Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS

Software-based implementation of Secure AES Frame Encryption (SAFE) for CAN FD in multicore environment

---

TRABAJO RECEPCIONAL que para obtener el **GRADO** de
ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: **ALBERTO CONTRERAS ESTRADA E IVAN EDGAR REYES OLVERA**

Asesor **ABRAHAM OTERO TEZMOL**

Tlaquepaque, Jalisco. agosto de 2022.

Software-based implementation of Secure AES Frame Encryption (SAFE) for CAN FD in multicore environment

Reyes Olvera Ivan Edgar
Guadalajara, Mexico
edgar.reyes@iteso.mx

Contreras Estrada Alberto
Guadalajara, Mexico
a.contreras@iteso.mx

Abraham Tezmol Otero
Guadalajara, Mexico
ATEZMOL@iteso.mx

***Abstract*— The SAFE kernel software implementation was born as an inexpensive alternative from the original implementation which uses a field programmable gate array (FPGA). This paper proposes to replace the FPGA with a microprocessor, which will drastically reduce the cost of the implementation. The new implementation includes the MIMXRT1170 EVK that integrates two microprocessors, a Cortex M4 running at 400 MHz and a Cortex M7 running at 1 GHz. After the software implementation, the multicore environment met the required time deadlines and achieved the desire performance, keeping the CAN FD bus occupation at 100% and the encryption and decryption time less than 50 microseconds, which represents 25% of the microprocessor overhead. In future works the implementation could be adjusted to one microprocessor, such as the cortex M7 due to it is powerful capability to process data which also reduces the cost and the complexity of the implementation.**

Keywords*— *Cybersecurity, AES algorithm, Multicore, Cortex M7, Cortex M4, CAN FD, MIMX RT1170-EVK

I. Introduction

Electronic control units (ECUs), such as anti-lock braking system (ABS), air bags and the main computer are embedded systems that control the vehicle's functionalities and communicate with each other through a communication protocol. The communication between all ECUs was usually implemented over the control area network (CAN) protocol but the most recent secure requirements include the CAN flexible data rate (CAN FD) protocol instead by allowing a higher data transfer rate and message size than the CAN protocol.

The advanced encryption standard (AES) algorithm is a well-known data cipher algorithm adopted by the National Institute of Standards and Technology (NIST) as a Federal Information Processing Standard (FIPS) that is used in several cybersecurity applications to ensure confidentiality of shared information. AES needs 3 inputs: a key which can be 128, 192 or 256 bits, an initialization vector (IV) of 16 bytes, and the message. Then the output is a ciphered message which can only be deciphered with the key.

The SAFE kernel is a security software module created to add encryption functionalities using the CAN FD protocol, and its main functionalities include adding a different 128-bit private key to every sent message, cipher the message, read the 128-bit private key from every received message and decipher the message. Any of these 2 processes plus the required time to communicate both cores must be performed by the SAFE kernel in 50 micro-seconds or less.

The existing SAFE kernel implementation is based on a microprocessor Cortex M7 and a Field-Programmable Gate Array (FPGA) which is more expensive than a microprocessor and faster for data processing. The purpose of this paper is to reduce the cost of the current implementation by using a multicore approach while meeting the 50 micro-second deadline.

The document was divided in three sections, the first contain the introduction to the paper, the second section describe the methodology used to achieve the development and implementation for the SAFE kernel, and the third section that shows the results obtained.

II. Methodology

In this paper we propose to replace the FPGA used in a security module for SAFE Kernel implementation with a microprocessor to reduce the implementation cost and meet time deadlines.

Both cores start synchronously and run a non-preemptive scheduler. A communication mechanism called messaging unit (MU) was implemented to avoid memory access collisions between the cores.

The described time deadline originates from the CAN bus constraints, assuming that the CAN protocol is capable of sending uninterrupted messages, each message requires at least 200 microseconds (minimal message) on the bus to be sending so this time is assumed to be 100% utilization of the bus. The acceptable level of microprocessor overhead was set at 25% or less (50 microseconds) while the use of the CAN FD bus remains at 100% to optimize the implementation of the SAFE kernel and allow the M4 cortex to execute other tasks.

A. Non-preemptive scheduler

A non-preemptive scheduler was implemented in both cores using the system clock (systick) to trigger a hardware timer interruption every 500 microseconds as base time. This is the available time in every thread to perform the assigned tasks, so if this time is exceeded, the schedule will have an erratic behavior. The scheduler provides 4 execution threads (1, 10, 50 and 100 milliseconds) and 2 more independent threads (2-A and 2-B milliseconds), which have the following interconnections (Table II.1):

Table 1. Sheduler interconnection threads.

PRIMARY THREAD	DEPENDENT THREAD
1MS THREAD	100MS THREAD
2MS A THREAD	50MS THREAD
2MS B THREAD	10MS THREAD

The scheduler initialization, functionalities and interaction with the hardware and the user are described in the following sequence diagram:

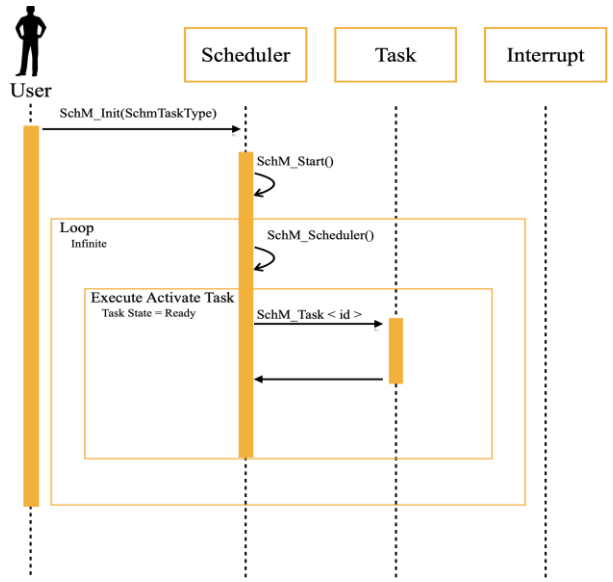

Figure 1. Scheduler architecture.

B. *Messaging Unit*

The message unit is a mechanism to communicate both cores by using hardware flags. This software module was provided by NXP Semiconductors as a part of the software development kit (SDK). Application programming interfaces (APIs) were created, and the module was modified to write and read data from shared memory, implementing the needed logic to avoid collisions while accessing the shared memory. The MU has 4 channels for general purpose as shown in Figure 2 and channel 0 (CH0) was used to communicate both cores.

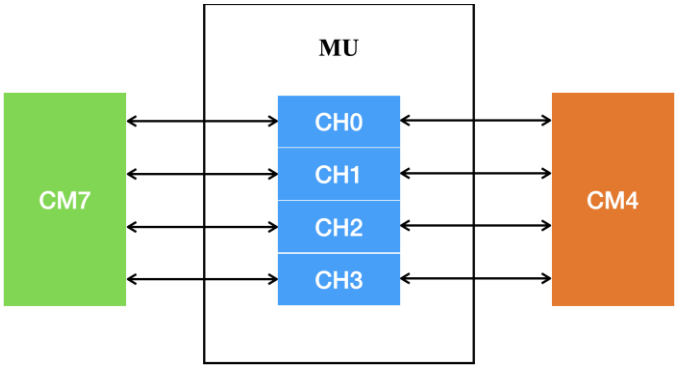

Figure 2. Message unit structure.

C. *Cortex M4*

The system clock signal in this core and its peripherals was configured to maximum speed, which is 400 MHz, to achieve the best performance and improve the task execution times for the non-preemptive scheduler and the CAN FD communication, reducing the microprocessor overhead and making possible to the microprocessor unit (MCU) to keep the CAN FD bus usage at 100%.

This core manages the CAN FD communication protocol, which was provided by NXP Semiconductors with the SDK. The CAN FD peripheral was configured as follows:

- Data rate: 2000000.
- Baud rate: 1000000.
- Message ID format: Standard (11 bits length).
- Message TX ID: 0xAAA
- Message RX ID: 0xAAA
- DLC: 16

The following APIs were created:

- **CANFD_init**: Perform the initial configuration for CAN FD peripheral, set the multiplexed pines and clock signal.
- **CANFD_trigger_standard_tx**: Trigger the transmission of the standard ID buffer through the CAN FD bus.
- **CANFD_trigger_standard_rx**: Trigger the listening mode and reception of a standard ID buffer through the CAN FD bus.
- **CANFD_trigger_extended_TX**: Trigger the transmission of the extended ID buffer through the CAN FD bus.
- **CANFD_trigger_extended_RX**: Trigger the listening mode and reception of an extended ID buffer through the CAN FD bus.
- **CANFD_set_Baudrate**: Set baud rate speed into CAN FD configuration structure.
- **CANFD_set_Datarate**: Set data rate speed into CAN FD configuration structure.
- **CANFD_set _RX_ID**: Set ID to reception buffer into CAN F configuration structure.
- **CANFD_set_TX_ID**: Set ID to transmission buffer into CAN FD configuration structure.
- **CANFD_set_message_format**: Set CAN FD message ID format.
- **CANFD_print_TX_buffer**: Print CAN FD transmitted payload buffer.
- **CANFD_print_RX_buffer**: Print CAN FD received payload buffer.
- **CANFD_print_config**: Print CAN FD current configuration.

D. *Cortex M7*

The system clock signal in this core and its peripherals was configured to maximum speed, which is 1 GHz, to achieve the best performance and improve the task execution times for the

non-preemptive scheduler and the SAFE kernel reducing the microprocessor overhead and keeping it below 25%.

The implemented functions of the SAFE kernel are as follows:

- Initialization: AES-192 hardware is initialized; a random number of 4 bytes is generated which will be the public key, and the lookup table is loaded to RAM memory. From this internal process, the private key will be acquired for each message.
- Send a message: to send a message, a random number is calculated and used as an index to extract a private key from the lookup table; with this key and the public key the message is ciphered, then the public key, private key and the ciphered message are joined together in a buffer that is sent through the MU to the Cortex M4 to be later sent by CAN FD.
- Receive a message: when receiving a message through the messaging unit, the private key must be extracted, set in the AES algorithm to decipher the message, and return a pointer to the buffer where the deciphered message is stored.

The AES algorithm is performed by the Cortex M7 as well; this software package was provided by NXP Semiconductors with SDK. The CAN FD peripheral was configured as follows:

- 128 bits key length.
- MBEDTLS version: 2.27.0.
- Cryptographic accelerator and assurance module (CAAM) hardware accelerated.

The following APIs were created:

- **AES_init**: Perform the initial configuration for AES peripheral, set the multiplexed pines and clock signal.
- **AES_print_features**: Print the current AES configuration.
- **AES_cipher**: Cipher a message.
- **AES_Decipher**: Decipher a message.
- **AES_set_key_length**: Update or set the key length.
- **AES_set_cipher_key**: Update or set the key to cipher a message.
- **AES_set_DEcipher_key**: Update or set the key to decipher a message.

The physical implementation was made as shown in Figure 3. Each MIMXRT1170 was connected to a different computer in this way each computer has an emulated serial terminal that connects each card to the computer.

The CAN bus (CAN high and CAN low) was connected between the boards using the black and red braided cable shown at the bottom of the figure, this cable uses 2 resistors of 120 ohms at each end as required in the protocol to function properly.

The logic analyzer is connected to the general-purpose pins used to measure the time periods required for the project by setting the logical level to 1 when the process to be measure start and setting the logical level to 0 when the process ends. The grounds of the boards are independent, but this does not affect the measurement since we are only interested in the time that the logic 1 persists and not its magnitude.

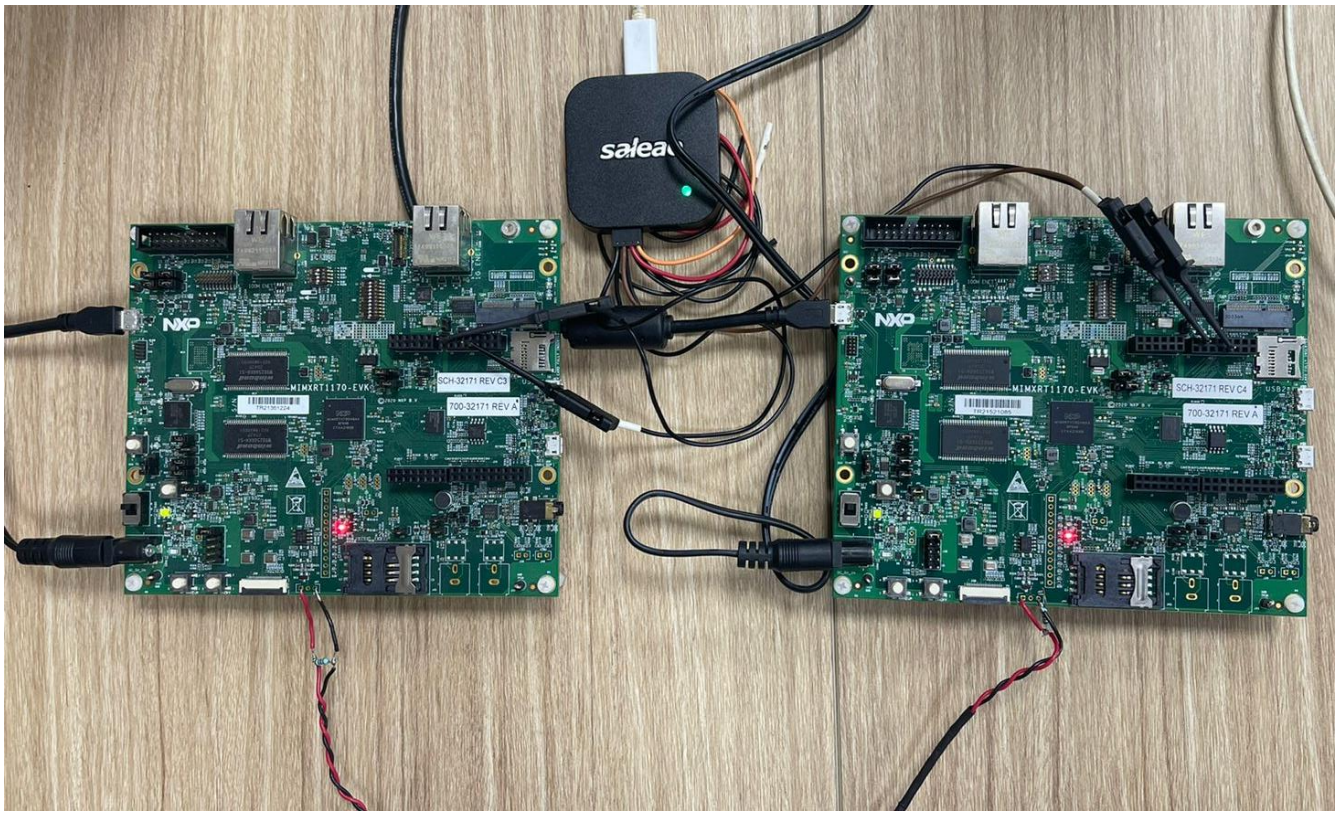

Figure 3. Physical implementation of the SAFE kernal and digital osciloscope to analize the CAN FD.

III. RESULTS

The test frame applied to the software implementation developed in this paper was as follows:

- Node A: the cortex M7 was configured to cipher 10 messages and use the MU to transfer the messages to the cortex M4; the cortex M4 was configured to send the ciphered messages through CAN FD.
- Node B: the cortex M4 was configured to receive ciphered messages through CAN FD and use the MU to transfer the messages to the cortex M7 where the SAFE kernel will read, decipher and print the messages on the screen.

A logical analyzer was used to measure the spent time for the following process:

1. For the SAFE kernel to cipher every message.
2. For the MU to transfer every message from cortex M7 to cortex M4.
3. For the CAN FD to transfer every message through the physical bus.
4. For the MU to transfer every message from cortex M4 to cortex M7.
5. For the SAFE kernel to decipher every message.
6. For the scheduler to run on time the 4 execution threads.

The success criteria included the achievement of the following requirements:

- The test frame should not have any warnings or errors in the compilation.
- The 10 messages must be successfully transmitted and received by CAN FD.
- The 10 messages printed on the screen must be identical to the 10 original messages.
- The time spent on message encryption and transmission over the MU between cores must be equal to or less than 50 microseconds for all messages.
- The time used by the CAN FD protocol to transmit messages over the physical bus should be around 280 micro-seconds.
- The messages must be transmitted over CAN FD one after the other, which means that the bus is 100% occupied.

The time was averaged over the 10 sent messages and the results are shown in (Table 2).

Table 2. Comparison of software vs FPGA implementations.

Criteria software implementation	**Software implementation (micro-seconds)**	**FPGA implementation (micro-seconds)**
Transmission process		
Time spent by the SAFE kernel to cipher	9.06	0.41
Time spent by the MU to transfer message from M7 to M4	12.86	183
SUM	21.92	183.41
Reception process		
Time spent by SAFE kernel to decipher.	12.60	0.41
Time spent by the MU to transfer message from M4 to M7.	12.86	183
SUM	25.46	183.41
Time spent by the CAN FD protocol to transmit a message through the physical bus	293	291

IV. CONCLUSIONS

The results obtained in this paper show that software implementation of the SAFE kernel is a feasible option. Using the multicore approach, the cost of using the SAFE kernel message encryption as a security mechanism was reduced by 90%, making an available solution for production volumes in the automotive industry. Finally, the time deadlines were met and the overhead of the microprocessor hosting the SAFE kernel was calculated at 23.69%, which is below than the 25% limit. The CAN FD bus remained at 100% occupancy and the test frame was successfully executed and tested. The actual implementation could be improved by using just one microprocessor and thus reduce the complexity and the cost. This approach should be analyzed to guarantee a microprocessor overhead below 25%, which is an allowed percentage in the automotive industry.

V. REFERENCES

Lugo-Meneses, C.A. & Peralta-Reynoso, D. (2019). Secure AES Frame Encryption for CAN FD. Trabajo de obtención de grado, Especialidad en Sistemas Embebidos. Tlaquepaque, Jalisco: ITESO. https://rei.iteso.mx/handle/11117/5973. [Accessed September 15, 2021].

Abraham Tézmol, Francisco Martínez "Scheduler specification – Embedded System" DESI ITESO A.C. (September 2, 2018), [Accessed September 20, 2021].

NXP B.V. 2020-2021 "i.MX RT1170 Processor Reference Manual" Date of Release (5/2021), document identifier - IMXRT1170RM. https://www.nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/i-mx-rt1170-evaluation-kit:MIMXRT1170-EVK. [Accessed September 21, 2021].

Eiza, M. Hashem, and Q. Ni. “Driving with Sharks: Rethinking Connected Vehicles with Vehicle Cybersecurity.” IEEE Vehicular Technology Magazine 12, no. 2 (June 2017): 45–51. [Online], available: IEEE, https://doi.org/10.1109/MVT.2017.2669348. [Accessed February 3, 2022].

[Hartwich, Florian, and Robert Bosch GmbH. “CAN with Flexible Data-Rate,” 2012, 9. [Online], available: https://www.can-cia.org/fileadmin/resources/documents/proceedings/2012_hartwich.pdf [Accessed March 8, 2022].

Microchip, “Atmel-44003-32-Bit-Cortex-M7-Microcontroller-SAM-V71Q-SAM-V71N-SAM-V71J_Datasheet.Pdf.”, February 2015 [Revised October 2016]. [Online], available: Microchip, http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-44003-32-bit-Cortex-M7-Microcontroller-SAM-V71Q-SAM-V71N-SAM-V71J_Datasheet.pdf.

D. Schneider. “Jeep Hacking 101.” IEEE Spectrum: Technology, Engineering, and Science News, August 6, 2015. [Online], available: IEEE, https://spectrum.ieee.org/cars-that-think/transportation/systems/jeep-hacking-101. [Accessed March 9, 2022].

D. S. Moeller, R. W. Pashby, C. Joe Holmes. “Method for OTA Updating vehicle Electronic Control Unit”, U.S. patent 2016/0364232, Dec. 15, 2016. [Online], Available: Google Patents, https://patentimages.storage.googleapis.com/40/08/8d/3686dcff8f03da/US20160364232A1.pdf. [Accessed March 10, 2022].

J. Daemen, and V. Rijmen, The Rijndael Block Cipher, n.d., 47. NIST. [Online], available: NIST, https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf.