétrica y criptografía hibrida."*. 11 de abril, 2016. Web Blog. Source: https://es.paperblog.com/tipos-de-criptografia-simetrica-asimetrica-y-hibrida-3713787/

[6] github, *Arduino Cryptography Library*, Web Repository, Source: https://rweather.github.io/arduinolibs/crypto.html

[7] Donohue, Brian. *"¿Qué Es Un Hash Y Cómo Funciona?".* 10 de Abril, 2010. KasperSkyLab daily. Souce: https://latam.kaspersky.com/blog/que-es-un-hash-y-como-funciona/2806/

[8] Mucevski, Kiril. *Automotive CAN Bus System Explained*, linkedIn article, Source: https://www.linkedin.com/pulse/automotive-can-bus-system-explained-kiril-mucevski/

[9] Pro Car Mechanics, *How an Engine Control Unit Affects Performance*. Web Blog, Source: http://procarmechanics.com/how-an-engine-control-unit-affects-performance/

[10] Dahikar, Shrikant, *Basic of CAN (Controller Area Network) for vehicle ECU communication*, linkedIn article, Source: https://www.linkedin.com/pulse/basic-can-controller-area-network-vehicle-ecu-shrikant-dahikar/

[11] ARDUINO, *ARDUINO NANO*, Web Store, Source: https://store.arduino.cc/usa/arduino-nano

[12] Currie, Roderick. *"Basic Manipulation of a Modern Automovile Through CAN Bus Reverse Engineering"*. SANS Institute InfoSec Reading Room. Source: https://www.sans.org/reading-room/whitepapers/threats/hacking-bus-basic-manipulation-modern-automobile-through-bus-reverse-engineering-37825

[13] Bruton, Jennifer Ann. *"Securing CAN Bus Communication: An Analysis of Cryptographic Approaches*". Extracto de tesis doctoral, National University of Ireland, Galway , Agosto de 2014. Source: http://www.osna-solutions.com/wp-content/uploads/Securing-CAN-Bus-Communication-Thesis-Extract.pdf

[14] Abu Al-Haija, Qasem, Al Tarayrah, Mashhoor, Al-Qadeeb, Hasan, Al-Lwaimi, Abdulmohsen. *"A Tiny RSA Cryptosystem Based On Arduino Microcontroller Useful For Small Scale Networks*". Scientific Survey, Source: https://www.sciencedirect.com/science/article/pii/S1877050914009466

[15] EVITA, Web Archive, Source: http://friedewald.website/category/research/evita

[16] Gemalto, *"Hardware Security Modules (HSMs)"*, Online Web Store, Source: https://safenet.gemalto.com/data-encryption/hardware-security-modules-hsms/

[17] Escrypt, *"Design Workshop: Automotive HSMs"* Presentación Powepoint. Source: http://auto2015.bosch.com.cn/ebrochures2015/energizing_powertrain/etas/150216_schulung_cycurhsm.pdf

[18] Rouse, Margaret. *CHAP (Challenge Handshake Authetication Protocol)*. Web Wiki, Source: https://searchsecurity.techtarget.com/definition/CHAP-Challenge-Handshake-Authentication-Protocol

[19] A. Llorca, Águeda. *"VPN: ¿qué es y para qué sirve?"*. nobbot, web blog, Source: https://www.nobbot.com/tecnologia/mi-conexion/vpn-¿que-es-y-para-que-sirve/

[20] Rolls Royce, Indianapolis. Source: https://www.rolls-royce.com/country-sites/northamerica/rolls-royce-in-the-us/rolls-royce-indianapolis.aspx

[21] Mikroe, *"CEC1702 Clicker"*. Page Store. Source: https://www.mikroe.com/clicker-cec1702

[22] Mikroe. *"mikroProg for CEC"*. Page Store. Source: https://www.mikroe.com/mikroprog-cec

[23] Espressif, *ESP32 Overview*, Web Market, Source: https://www.espressif.com/en/products/hardware/esp32/overview

[24] adafuit, *BeagleBone Black Rev C*, Web Store, Source: https://www.adafruit.com/product/1876

[25] dx, *Módulo de bus CAN MCP2515 TJA1050 Módulo receptor SPI*, Web Store, Source: http://www.dx.com/es/p/mcp2515-can-bus-module-tja1050-receiver-spi-module-blue-434988#.Wy57-FUzaDI

[26] mangoboard, *CAN Transciever Module*, Web Store, Source: http://www.mangoboard.com/main/view.aspidx=424&pageNo=1&cate1=9&cate2=165&cate3=

[27] github, *CAN Bus Shield*, Web Reposotary, Source: https://github.com/Seeed-Studio/CAN_BUS_Shield

[28] Webdemeyer, Thomas. *BeagleBone CANbus & Python*, Blog, Source: http://www.thomas-wedemeyer.de/beaglebone-canbus-python.html

[29] github, *CAN utils*, Web Repository, Source: https://github.com/linux-can/can-utils

[30] linklayer, SocketCan, Web Wiki, Source: https://wiki.linklayer.com/index.php/SocketCAN

[31] De Luz, Sergio. *Criptografía: Algoritmos de cifrado de clave asimétrica*. Web Blog, Source: https://www.redeszone.net/2010/11/16/criptografia-algoritmos-de-cifrado-de-clave-asimetrica/

[32] Rouse, Margaret. *CHAP (Challenge Handshake Authentication Protocol)*. Web Wiki, Source: https://searchsecurity.techtarget.com/definition/CHAP-Challenge-Handshake-Authentication-Protocol

[33] Carl Villanueva, John. "What Is HMAC And How Does It Secure File Transfers?". Web Blog, Source : https://www.jscape.com/blog/what-is-hmac-and-how-does-it-secure-file-transfers

[34] Aumasson, Jean-Philippe, J. Bernstein, Daniel. *SipHash: a fast short-input PRF*. Web Blog, Source: https://131002.net/siphash/

[35] Cloudfare, *"What is a DDoS Attack?"*, Service Webpage, Source: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/

[36] Spacey, John. *What is Key Streching?*, Web Blog, Source: https://simplicable.com/new/key-stretching-definition

[37] Aumasson, Jean-Philippe, Neves, Samuel, Wilcox-O´Hearn, Zooko, Winnerlein, Christian. *Blake2 – fast secure hashing*, Web Blog, Source: https://blake2.net/

[38] Landman, Nathan. Williams, Chistopher. Ross, *Elli. Secure Hash Algorithms*. Web Blog, Source: https://brilliant.org/wiki/secure-hashing-algorithms/

[39] Y. Nir, Check Point, A.Langley, ChaCha20 and Poly1305 for IETF Protocols, Scienctific Surver, Source: https://tools.ietf.org/html/rfc7539

[40] Carl Villanueva, John, *"An Introduction to Stream Ciphers and Block Ciphers"*. Jscape Web Blog, Source: https://www.jscape.com/blog/stream-cipher-vs-block-cipher

[41] KK, *Replay-Attack*, Web Wiki, Source: http://www.crypto-it.net/eng/attacks/replay.html

[42] A Jones, David. *"True random number generators for a more secure IoT"*. Tech Design Forums, Source: http://www.techdesignforums.com/practice/technique/true-random-number-generators-for-more-secure-systems/

[43] Adjal, Abderraouf, *"ArduinoSpritzCipher"*. GitHub. Source: https://github.com/abderraouf-adjal/ArduinoSpritzCipher

ANEXO A

```
/*
*Author:Daniel Moreno
*Single NODE/ECU demo
*/

//Libraries
#include "Crypto/ChaCha.h"
#include "Crypto/ChaCha.cpp"
#include "Crypto/Cipher.h"
#include "Crypto/Cipher.cpp"
#include "Crypto/Crypto.h"
#include "Crypto/Crypto.cpp"
#include "Hash/BLAKE2s/BLAKE2s.h"
#include "Hash/BLAKE2s/BLAKE2s.cpp"
#include "Hash/BLAKE2s/Hash.h"
#include "Hash/BLAKE2s/Hash.cpp"
#include "Hash/SipHashC/siphash.c"
#include <cstdlib>
#include <random>
#include <cmath>
#include <chrono>
#include <iostream>
#include <unistd.h>
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <linux/can.h>
#include <linux/can/raw.h>

//Constants
#define ID_RECON 0x02
#define ID_CHALLENGE 0x03
#define ID_WINNER 0x04
#define ID_BEGIN 0x05
#define TIMEOUT_CHALLENGER_micros 10000
#define SOCKET_NAME "can1"


//Global variables
std::random_device rd;
std::mt19937_64 e2(rd());
std::uniform_int_distribution<uint64_t>
       dist(std::llround(0), std::llround(std::pow(2,63)));
uint8_t rndm[8];

typedef void *(*StateFunc)();
StateFunc statefunc;
uint8_t const sharedKey[16] = "Senior Projectz";
uint8_t s1[8];
uint8_t s2[8];
ChaCha cipher;//Single node only
```

```
BLAKE2s blake;
int canSocket;
struct can_frame rxFrame;

//states
void *sendChallenge();
void *waitChallenger();
void *verifyChallenger();
void *sendAck();
void *recieveCiphertext();

//timing utils
auto startProtocol = std::chrono::high_resolution_clock::now();
auto beginSeq = std::chrono::high_resolution_clock::now();
auto endAuth = std::chrono::high_resolution_clock::now();
auto endKeyAgree = std::chrono::high_resolution_clock::now();
long int rndmTime;

//utils
bool open_port(const char *port){ //returns the socket
    struct ifreq ifr;
    struct sockaddr_can addr;

    /* open socket */
    canSocket = socket(PF_CAN, SOCK_RAW, CAN_RAW);
    if(canSocket < 0)
    {
        return 0;
    }
    /* convert interface string port into interface index */
    addr.can_family = AF_CAN;
    strcpy(ifr.ifr_name, port);

    if (ioctl(canSocket, SIOCGIFINDEX, &ifr) < 0)
    {
        return 0;
    }

    /* setup address for bind */
    addr.can_ifindex = ifr.ifr_ifindex;

    fcntl(canSocket, F_SETFL, O_NONBLOCK);

    if (bind(canSocket, (struct sockaddr *)&addr, sizeof(addr)) < 0)
    {
        return 0;
    }

    return 1;

}

bool writeOnCAN(unsigned long id, uint8_t* data, size_t dlc){
    struct can_frame frame;
    int nbytes;


    /* first fill, then send the CAN frame */
    frame.can_id = id;
    frame.can_dlc = dlc;
```

```
    memcpy(frame.data, data, frame.can_dlc);

    nbytes = write(canSocket, &frame, sizeof(frame));

    if(nbytes < 0)
    {
       std::cout << "Error sending frame";
       return 0;
    }

    return 1;
}

bool read_port(){
  int nbytes = read(canSocket, &rxFrame, sizeof(struct can_frame));

  if (nbytes < 0) {
    //std::cout << "nothing to read" << std::endl;
    return 0;
  }

  /* paranoid check ... */
  if (nbytes < sizeof(struct can_frame)){
    //std::cout << "what I read, was wrong" << std::endl;
    return 0;

  }

  return 1;
}

void getRndm(uint8_t* data, size_t len) {
    uint64_t random = dist(e2);
    memcpy(data, &random, len);
}

void printHex(uint8_t Data) {
  if (Data < 0x10)
    printf("0");
  printf("%x ", Data);
}

void printArray(uint8_t Source[], uint8_t LEN) {
  for (int Idx = 0; Idx < LEN; Idx++)
    printHex(Source[Idx]);
  printf("\n");
}

bool array_cmp(uint8_t *a, uint8_t *b, uint8_t len_a, uint8_t len_b){
     uint8_t n;

     if (len_a != len_b)
      return 0;

     for (n=0;n<len_a;n++)
      if (a[n]!=b[n])
        return 0;

     return 1;
}
```

```
//define states
void *sendChallenge(){
  //Time utils
  startProtocol = std::chrono::high_resolution_clock::now();
  getRndm(rndm,sizeof(rndm));
  //Time utils
  auto endRndm = std::chrono::high_resolution_clock::now();
  rndmTime = std::chrono::duration_cast<std::chrono::microseconds>(endRndm-
startProtocol).count();

  writeOnCAN(ID_CHALLENGE,rndm,sizeof(rndm));
  beginSeq = std::chrono::high_resolution_clock::now();

  return (void *) waitChallenger;
}

void *waitChallenger(){
  bool arrived = false;
  bool timeout = false;
  unsigned long int timeoutmicros = TIMEOUT_CHALLENGER_micros;
  auto start = std::chrono::high_resolution_clock::now();


  while(!(arrived||timeout)){
    auto end = std::chrono::high_resolution_clock::now();
    arrived = read_port();
    timeout = (std::chrono::duration_cast<std::chrono::microseconds>(end-
start).count() >= TIMEOUT_CHALLENGER_micros);
  }


  if(arrived){
    /*debug stuff
    printf("id=0x%x dlc = %d data= ",rxFrame.can_id, rxFrame.can_dlc);
    printArray(rxFrame.data,rxFrame.can_dlc);
    */
    return (void *)verifyChallenger;
  }

  //debug stuff
  std::cout << "timeout!" << std::endl;

  return (void *)sendChallenge; //just in case something weird happens
}

void *verifyChallenger(){
  uint8_t s1Data[12];
  uint8_t s2Data[12];
  uint32_t IDbuf = ID_WINNER;

  //Load s1Data
  memcpy(s1Data,rndm,sizeof(rndm));
  memcpy(s1Data+sizeof(rndm), &rxFrame.can_id, sizeof(rxFrame.can_id));

  //calculate s1
  siphash(s1Data, sizeof(s1Data), sharedKey, s1, sizeof(s1));

  //verify challenger
  if(!array_cmp(s1,rxFrame.data,sizeof(s1),sizeof(rxFrame.data)))
    return (void *)sendChallenge;
```

```
  //Load s2Data
  memcpy(s2Data,s1,sizeof(s1));
  memcpy(s2Data+sizeof(s1), &IDbuf, sizeof(IDbuf));

  //calculate s2
  siphash(s2Data, sizeof(s2Data), sharedKey, s2, sizeof(s2));

  /*debug stuff
  std::cout << "S1 DATA: ";
  printArray(s1Data,sizeof(s1Data));
  std::cout << "S1: ";
  printArray(s1,sizeof(s1));
  std::cout << "S2 DATA: ";
  printArray(s2Data,sizeof(s2Data));
  std::cout << "S2: ";
  printArray(s2,sizeof(s2));
  */
  return (void*) sendAck;

}

void *sendAck(){ //Now is a bit barebones, in multinode he will check if list
                 //is empty, then decide if send another challenge or start
                 //with deciphering
  cipher = ChaCha(20);
  uint8_t hash[32];

  //Calculate new key
  blake.reset();
  blake.update(sharedKey,sizeof(sharedKey));
  blake.update(s1,sizeof(s1));
  blake.update(s2,sizeof(s2));
  blake.finalize(hash, sizeof(hash));

  //initialize cipher
  cipher.setKey(hash,sizeof(hash));
  cipher.setIV(s1,sizeof(s1));
  cipher.setCounter(s2,sizeof(s2));

  writeOnCAN(ID_WINNER,s2,sizeof(s2));
  endAuth = std::chrono::high_resolution_clock::now();

  return (void *) recieveCiphertext;
}

void *recieveCiphertext(){
  //send 8 ciphered messages and go back to waitChallenge
  uint8_t plaintext[8];

  for(uint8_t n = 0; n < 8; n++)
  {
    while(!read_port());
    cipher.decrypt(plaintext,rxFrame.data,sizeof(rxFrame.data));

    //debug stuff
    if(n==0)
      endKeyAgree = std::chrono::high_resolution_clock::now();
    std::cout << "ciphertext: ";
    printArray(rxFrame.data,sizeof(rxFrame.data));
    std::cout << "Plaintext: ";
    printArray(plaintext,sizeof(plaintext));
```

```
  }

  //Just to do the sequence
  cipher.clear();

  std::cout << "Coming up with a random took " << rndmTime << " microseconds" <<
std::endl;
  std::cout << "Authentication took " <<
std::chrono::duration_cast<std::chrono::microseconds>(endAuth-beginSeq).count()
<< " microseconds" << std::endl;
  std::cout << "The first message received in " <<
std::chrono::duration_cast<std::chrono::microseconds>(endKeyAgree-
endAuth).count() << " microseconds after authentication" << std::endl;
  std::cout << "The protocol took " <<
std::chrono::duration_cast<std::chrono::microseconds>(endKeyAgree-
startProtocol).count() << " microseconds to initialize encrypted communication"
<< std::endl;
  std::cout << std::endl;
  std::cout << std::endl;
  sleep(4);

  return (void *) sendChallenge;
}

//main
int main(){

  //setup
  char socketName[] = SOCKET_NAME;

  if(!open_port(socketName)){
    std::cout << "Error opening socket, closing program" << std::endl;
    return 1;
  }

  //state machine
  statefunc = sendChallenge; //Beginning state

  while(1){statefunc = (StateFunc)(*statefunc)();}

  return 0;
}
```

Código del BeagleBone Black para CAN-CHAP single node.

```
/*
* Author: Daniel Moreno
* Demo of CAN-CHAP, single node network
*/

//Libraries
#include <ChaCha.h>
#include <Crypto.h>
#include <BLAKE2s.h>
#include <SPI.h>
#include <mcp_can.h>
#include "SipHashC/siphash.c"

//Constants
#define ID_NODE 0x102 //Different for every node
#define ID_RECON 0x02
```

```
#define ID_CHALLENGE 0x03
#define ID_WINNER 0x04
#define ID_BEGIN 0x05
#define CS 10
#define CAN0_INT 2 //Pin for input
#define TIMEOUT_ACK_micros 10000

//Global variables
typedef void *(*StateFunc)();
StateFunc statefunc;
uint8_t const sharedKey[16] = "Senior Projectz";
uint8_t rxBuf[8];
uint8_t rxLen;
uint8_t s1[8];
uint8_t s2[8];
ChaCha cipher;
BLAKE2s blake;
MCP_CAN CAN0(CS);
bool RECV_INT_FLAG = false;

//States
void *waitChallenge();
void *challengeAccepted();
void *waitAck();
void *whatAmI();
void *winner();
void *sendCipher(); //For now, no checking if cipher is out of sync

//Set flag function
void MCP2515_ISR()
{
    RECV_INT_FLAG = true;
}

//time utils
long int beginning;
long int auth;
long int ciphercreate;
long int daend;

//utils
void PrintArray (uint8_t Source[], uint8_t LEN)
{
  for (int Idx = 0; Idx < LEN; Idx++)
    SerialPrintHex (Source[Idx]);
  Serial.println();
}

void SerialPrintHex (uint8_t Data )
{
  if (Data < 0x10) Serial.print('0');
  Serial.print(Data, HEX);
  Serial.print(" ");
}

bool sendmsg(uint8_t* buf, int len) {
  // send data:  ID = ID_NODE, Standard CAN Frame, Data length = 8 bytes, 'buf' =
array of data bytes to send
  byte sndStat = CAN0.sendMsgBuf(ID_NODE, 0, len, (const byte*) buf);
  if(sndStat == CAN_OK){
    return true;
```

```
  }
  return false;
}

boolean array_cmp(uint8_t *a, uint8_t *b, uint8_t len_a, uint8_t len_b){
     uint8_t n;

     if (len_a != len_b)
      return false;

     for (n=0;n<len_a;n++)
      if (a[n]!=b[n])
        return false;

     return true;
}

//Define States
void *waitChallenge(){
  bool challenge = 0;
  long unsigned int rxId = 0;

  while(!challenge){
    if(!digitalRead(CAN0_INT)) {
        CAN0.readMsgBufID(&rxId, &rxLen, rxBuf);

        if(rxId == ID_CHALLENGE && rxLen == 8){
          challenge = 1;
          beginning = micros();
        }
    }
  }

  return (void *) challengeAccepted; //next state
}

void *challengeAccepted(){
   uint8_t s1Data[12];
   uint8_t s2Data[12];
   uint32_t IDbuf = ID_NODE;

   //Load s1Data
   memcpy(s1Data,rxBuf,sizeof(rxBuf));
   memcpy(s1Data+sizeof(rxBuf), &IDbuf, sizeof(IDbuf));

   //Calculate S1
   siphash(s1Data, sizeof(s1Data), sharedKey, s1, sizeof(s1));

   //Load s2Data
   IDbuf = ID_WINNER;
   memcpy(s2Data,s1,sizeof(s1));
   memcpy(s2Data+sizeof(s1), &IDbuf, sizeof(IDbuf));

   //Calculate s2
   siphash(s2Data, sizeof(s2Data), sharedKey, s2, sizeof(s2));

   /*DebugStuff
   Serial.print("S1 DATA: ");
   PrintArray(s1Data,sizeof(s1Data));
   Serial.print("S1: ");
   PrintArray(s1,sizeof(s1));
```

```
   Serial.print("S2 DATA: ");
   PrintArray(s2Data,sizeof(s2Data));
   Serial.print("S2: ");
   PrintArray(s2,sizeof(s2));
   //DebugStuff*/

   //send s1
   while(!sendmsg(s1, sizeof(s1)));

   return (void *) waitAck;
}

void *waitAck(){
  long start = micros();
  long end = 0;
  long unsigned int rxId = 0;

  do{
    end = micros();
    if(!digitalRead(CAN0_INT)) {
        CAN0.readMsgBufID(&rxId, &rxLen, rxBuf);

        if(rxId == ID_WINNER){
          return (void *) whatAmI;
        }

        if(rxId == ID_CHALLENGE){
          return (void *) challengeAccepted;
        }
    }

  }while(end-start <= TIMEOUT_ACK_micros);
}

void *whatAmI(){

   if(array_cmp(s2,rxBuf, sizeof(s2),sizeof(rxBuf)))
    return (void *) winner;

   return (void *) waitChallenge;
}

void *winner(){
  //time utils
  auth = micros();

  //In multinode, wait for ID begin, for now, send as soon as Blake2s is done and
cipher set
  cipher = ChaCha(20);
  uint8_t hash[32];

  //Calculate new key
  blake.reset();
  blake.update(sharedKey,sizeof(sharedKey));
  blake.update(s1,sizeof(s1));
  blake.update(s2,sizeof(s2));
  blake.finalize(hash, sizeof(hash));

  //initialize cipher
  cipher.setKey(hash,sizeof(hash));
  cipher.setIV(s1,sizeof(s1));
```

```
  cipher.setCounter(s2,sizeof(s2));

  //time utils
  ciphercreate = micros();

  return (void *) sendCipher;
}

void *sendCipher(){
  //send 8 ciphered messages and go back to waitChallenge
  uint8_t plaintext[]  = {0x0D, 0x0A, 0x01, 0x01, 0x01, 0x0E, 0x07, 0x00};
  uint8_t ciphertext[8];

  for(uint8_t n = 0; n < 8; n++)
  {
     cipher.encrypt(ciphertext,plaintext,sizeof(plaintext));
     while(!sendmsg(ciphertext,sizeof(ciphertext)));
     if(n == 0)
      daend = micros();
     delay(10);
  }

  //Just to do the sequence
  cipher.clear();

  Serial.println();
  Serial.println();
  Serial.print("Mutual authentication took: ");
  Serial.print(auth - beginning);
  Serial.print(" microseconds after recieving challenge");
  Serial.println(" microseconds");
  Serial.print("Cipher creation took ");
  Serial.print(ciphercreate - auth);
  Serial.println(" microseconds");
  Serial.print("First cipher sent after authentication took: ");
  Serial.print(daend - ciphercreate);
  Serial.println(" microseconds");

  Serial.print("The protocol took ");
  Serial.print(daend - beginning);
  Serial.println(" microseconds to initialize communication with ECU after
recieving challenge");

  return (void *) waitChallenge;
}

// Normal arduino stuff
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.println("Ready.");

  // Initialize MCP2515 running at 16MHz with a baudrate of 500kb/s and the masks
and filters disabled.
  while(!(CAN_OK == CAN0.begin(CAN_500KBPS))) {
    if(CAN0.begin(CAN_500KBPS) == CAN_OK)
      Serial.println("MCP2515 Initialized Successfully!");
    else Serial.println("Error Initializing MCP2515...");
  }

  pinMode(CAN0_INT, INPUT);
```

```
// Configuring pin for INT input

  attachInterrupt(CAN0_INT, MCP2515_ISR, FALLING); // start interrupt

  statefunc = waitChallenge; //Beginning state

}

void loop() {
  // put your main code here, to run repeatedly:
  statefunc = (StateFunc)(*statefunc)();
}
```

Código del Arduino Uno para CAN-CHAP single node