Security-Aware Design of Time-Critical Automotive Cyber-Physical Systems

Vipin Kumar Kukkala, Thomas Bradley, and Sudeep Pasricha

1 Introduction

Today's vehicles are complex cyber-physical systems with tens of interconnected Electronic Control Units (ECUs) that control various subsystems in the vehicle. The introduction of Advanced Driver Assistance Systems (ADAS) in vehicles to support the goals of autonomy has resulted in an increase in the number of ECUs, which in turn has increased the complexity of the in-vehicle network that connects the ECUs. Moreover, state-of-the-art ADAS relies on information from various external systems using advanced communication protocols such as vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) [1]. These advances increased the complexity of automotive systems, which introduced several other challenges related to reliability [2–6], real-time performance [7–10] and security [11–15] of automotive systems. In this chapter, we focus on improving security in automotive systems. The increased connectivity of today's vehicles has made them highly vulnerable to various sophisticated cyber-attacks. Therefore, ensuring the security of automotive systems is a crucial concern and will become further crucial as connected and autonomous vehicles (CAVs) become more ubiquitous.

The most commonly seen cyber-attacks on vehicles include masquerade, replay, and denial of service (DoS) attacks [16]. In a *masquerade* attack, the attacker

V. K. Kukkala (✉)
NVIDIA, Santa Clara, CA, USA
e-mail: vipin.kukkala@colostate.edu

T. Bradley
Department of Systems Engineering, Colorado State University, Fort Collins, CO, USA
e-mail: thomas.bradley@colostate.edu

S. Pasricha
Colorado State University, Fort Collins, CO, USA
e-mail: sudeep.pasricha@colostate.edu

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

V. K. Kukkala, S. Pasricha (eds.), *Machine Learning and Optimization Techniques for Automotive Cyber-Physical Systems*, https://doi.org/10.1007/978-3-031-28016-0_4

pretends to be an existing ECU in the system. In a *replay* attack, the attacker eavesdrops on the in-vehicle network, captures valid messages transmitted by other ECUs, and sends them on the network in the future. In a *DoS* attack, the attacker ECU floods the in-vehicle network with random messages, thereby preventing the normal operation of valid ECUs. Most of these attacks require access to the in-vehicle network, which can be acquired either physically (e.g., using on-board diagnostics (OBD-II)) or remotely (e.g., using LTE or Bluetooth). Various real-world approaches to gaining access to the in-vehicle network and taking control of the vehicle by sending malicious messages are discussed in detail in [17–20].

Traditional in-vehicle network protocols, such as controller area network (CAN), FlexRay, etc., fail to address key security concerns such as confidentiality, authentication, and authorization as they do not have any inherent security features. Thus, additional security mechanisms (e.g., encryption-decryption) must be implemented in ECUs to prevent unauthorized access to the in-vehicle network. The two most widely used encryption techniques are - *symmetric key encryption* and *asymmetric key encryption*. The former uses the same key for encryption and decryption operations, while the latter uses a public-private key pair that has a strong mathematical relation. Both mechanisms incur computational overhead on the ECUs, which may catastrophically delay the execution of real-time automotive tasks and message transfers, e.g., a delay in the messages from impact sensors to airbag deployment systems could lead to severe injuries for vehicle occupants. Thus, it is highly crucial to carefully introduce security mechanisms in the vehicles.

The individual ECU utilizations of a FlexRay-based automotive system consisting of four ECUs running 12 different hard real-time automotive applications (each with multiple tasks) is illustrated in Fig. 1. Each ECU has a real-time utilization due to the execution of real-time automotive tasks (*RT Util*) and a security utilization because of the execution of security operations (*Sec Util*). The numbers on top of each bar show the number of applications that miss their deadlines when executed on the corresponding ECU. Along the x-axis, the *no security mechanism* case has no security mechanism implemented (hence it only has the real-time utilization), while in the *unoptimized security mechanism* case, all the ECUs employ AES-256 for encryption and decryption of messages. In the latter case, it can be seen that the total utilization for ECUs 3 and 4 (sum of real-time and security task utilizations) exceeds 100% (represented by the red dotted line) because of the overhead of security-specific encryption/decryption task executions, resulting in missed deadlines for four applications. Lastly, the *optimized security mechanism* case represents our goal in this work, to integrate all required security mechanisms while keeping utilization of all ECUs below 100%, without any deadline violations.

In this chapter, we present a novel security framework called *SEDAN*, which was first introduced in [11]. *SEDAN* is a lightweight (minimal overhead on the ECUs) security framework that aims to maximize the overall security of the automotive system without violating real-time deadline constraints and per-message security constraints. Moreover, the *SEDAN* framework employs symmetric key cryptography as it is less computationally intense compared to the asymmetric key cryptography to enhance the security of the vehicle. Our novel contributions in *SEDAN* are:

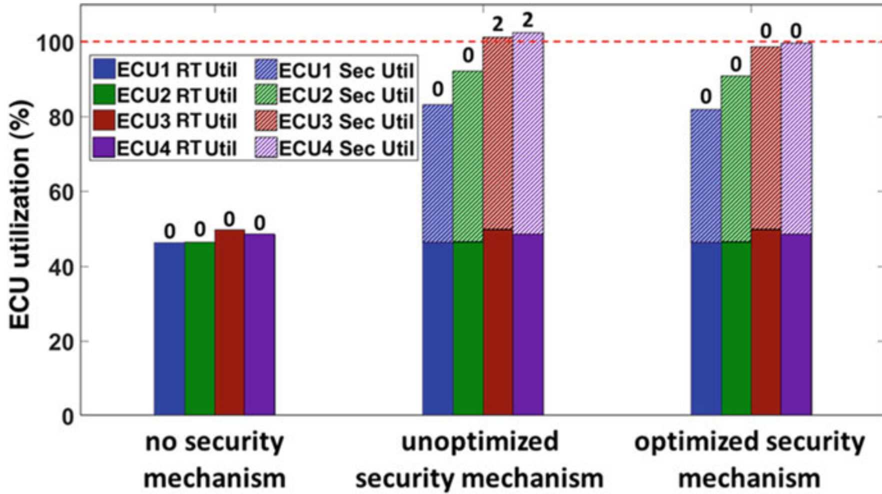


Fig. 1 Motivation for a lightweight (low overhead) vehicular security framework. The number on top of each bar indicates the number of missed real-time application deadlines

- We introduced a novel quantitative methodology to derive the security requirements for various messages in an automotive system based on ISO 26262 standard and formulated a new metric to quantify the overall security of a system;
- We devised a heuristic-based key management technique to provide adequate security for various message types and ensure that the utilization of all ECUs is below 100%;
- We developed an approach for the joint exploration and synthesis of message schedules and security characteristics in TDMA-based automotive systems and also proposed a technique to efficiently map tasks to ECUs while meeting real-time message deadlines and ECU utilization goals;
- We extracted network traffic and ECU execution data from a real-world vehicle (2016 Chevrolet Camaro) and compared *SEDAN* with [21], the best-known prior work in the area, to demonstrate the effectiveness and scalability of *SEDAN*.

2 Related Work

Security in automotive systems was not a primary concern until recently. The first full vehicle hack in 2010 [17] highlighted the need for concrete security measures in automotive systems. In [17], the researchers had physical access to the vehicle and were able to control various systems in the vehicle by injecting custom messages into the CAN bus. Moreover, they reverse-engineered a subset of the ECUs and were able to update the firmware on those ECUs by sending custom CAN messages. Later in [18], they were able to perform the same attacks remotely. In [19], the

researchers hacked the radio in a 2014 Jeep Cherokee and were able to control the vehicle remotely. They used the telematics system in the radio to send remote messages to the vehicle, which were injected into the CAN bus to take control of various vehicular subsystems. In [20], the authors recently developed a Trojan app that was executed on a smartphone connected to the vehicle infotainment system via Bluetooth. They used this app to send custom CAN messages into the in-vehicle network. All these attacks have raised serious concerns about security in automotive systems.

Since the traditional in-vehicle network protocols do not provide any security features, it is hard to prevent unauthorized access to the in-vehicle network. However, one of the popular solutions in the literature to prevent unauthorized access is authenticating the sender ECU using message authentication codes (MACs). Several works, such as [22–28], advocate the use of MACs to improve security in automotive systems. In [23], a mixed integer linear programming (MILP) formulation was proposed to minimize the overhead for MAC computation and end-to-end application latency in a CAN-based system. Moreover, the authors in [23] use the same MAC for a group of ECUs. In [24], the authors extended [23] to minimize the security risks associated with grouping different ECUs. An authentication protocol called LCAP was presented in [26] to encrypt messages that utilized hash functions to generate hashed MACs to authenticate ECUs. In [27], an RC4 encryption-based authentication was implemented to improve security in CAN-based systems. Another lightweight authentication scheme based on PRESENT [29] was introduced in [28] and evaluated on FPGAs. However, cryptanalysts have demonstrated successful attacks on both RC4 and PRESENT. In [30], a technique based on obfuscating CAN message identifiers (IDs) was presented to protect a fleet of vehicles. *However, all the above-mentioned techniques are designed for event-triggered protocols (such as CAN) and do not apply to more scalable and sophisticated time-triggered protocols.*

A lightweight authentication technique is proposed in [22] that uses cipher-based MACs that are generated using the ECU local time stamp and a secret key. However, this technique requires strong synchronization between the ECUs, and any uncertainty can result in a full system failure. In [31], a device-level technique is presented, which uses an enhanced network interface (NI) to authenticate ECUs in the system by using hardware-based security modules (HSMs). In [25], FPGAs are employed as co-processors for ECUs to handle all security tasks that are implemented based on the TESLA [32] protocol. However, the techniques in [25, 31] require additional compute resources and many modifications to the existing automotive systems, which is not very practical and cost-efficient. In [33], the authors proposed a virtual local network (VLAN) based solution for improving security in Ethernet-based automotive systems. They introduced an integer linear programming (ILP) model to minimize message routing times and authenticate the messages by making multiple message transmissions on different routes. However, this technique results in inefficient bandwidth utilization and poor scalability. A co-design framework is introduced in [34] to improve message response times while meeting security concerns. However, only a small subset of messages

are considered for encryption to guarantee control performance, which makes it impractical for safety-critical automotive systems and also exposes the system to various vulnerabilities.

A time delayed release of keys approach (adapted from the TESLA protocol [32]) is proposed in [21], in conjunction with simulated annealing based heuristic to minimize the end-to-end latency of messages by co-optimizing task allocation and message scheduling. This is one of the very few holistic frameworks that integrate the concept of security with real-time system design from the beginning of the system design phase. This work is extended in [35] by including V2V communication, using dedicated short-range communication (DSRC). In [36], a lightweight authentication technique for vehicles called LASAN is proposed, which uses the Kerberos protocol. The authors extended this work in [37] by presenting a comprehensive analysis of LASAN and compared with the TESLA [32] protocol. Though the LASAN technique demonstrated superior performance over other works, it has stringent requirements for a trusted centralized ECU, which creates a single point of failure. A security mechanism using different authentication methods was proposed for real-time systems in [38]. A group-based security service model is presented in [39] that tries to maximize the combined security of the system. However, as the model does not consider the timing constraints, it cannot be implemented in time-critical automotive systems.

An intrusion detection system (IDS) based on principal component analysis (PCA) is proposed in [40]. An IDS that detects the presence of an attacker by monitoring the increased transmission rates of the messages is proposed in [41]. In [42], the usage of reactive runtime enforcers called safety guards is proposed to detect the discrepancies between the input data from sensors and the output of the controllers. In [43], a challenge-response authentication approach was proposed to detect the presence of attackers. However, this technique requires prior and proprietary information about the sensors to function correctly.

All the above-mentioned prior works for securing time-triggered systems have various limitations: *(i)* they do not consider the utilization overhead on ECUs and latency overhead on messages due to the implemented security mechanisms, which results in over-optimistic results; *(ii)* they utilize only one key size for all messages, which that does not account for the heterogeneous security goals in real-time systems; *(iii)* they do not consider precedence constraints between tasks and messages, and; *(iv)* they consider homogenous single core ECUs which do not accurately represent today’s vehicles. In this chapter, we present the *SEDAN* framework that addresses these limitations of the prior works. Moreover, *SEDAN* improves the security in vehicles with time-triggered networks while satisfying all security, utilization, and message timing constraints. We demonstrate it for the FlexRay protocol, but it can be easily extended to other time-triggered protocols, e.g., TTEthernet.

3 Problem Definition

3.1 System and Application Model

In this subsection, we present the automotive system model that was considered in *SEDAN*, where multiple ECUs execute different time-critical applications and are connected using a FlexRay-based network, as shown in Fig. 2. Each ECU has of two major components: a host processor (HP) and a communication controller (CC). The HP primarily runs the automotive and security applications, whereas a CC acts as an interface between the HP and the in-vehicle network (in this case, FlexRay bus) and is responsible for packing message data into frames, sending and receiving messages, and filtering out unwanted messages. Moreover, *SEDAN* considers heterogeneous HPs with different numbers of cores, which aligns with the state-of-the-art. It is important to note that the heterogeneity in this work is limited to varying the number of homogeneous cores per HP (i.e., multicore parallelism).

Each automotive application consists of dependent and independent tasks that are mapped to different ECUs and executed in the corresponding HPs. If two dependent tasks are mapped to the same ECU, they exchange information using shared memory. Otherwise, the tasks communicate with each other by exchanging messages over the FlexRay bus. A message contains one or multiple signals that are generated as a result of task execution on the ECU. The Signals are packed into messages by the HP and are sent to the CC to transmit as FlexRay frames on the bus. Automotive applications can be categorized into one of two types: *(i)* time-triggered (periodic) or *(ii)* event-triggered (aperiodic). Most safety-critical applications, e.g., collision avoidance, lane keep assist, anti-lock braking, etc., are

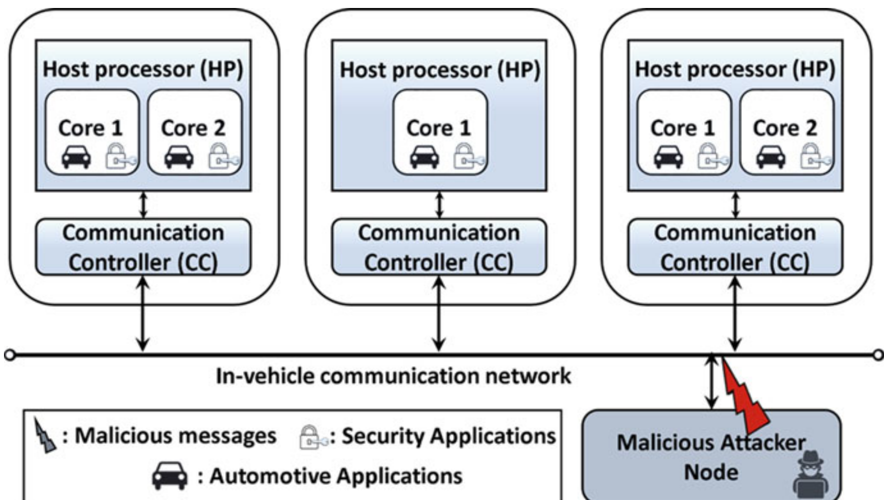


Fig. 2 Overview of the automotive system model used in *SEDAN*

time-triggered and generate time-triggered messages. Event-triggered messages are generated by maintenance and diagnostic applications. Moreover, much like real-time applications across other domains, the execution characteristics of time-critical automotive applications are known at design time. In this work, we focus on time-triggered applications as they significantly impact system performance and vehicle safety. Additionally, time-triggered messages generated by these applications have strict timing and deadline constraints. Thus, it is vital to optimize the security of the time-triggered messages while ensuring that no real-time deadline constraints are violated. In this work, we adapt various state-of-the-art standards, namely, Advanced Encryption Standard (AES) with key sizes 128,192 and 256 bits and evaluate Rivet-Shamir-Adleman (RSA) with key sizes 512, 1024, 2048, and 4096 bits, and Elliptic Curve Cryptography (ECC) with key sizes 256 and 384 bits to improve system security.

3.2 FlexRay Communication Protocol

FlexRay is an in-vehicle network protocol designed to support high-speed real-time complex automotive applications such as drive-by-wire applications. It supports both time-triggered and event-triggered transmissions and offers a data rate of up to 10 Mbps. The structure of the FlexRay protocol is illustrated in Fig. 3. A communication cycle is one complete instance of a communication structure that repeats periodically. Each communication cycle (also known as *cycle*) consists of a mandatory static segment, optional dynamic segment, optional symbol window, and mandatory network idle time. The static segment consists of multiple equally sized time slots that are used to send time-triggered messages. Each static segment slot consists of a header, payload (up to 254 bytes), and trailer segments. The TDMA media access scheme is employed in FlexRay static segment, where each ECU is assigned a particular static segment slot and a cycle number to transmit messages.

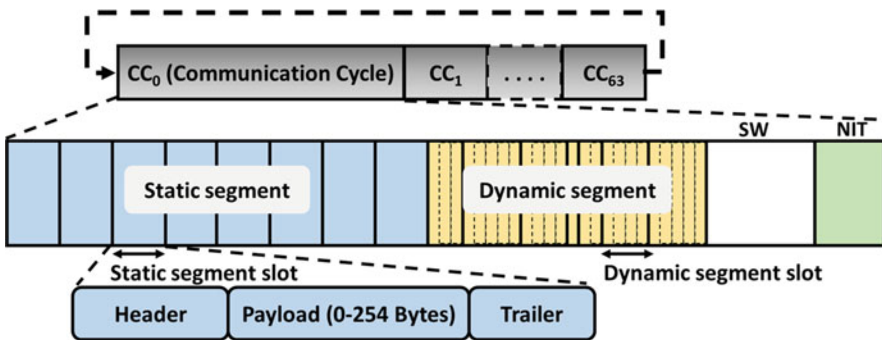


Fig. 3 Structure of the FlexRay protocol

On the other hand, the dynamic segment consists of variable-sized dynamic segment slots that are used to send event-triggered messages. Moreover, the dynamic segment employs a Flexible-TDMA media access scheme where the highest priority ECU gets access to the bus. The symbol window segment is used for signaling the start of the first communication cycle and network maintenance. Lastly, the network idle time segment helps with maintaining inter-ECU synchronization.

3.3 Attack Models

In this work, we focus on protecting the vehicle from masquerade and replay attacks as they are the most common, hard to detect, and can have a severe impact. The increased connectivity of modern vehicles with the external environment has created multiple pathways (attack vectors) to gain access to the in-vehicle network and ECUs. An attacker can choose a variety of attack vectors to gain access to the in-vehicle network and masquerade as an existing ECU or replay valid message transmissions to achieve malicious goals. In this study, we considered the most common and practical attack vectors in vehicles, which include connecting to the OBD-II port, connecting to systems that communicate with the external systems (such as infotainment systems), probe-based snooping on the vehicle bus, and replacing an existing ECU. Our framework can still be effective even when the attacker gains access to the in-vehicle network via other attack vectors.

3.4 Security Model

In this work, we focus on achieving the following key security objectives in vehicles: *(i)* confidentiality of message data and *(ii)* the authentication of ECUs. Meeting these objectives is crucial as it can help prevent masquerade and replay attacks. *Confidentiality* refers to the practice of protecting information from unauthorized ECUs, whereas *authentication* refers to the process of correctly identifying an ECU. In this study, we employ AES to achieve confidentiality by encrypting message data using a shared secret key. Moreover, we evaluate the choice of using RSA and ECC for setting up shared secret keys. However, it should be noted that neither RSA nor ECC is used for encrypting messages as they are much slower than AES. While AES with 128-bit keys (AES-128) is considered very secure today, the advent of quantum computing may challenge this assumption. Hence, we also consider AES-192 and AES-256. As each ECU can have messages of various criticalities, every ECU in the system can run all three variants of AES. Section 4.6 discusses the complete encryption/decryption flow in detail. Moreover, the key size for encrypting/decrypting messages is assigned based on the security requirements of a message, which is discussed in detail in Sect. 4.3.

3.5 Definitions

The *SEDAN* system model has the following inputs:

- Set of heterogeneous (1 or 2 core) ECUs $N = \{1, 2, \ldots, N\}$;
- Set of automotive applications $A = \{1, 2, \ldots, \lambda\}$ and set of tasks in the system $T = \{T_1 \cup T_2 \ldots \cup T_\lambda\}$, where T_a is the set of tasks in an application $a \in A$;
- Each task in T has a unique task ID $T_{ID} = \{1, 2, \ldots, G\}$;
- After task allocation, each task t is represented as $t_{q,n}$ where $q \in T_{ID}$ is the task ID, and $n \in N$ is the ECU to which the task t is mapped;
- Every task t is characterized by the 4-tuple $\{\tilde{a}_{q,n}, \tilde{p}_{q,n}, \tilde{d}_{q,n}, \tilde{e}_{q,n}\}$, where $\tilde{a}_{q,n}$,, $\tilde{p}_{q,n}$, $\tilde{d}_{q,n}$, and $\tilde{e}_{q,n}$ represent the arrival time, period, deadline, and execution time of the task, respectively;
- For each ECU $n \in N$, $S_n = \{s_{1,n}, s_{2,n} \ldots, s_{K_{n,n}}\}$ is the set of signals transmitted from the ECU; K_n is the total number of signals in n;
- Every signal $s_{i,n} \in S_n$, $(i = 1, 2 \ldots, K_n)$ is characterized by the 4-tuple $\{\overline{a}_{i,n}, \overline{p}_{i,n}, \overline{b}_{i,n}, \overline{d}_{i,n}\}$, where $\overline{a}_{i,n}$ $\overline{p}_{i,n}$, $\overline{b}_{i,n}$, and $\overline{d}_{i,n}$ are the arrival time, period, deadline, and data size (in bytes) of signal $s_{i,n}$ respectively;
- After frame packing, each ECU has a set of messages $M_n = \{m_{1,n}, m_{2,n}, \ldots, m_{R_n,n}\}$, where R_n is the total number of messages in n;
- Every message $m_{j,n} \in M_n$, $(j = 1, 2, \ldots, R_n)$ is characterized by the 5-tuple $\{a_{j,n}, p_{j,n}, d_{j,n}, b_{j,n}, \Delta_{j,n}, \psi_{j,n}\}$ where $a_{j,n}$, $p_{j,n}$, $d_{j,n}$, $b_{j,n}$, $\Delta_{j,n}$, and $\psi_{j,n}$ are the arrival time, period, deadline, data size (in bytes), and minimum security requirement of the message $m_{j,n}$ (see Sect. 4.3), respectively. $\psi_{j,n}$ is a binary variable that has a value = 1 when the security constraints of the message are satisfied. Otherwise $\psi_{j,n} = 0$;

Problem Objective: In this work, we focus on maximizing security (aggregate security value, described in Sect. 4.4) while synthesizing a design time schedule for time-triggered tasks and messages that satisfy three types of constraints: *(i)* real-time timing and deadline constraints for tasks and messages in all applications; *(ii)* minimum security constraints for each message in the system, *(iii)* ensure no ECU utilization exceeds 100%.

4 *SEDAN* Framework: Overview

A high-level overview of the *SEDAN* framework is illustrated in Fig. 4, with all the design time steps in gray boxes and the runtime steps in green boxes. The steps involved in the *SEDAN* framework can be mainly classified into two categories: *(i)* security operations that improve the security of the system and *(ii)* real-time operations that satisfy the application's real-time performance objectives. At *design time*, *SEDAN* begins by allocating tasks to available ECUs in the system and generates the set of signals needed for inter-task communication. These signals are packed into messages using a frame packing approach, and security requirements

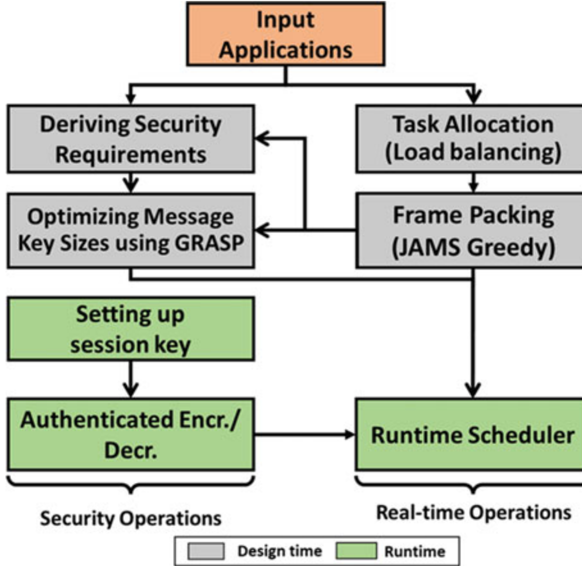


Fig. 4 Overview of the *SEDAN* framework

are derived for each message. The size of the keys used for encryption and decryption of the messages are optimized using a greedy randomized adaptive search procedure (GRASP) metaheuristic. At *runtime*, *SEDAN* starts with setting up the session keys, which will be used for generating keys used for authenticated encryption and decryption of messages. Lastly, a runtime scheduler schedules messages at runtime by using the previously generated keys and the optimal design time schedule. Each of these steps is discussed in detail in the subsequent subsections.

4.1 Task Allocation

This is the first step of the *SEDAN* framework and occurs at design time. The main goal of this step is to quickly allocate each task in the system to an available ECU that results in uniform real-time utilization across ECUs. This makes the *load-balancing task allocation scheme* a good choice for this step. Moreover, if there are some tasks that need to be allocated to certain ECUs, e.g., due to being in close proximity to sensors or actuators that they use heavily (or exclusively), we pre-allocate those tasks and do not include them in the set of mappable tasks for allocation.

For any task (t_q), the real-time utilization of the task ($\widetilde{U}_{t_q}$) is defined as the ratio of execution time ($\tilde{e}_q$) and the period ($\tilde{p}_q$) of the task, as shown in (1). The real-time

utilization of any given ECU ($\widetilde{U}_n$) is the sum of the real-time utilizations of the tasks ($\widetilde{U}_{tq,n}$) allocated to that ECU, and is computed using (2):

$$\widetilde{U}_{t_q} = \frac{\tilde{e}_q}{\tilde{p}_q} \tag{1}$$

$$\widetilde{U}_n = \sum_{q=1}^{G_n} \left(\widetilde{U}_{tq,n}\right) \tag{2}$$

Our proposed *load-balancing task allocation* scheme begins by initializing all the ECUs' real-time utilization ($\widetilde{U}_n$) to zero and computing the real-time utilization of all the tasks ($\widetilde{U}_{t_q}$) in the system using (1). The allocation subsequently occurs in three steps: *(i)* the set of ECUs in the system is sorted in the increasing order of the ECU real-time utilization ($\widetilde{U}_n$); *(ii)* the first unallocated task in the set of tasks (T), sorted in decreasing order of real-time utilization, is selected and allocated to the least loaded ECU (i.e., ECU with the lowest utilization); and *(iii)* the task's real-time utilization ($\widetilde{U}_{t_q}$) is added to the allocated ECU's real-time utilization ($\widetilde{U}_n$). These three steps are repeated until all the unallocated tasks in T are allocated. If any task, $t \in T$, cannot be allocated to an ECU during this process, then there exists no solution for the given configuration. Otherwise, at the end of this step, each task in the system is allocated to an ECU. After the task allocation step, the set of signals S_n is generated for each ECU based on the precedence constraints of tasks in the application.

We also explored other allocation schemes that minimize the total communication volume between ECUs. However, it resulted in allocations that resulted in non-uniform load across ECUs, which violated the ECU utilization constraints after implementing security mechanisms.

4.2 Frame Packing

Frame packing is defined as the grouping of signals in each ECU into messages. This is done to maximize the bandwidth utilization of the communication bus. The set of signals generated by the task allocation step is given as the input to this step. The following conditions need to be satisfied to successfully pack the signals into messages: *(i)* for any two signals to be packed into the same message, they must originate from the same source ECU; *(ii)* signals with the same periods are packed together to avoid multiple message transmissions; and *(iii)* the total computed payload of the message is the sum of the size of the cipher generated by AES and the size of the MAC; and should not exceed the maximum possible FlexRay payload size. Because of the nature of AES, *the size of the generated cipher is independent of the key size*. However, the size of the cipher is dependent on the

input size to the AES, which is the sum of signal sizes grouped in that message. Thus, the cipher size can be expressed as $\lceil$*sum of signal sizes in the message*/16$\rceil$, and the size of MAC is set to the maximum of the minimum required MAC size (49 bits, explained further in Sect. 4.3; a designer can also use a value greater than 49). In this work, we adapted a fast *greedy frame packing* heuristic proposed in [2] and enhanced it by integrating the computed payload size definition to generate a set of messages for each ECU.

4.3 Deriving Security Requirements

In this subsection, we present a novel methodology used in *SEDAN* to derive security requirements for each message. We employ a risk classification scheme defined in ISO 26262 [44] known as the Automotive Safety Integrity Level (ASIL) as the basis for deriving security requirements for each message in the system. Four different ASILs: ASIL-A, ASIL-B, ASIL-C, and ASIL-D, are defined in the ISO 26262 standard to classify applications based on their risk upon failure. Applications classified as ASIL-D have the lowest failure rate limit indicating high criticality, while ASIL-A applications are less critical and subject to fewer security requirements. The underlying assumption for deriving security requirements based on ASIL groups is that the applications that demand high safety levels are more critical and need to be better protected from cyber-attacks. Hence, the higher the safety requirement, the higher the security requirement.

In this work, we define two security requirements for every message based on their ASIL classification.

The first requirement is the *minimum key size* required to encrypt the message depending on its ASIL group, which is as follows: ASIL-A (128 bits), ASIL-B (128 bits), ASIL-C (192 bits), and ASIL-D (256 bits). The following methodology is followed to derive ASIL groups for all messages in the system. Each application is assigned an ASIL depending on the criticality and tolerance to failure. Each task in that application inherits the same ASIL, and so do the signals generated by these tasks. When these signals are packed into messages, the highest ASIL group among the signals in that message is assigned as the ASIL group ($m_{j,n}^{AG}$) of the message. We also assign a security score $\left(m_{j,n}^{SS}\right)$ to each safety-critical message depending on its assigned key size. In this study, we consider the following score based on the key size: 128-bit key (score = 1), 192-bit key (score = 2), and 256-bit key (score = 3). Additionally, each message is assigned a weight value called *ASIL weight* $\left(m_{j,n}^{AW}\right)$. A high *ASIL weight* value indicates a high message criticality and is analogous to a Risk Priority Number (RPN) that can be calculated using Hazard Analysis and Risk Assessment (HARA) approaches [45]. Using the above-mentioned metrics, we derive a *security value* $\left(m_{j,n}^{SV}\right)$ for each message as shown in (3). Lastly, to

quantitatively compare the security of different systems, we propose a metric called *Aggregate Security Value (ASV)*, which is computed using (4).

$$m_{j,n}^{SV} = m_{j,n}^{AW} * m_{j.n}^{SS} \tag{3}$$

$$\text{Aggregate Security Value } (ASV) = \frac{\sum_{n=1}^{N} \sum_{j=1}^{R_n} \left(\psi_{j,n} * m_{j,n}^{SV}\right)}{\sum_{n=1}^{N} R_n} \tag{4}$$

where $\psi_{j,n}$, and R_n are defined in Sect. 3.5. ASV is the ratio of the sum of security values of all messages in the system for which minimum security requirements are satisfied to the total number of messages in the system. ASV can be used to compare the security of various systems using the same encryption scheme. A system with a higher ASV value is more secure than a system with a lower value.

The <u>second</u> requirement is the minimum number of Message Authentication Code (MAC) bits required for a message based on the assigned ASIL group. This is derived using the failure rate limit of the ASIL group of the message. The failure rate limit is typically expressed as FIT (Failure in Time), which denotes the maximum number of acceptable failures per 1 billion hours of usage. Based on the specifications in the standard, ASIL-D has 10 FIT, ASIL-B and C have 100 FIT, and ASIL-A has 1000 FIT as their maximum limits. In other words, ASIL-D applications need less than 10^{-8} failures per hour, while ASIL-A applications can have up to 10^{-5} failures per hour. In this work, we derive the security requirements for each message in the system using the following method:

- Consider a message ($m_{j,n}$) with period ($p_{j,n}$) (in milliseconds);
- The number of transmissions of $m_{j,n}$ per second are $10^3/p_{j,n}$.
- The number of transmissions of $m_{j,n}$ per hour are $(3600{*}10^3)/p_{j,n}$.
- If there are k bits in the MAC field of a message, the *probability of failure due to an attacker guessing a valid MAC* (e.g., using brute-forcing or other methods) is 2^{-k} for one transmission of that message;
- Therefore, the probability of failure due to a compromised MAC for an hour-long transmission is $((3600{*}10^3)/p_{j,n}){*}2^{-k}$.
- For an ASIL-D application, the probability of failure needs to be less than 10^{-8} per hour, i.e., $((3600{*}10^3)/p_{j,n}){*}2^{-k} \leq 10^{-8}$.
- Thus, the minimum number of MAC bits ($\Delta_{j,n}$) required for the message ($m_{j,n}$) according to the ASIL-D requirement is:

$$\Delta_{j,n}(D) = k \geq \left\lceil Q + \log_2\left(\frac{1}{p_{j,n}}\right)\right\rceil \tag{5}$$

where Q is a constant and has a value of 48.35 for ASIL-D. Similarly, the minimum number of MAC bits required ($\Delta_{j,n}$) for other ASIL groups are calculated using

(5) by using $Q = 45.04$ for ASIL-B and ASIL-C and $Q = 41.72$ for ASIL-A. The different values of Q for each ASIL group are computed based on the FIT limit of that ASIL. Thus, for an ASIL-D message, for the most stringent (smallest) period, we observed (=1 ms), $\Delta_{j,\,n}(D) = 49$ bits (thus this is used in frame packing).

4.4 Optimizing Message Key Sizes Using GRASP

This is the last step of the design time process in *SEDAN*. This step aims to assign an optimal key size for each message in the system that maximizes the ASV while meeting all the security requirements and real-time deadline constraints. Additionally, we model the overhead caused by the security tasks (i.e., encryption and decryption) in terms of the additional ECU utilization (security-induced utilization) and latency (response time) of the message. For any given message ($m_{j,n}$) that is encrypted or decrypted using a block cipher, the security-induced ECU utilization ($\overline{U}_{m\,j,n}$) due to the message is computed using (6).

$$\overline{U}_{m\,j,n} = \left(\left\lceil \frac{b_j}{b_{size}} \right\rceil * \frac{T_{encr/decr}}{p_j} \right) \tag{6}$$

where b_{size} denotes the block size in bytes, and $T_{encr/decr}$ represents the time taken to encrypt or decrypt one block of data. Since AES is the encryption algorithm used in this study, the above equation can be re-written as shown in (7).

$$\overline{U}_{m\,j,n} = \left(\left\lceil \frac{b_j}{16} \right\rceil * \frac{T_{AES(X)}}{p_j} \right) \tag{7}$$

where $T_{AES(X)}$ is the time taken to encrypt or decrypt one block (16 Bytes) of data using AES with an X-bit long key (where X can be 128, 192, or 256). The security-induced utilization of any ECU $\left(\overline{U}_n\right)$ (computed using (8)) is the sum of the security-induced utilizations of all transmitted and received messages $\left(\overline{U}_{m\,j}\right)$ for that ECU. Hence, the total utilization of any ECU (U_n) is the sum of the real-time utilization $\left(\widetilde{U}_n\right)$ and security-induced utilization ($\overline{U}_n$) as shown in (9). Moreover, to avoid uncertainties and undesired latency overheads, we always ensure that the utilization of any ECU does not exceed 100%.

$$\overline{U}_n = \sum\nolimits_{j=1}^{R_n} \overline{U}_{m\,j,n} \tag{8}$$

$$U_n = \widetilde{U}_n + \overline{U}_n \tag{9}$$

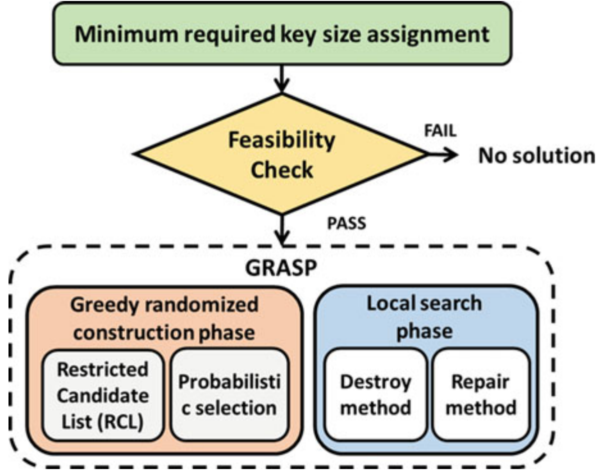


Fig. 5 Overview of the GRASP-based optimal message key size allocation step in *SEDAN*

In this study, we propose a heuristic approach to achieve this goal based on the *greedy randomized adaptive search procedure* (*GRASP*) metaheuristic [46]. An overview of this approach is illustrated in Fig. 5. Our proposed approach begins by taking the set of messages from the output of frame packing (Sect. 4.2) and the derived security requirements (Sect. 4.3) as inputs. An *initial solution* is generated by assigning the minimum required key sizes for all the messages based on the derived security requirements. This initial solution is subjected to a feasibility check which investigates the: *(i)* total ECU utilization (U_n) for all ECUs and *(ii)* number of missed deadlines using a design time scheduler. Moreover, we adapt the fast design time scheduling heuristic proposed in [2] to generate an optimal design time schedule. The initial solution is given to the GRASP only when there are no utilization violations at any ECU (i.e., $U_n \leq 100\% \; \forall$ ECUs) and deadline misses for any message. If any of the above-mentioned conditions fail, the optimal message key size allocation step terminates, and the system does not have a feasible solution. GRASP intelligently explores various message key sizes (that are greater than or equal to the minimum key size requirement for a message) and design time schedule configurations (i.e., assigning messages and ECUs to FlexRay static segment slots) to select a solution that maximizes ASV, with no security violations, real-time deadline misses, and ECU utilization violations (i.e., no ECU utilization exceeds 100%).

The GRASP metaheuristic is an iterative process in which each iteration has two major phases: *(i) greedy randomized construction phase* that tries to build a local feasible solution and *(ii) local search phase* that tries to investigate the neighborhood for a local optimum. In the end, the best overall solution is chosen as the final solution. The greedy randomized construction phase has two key aspects- the *greedy aspect* and the *probabilistic aspect*. The greedy aspect involves generating

a Restricted Candidate List (RCL), which consists of the best elements that will improve the partial solution (solution within the greedy randomized construction phase). And the probabilistic aspect involves selecting a random element from the RCL, which will be incorporated into the partial solution. It is important to note that the solutions generated during the greedy randomized construction phase are not necessarily optimal. Hence, a local search phase is used to improve the partial solution from the greedy randomized construction phase. The local search is an iterative process that uses destroy and repair mechanisms to search for local optimum within a defined neighborhood. The best solution is updated if an improved solution is found during the local search.

Algorithm 1: GRASP Based Optimal Message Key Size Assignment

Inputs: Set of nodes (N), Set of all messages (M), *init_solution*, *max_iterations*, RCL threshold (α), and destroy-repair threshold (β)

```
1:  best_solution ← init_solution
2:  for iteration = 1, …, max_iterations do
3:  |   current_solution ← greedy_randomized_construction(α, N, M)
4:  |   current_solution ← local_search(β, N, M, current_solution)
5:  |   if current_solution > best_solution do
6:  |   |   best_solution ← current_solution
7:  |   end if
8:  end for
```

Output: Optimal message key sizes for every message that results in maximum ASV and a feasible design time schedule with no deadline misses, security violations, and utilization of all ECUs below 100%.

Algorithm 1 presents an overview of our GRASP-based optimal message key size assignment approach. The inputs to Algorithm 1 are a set of nodes (N), a set of all the messages in the system (M), and the minimum required message key size assignment (*init_solution*), which is the initial solution given to GRASP to reduce the search space. In addition, the tunable parameters such as maximum iterations (*max_iterations*), RCL threshold (α), and a destroy-repair threshold (β) are given as input to GRASP to efficiently look for solutions in the search space. The algorithm starts by assigning the *init_solution* to the *best_solution* in step **1**. GRASP iteratively tries to find a better solution in steps **2–8** until *max_iterations* is reached. In each iteration *greedy_randomized_construction()* in step **3**, generates a local feasible solution (*current_solution*) which is updated using *local_search()* in step **4**. If a better solution is found at the end of the local search phase, the *best_solution* is updated in steps **5–7**. The output of the algorithm is an optimal message key size for every message and a feasible design time schedule with no deadline misses, no security violations, and no ECU utilization exceeding 100%. *Note*: Every solution in GRASP consists of two attributes *(i)* key sizes for all the messages and *(ii)* ASV of the system as a result of the key size assignment. Moreover, every solution generated by GRASP ensures that no message is allocated a key size less than the key size assigned in the initial solution, and the overall system ASV is always greater than the ASV of the initial solution.

4.4.1 Greedy Randomized Construction Phase

The greedy randomized construction phase tries to generate a feasible solution in every iteration of GRASP by increasing the key sizes of some of the non-ASIL-D messages. The goal here is to maximize the ASV of the system without any deadline, security, and ECU utilization violations. Moreover, it also ensures that no message is allocated a key size less than the key size allocated in the initial solution (minimum required key size). The solution generated by the greedy randomized construction phase will be given as the input to the local search phase for refinement.

Algorithm 2: greedy_randomized_construction (α, *N*, *M*)

Inputs: RCL threshold (α), set of nodes (*N*), and set of all messages (*M*)

1: $\widetilde{M} \leftarrow \{m \in M \mid m^{AG} \neq \text{ASIL-D}\}$
2: **Increment** m^{SS} by 1 $\forall\, m \in \widetilde{M}$ and compute m^{SV}
3: **Sort** $\widetilde{M}$ in the increasing order of m^{SV}
4: **while** $\widetilde{M} \neq \{\ \}$ **do**
5: $SV_{min} = \min(\{m^{SV}\ \forall\, m \in \widetilde{M}\})$
6: $SV_{max} = \max(\{m^{SV}\ \forall\, m \in \widetilde{M}\})$
7: $RCL \leftarrow \{m \in \widetilde{M} \mid m^{SV} \geq SV_{min} + \alpha * (SV_{max} - SV_{min})\}$
8: $\overline{m} \leftarrow$ random element from *RCL*
9: **Increment** the key size of $\overline{m}$ to the next higher key size
10: **if feasibility_check**( ) == *false* **do**
11: **Revert** the key size of $\overline{m}$ back to its previous key size
12: **Decrement** m^{SS} by 1 for $\overline{m}$ and compute m^{SV}
13: **end if**
14: **Remove** $\overline{m}$ from $\widetilde{M}$
15: **end while**
16: *current_solution* $\leftarrow$ {**calculate_ASV**( ), message key size assignment}

Output: Local feasible solution that results in a feasible schedule with no deadline misses, security violations, and utilization of all ECUs below 100%.

Algorithm 2 shows the pseudocode of the greedy randomized construction phase where the inputs are: set of nodes (*N*), set of messages (*M*), and RCL threshold (α). A set of non-ASIL-D messages ($\widetilde{M}$) is generated in step **1**. In step **2**, the security score of each message (m^{SS}) in $\widetilde{M}$ is incremented by one, and the security values of the messages (m^{SV}) are updated using (3). The $\widetilde{M}$ is sorted in the increasing order of m^{SV} and the ties are resolved based on the message period in step **3**. In steps **4**–**15**, the algorithm tries to find a local solution by incrementing key sizes for some messages that would result in no deadline, security, and ECU utilization violations. The minimum (SV_{min}) and maximum (SV_{max}) security values of messages in $\widetilde{M}$ are computed in steps **5**, **6** respectively. The *RCL* consists of messages in $\widetilde{M}$, that will result in increased ASV when their key size is incremented. Hence, the messages whose security value (m^{SV}) is within the interval $[SV_{min} + \alpha\ (SV_{max} - SV_{min}), SV_{max}]$ are added to the *RCL* in step **7**. This is the greedy aspect of the greedy randomized construction step. Moreover, GRASP employs an RCL threshold ($\alpha \in [0, 1]$) to

regulate the quality of the generated RCL. The threshold (α) controls the amount of greediness and randomness in the algorithm. The $\alpha = 1$ case corresponds to a pure greedy approach, while $\alpha = 0$ is equivalent to a purely random approach. A random message ($\overline{m}$) is selected from the *RCL* (probabilistic selection) in step **8**, and its key size is incremented to the next higher key size in step **9** (i.e., 128 $\rightarrow$ 192 or 192 $\rightarrow$ 256). The *feasibility_check()* in step **10**, checks for any *(i)* ECU utilization violations (i.e., any ECU utilization >100%) and *(ii)* deadline misses using the design time scheduling heuristic proposed in [2]. If any of the above-mentioned checks fail, the *feasibility_check()* will return *false* and reverts the key size of ($\overline{m}$) back to its previous key size in step **11**. Moreover, the m^{SV} of $\overline{m}$ is re-computed after decrementing the m^{SS} by one in step **12**. Otherwise, the key size increment is left unchanged. The message ($\overline{m}$) is removed from $\tilde{M}$ and the steps **5–14** are repeated until there are no messages left in $\tilde{M}$. Lastly, in step **16**, the current message key size assignment and the ASV of the system (using *calculate_ASV()*) are assigned to the *current_solution*. The function *calculate_ASV()* is implemented using (4).

4.4.2 Local Search Phase

The local search phase tries to iteratively improve the solution found in the greedy randomized construction phase by investigating a defined neighborhood in the search space. The local search phase achieves this by using *destroy* and *repair* methods, which remove a part of the solution and recreate a feasible solution, respectively. In this study, we define the neighborhood as the set of solutions that are generated by randomly changing key sizes for β number of messages. The parameter β is known as the destroy-repair threshold, which controls how much to destroy or repair in each iteration of the local search. These random changes in message key sizes help in recovering from suboptimal ordering (sorting in the increasing order of m^{sv}) of messages in the greedy randomized construction phase.

The pseudocode of the local search procedure is illustrated in Algorithm 3. The *destroy()* function in steps **1–4** randomly selects a message from the set of messages that are allocated a key size higher than the minimum required key size and decreases the key size to the next smaller key size. The function *min_score()* in step **2** returns the minimal security score demanded by the assigned ASIL group. The *repair()* method in steps **5–18** aims to increase the key size for β non-ASIL-D messages and computes the local solution using *local_solution()*. The *repair()* step always selects a message that results in a maximum increase in the ASV of the system (as shown in step **8**). The ties in step **8** are resolved based on the ASIL group, and if multiple messages have the same ASIL group, one message is selected at random. In steps **19–29**, the local search algorithm iteratively explores the neighborhood around the *current_solution* using *destroy()* and *repair()* to find a better solution. In each iteration, the value of β is chosen randomly from [2, β_{max}]. In steps **21–24**, the function *destroy()* is modeled as a stochastic process that is controlled by the key decrease probability (p_{kd}). Lastly, the *current_solution*

is updated if a better *local_solution* is found in the repair method in steps **25–28**. In each iteration of GRASP, at the end of the local search phase, a local optimum is found if there exits one. Otherwise, the solution remains unchanged from the greedy randomized construction phase.

Algorithm 3: local_search (β, *N*, *M*, *current_solution*

Inputs: Destroy-repair threshold (β), set of nodes (*N*), set of all messages (*M*), and *current_solution*

1: **function** *destroy* (M)
2: $M_d = \{m \in M \mid m^{SS} >$ **min_score**$(m^{AG})\}$
3: **Decrement** the key size of a random message ($\overline{m}$) in M_d
4: **end function**
5: **function** *repair* (β, M, N)
6: $M_r = \{m \in M \mid m^{AG} \neq$ ASIL-D$\}$
7: **while** ($\beta > 0$) or ($M_r \neq \{\ \}$) **do**
8: $\overline{\overline{m}} = \{m \in M_r \mid \Delta$ASV is maximum$\}$
9: **Increment** the key size of the message ($\overline{\overline{m}}$)
10: **if feasiblity_check( )** == *false* **do**
11: **Revert** the key size of ($\overline{\overline{m}}$) back to the previous key size
12: **else do**
13 $\beta = \beta - 1$
14: **end if**
15: **Remove** ($\overline{\overline{m}}$) from M_r
16: **end while**
17: **return** {**calculate_ASV( )**, message key size assignment}
18: **end function**
19: **for** *local_iteration* = 1,…, *max_local_iterations* **do**
20: β = **random_integer**(2, β_{max})
21: **if** p_{kd} > **random**(0,1) **do**
22: **destroy** (M)
23: $\beta = \beta - 1$
24: **end if**
25: *local_solution* ← **repair** (β, M, N)
26: **if** *local_solution* > *current_solution* **do**
27: *current_solution* ← *local_solution*
28: **end if**
29: **end for**

Output: Local optimum with in the defined neighborhood- if there exists one; Otherwise, the same solution as greedy_randomized_construction( ).

It is important to note that when the message key size is changed, the size of the output cipher and MAC (or the message size) remains unchanged. The key size only affects the time taken to encrypt/decrypt the message, which impacts the security-induced utilization of the sender and receiver ECUs. Moreover, the real-time utilization of the ECUs also remains unchanged, as the execution time of time-triggered tasks does not change with changing message key sizes.

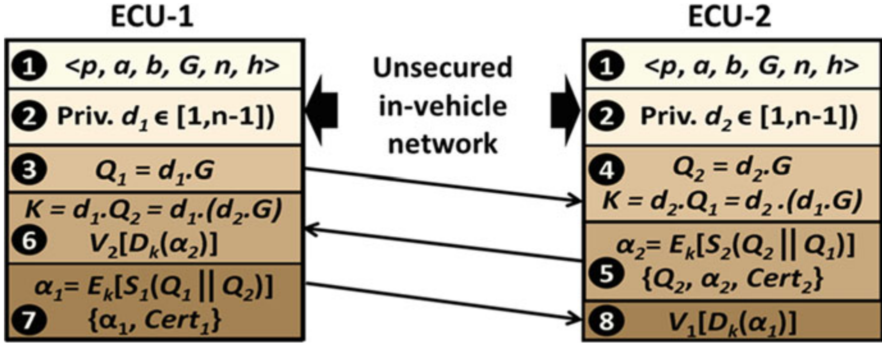


Fig. 6 Overview of steps involved in setting up a session key using the STS protocol with ECC

4.5 Setting Up Session Key

In this subsection, we discuss the first runtime step of the *SEDAN* framework. It involves settings up session keys required for generating keys that will be used for the encryption and decryption of messages. This is a crucial step in improving the security of the vehicle, as using the same key every time for encryption and decryption for the entirety of the vehicle's lifetime makes the system highly vulnerable to cyber-attacks. Hence, during runtime, we generate a new key for every session (called session key), which will be used for generating keys that will be used for encryption and decryption of messages.

A session is defined as the time duration between the start of a vehicle to turning off the vehicle. Since we use symmetric key encryption, all ECUs in the system need to have the same secret key to function properly. As traditional automotive networks do not have any inbuilt security features, exchanging the session keys between ECUs over an unsecured channel is a major challenge. In this work, we adapt the Station-to-Station (STS) key agreement protocol [47], which is based on the famous Diffie-Hellman key exchange method [45], to the automotive domain (as simple Diffie-Hellman is vulnerable to man-in-the-middle attacks), to securely transfer session keys between ECUs over an unsecured FlexRay bus. Moreover, within the STS protocol, we employ elliptic curve cryptography (ECC) as the basis for key agreement instead of RSA. This is mainly because ECC is faster and has a lower memory footprint for the same level of security compared to the RSA (as discussed in Sect. 5.2). The overview of steps involved in STS protocol with ECC for two ECU cases is illustrated in Fig. 6.

The STS approach begins with two ECUs agreeing upon a set of domain parameters that define the elliptic curve. These parameters are shown in the first step in Fig. 6, where the parameter p defines the field, a and b define the elliptic curve, G is the generator and n is its order, and h is the co-factor. Additionally, each ECU utilizes an asymmetric key pair for authentication operations (sign and verify). In the second step, each ECU generates a random private number (d_1 in

ECU 1 and d_2 in ECU 2), which is not shared with any other ECU in the system. In step 3, ECU1 performs an elliptic curve scalar multiplication (hereafter referred to as scalar multiplication) of the private number d_1 and generator G. The output Q_1 is transmitted to ECU2 over an unsecured FlexRay bus. In step 4, a similar scalar multiplication between d_2 and G is performed at ECU2, but the output Q_2 is not sent to ECU1. ECU2 then computes the common secret key K (session key) by performing the scalar multiplication of the private number d_2 and the received output Q_1. In step 5, ECU2 computes the signature (S_2 ()) of the concatenation of Q_2 and Q_1 (represented as $Q_2 \mid\mid Q_1$) using its private key of the asymmetric key pair. The output signature is encrypted (E_k ()) using the computed session key from the previous step, which produces the cipher α_2. The scalar multiplication output (Q_2), output cipher (α_2), and certificate ($Cert_2$) are all transmitted to ECU1 over the unsecured FlexRay bus. The certificate is issued by a trusted certificate authority (CA), which is used to prove the ownership of a public key. The certificate consists of the public key of the owner and signature of the CA and will be programmed in the ECUs by the manufacturer. The public key of the CA is used to verify the certificate and extract the public key of the owner. In step 6, when the ECU1 receives the output of step 5 from ECU2, it performs a scalar multiplication of private number d_1 and Q_2 to produce the shared secret key K (session key). Moreover, ECU1 utilizes the key K to decrypt (D_k ()) the received cipher (α_2) and verifies (V_1()) the decrypted output using the public key extracted from the certificate of ECU2 ($Cert_2$). The session key K is accepted by ECU1 only when the verification is successful, implying a successful authentication of ECU2. In step 7, ECU1 computes the signature (S_1()) of the concatenation of Q_1 and Q_2 (represented as $Q_1 \mid\mid Q_2$) using its private key of the asymmetric key pair. The resulting output is encrypted using the key K that generates the cipher (α_1), which is transmitted to ECU2 along with the certificate ($Cert_1$). Lastly, in step 8, at ECU 2, the received cipher (α_1) is decrypted using the key K, and the output is verified using the public key extracted from the certificate of ECU1 ($Cert_1$). The session key K is accepted to use for the session only when the verification is successful. Thus, all the ECUs are authenticated, and a common secret key (session key) is established at every ECU without actually exchanging the actual key over the unsecured bus. Additionally, the STS protocol uses no timestamps and provides perfect forward secrecy. Using a standard AES key schedule at every ECU, this session key is then used to generate 128-bit, 192-bit, and 256-bit keys. These resulting keys are used for encrypting and decrypting messages at runtime. Moreover, in order to avoid interference with the time-critical messages, the messages related to the security operations utilize a small number of reserved FlexRay frames. To speed up the startup process, we assume that the manufacturer pre-programs some of the session keys during manufacturing. New keys are generated continuously during the idle time of an ECU, saved in local memory, and used in future sessions. To further speedup this process, the public keys of the trusted ECUs can be pre-programmed in the ECU's tamper-proof memory, thereby avoiding the verification of the certificate, which saves both computation time and network bandwidth.

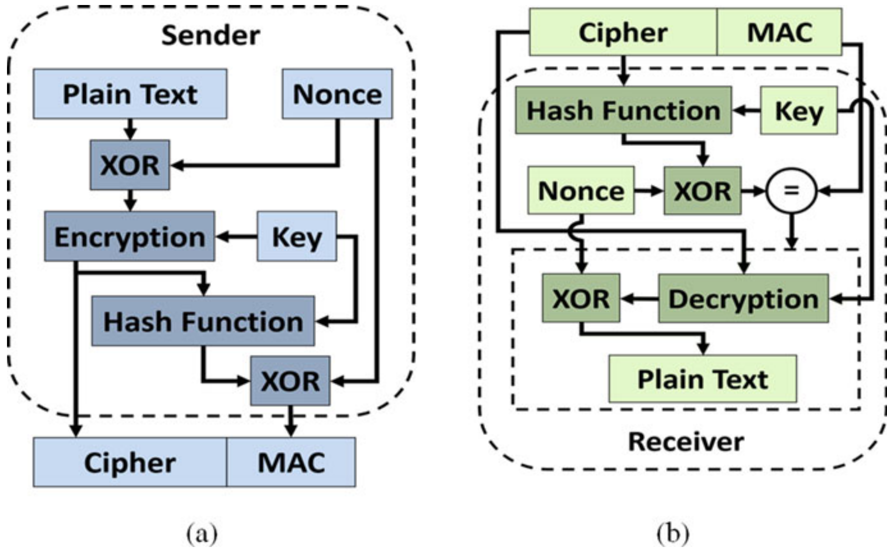


Fig. 7 (**a**) Authenticated encryption at sender ECU; (**b**) Authenticated decryption at receiver ECU

It is essential to highlight that, even if there was an attacker already in the system during the key setup phase, the attacker could not compute the secret key with the publicly available results due to the discrete logarithm problem [48]. Moreover, the common man-in-the-middle attack that breaks the standard Diffie-Hellman approach [49] fails with STS as the attacker cannot authenticate successfully.

4.6 Authenticated Encryption/Decryption

In this subsection, we discuss the various steps involved in authenticated encryption employed in *SEDAN*. Authenticated encryption refers to simultaneously providing a message with confidentiality and authenticity, which is a well-known technique in the literature. We discuss this step in detail here to highlight how *SEDAN* leverages this process to achieve a more secure runtime system. The authenticated encryption and decryption phases are illustrated in Fig. 7a, b, respectively.

The authenticated encryption at the sender ECU begins with an XOR operation between the *plain text* (message data) and a *nonce* (random number), and the result is encrypted using AES with the key size assigned to the message (as discussed in Sect. 4.4). The XOR operation with a nonce is performed to avoid generating the same cipher every time when the input data is the same for long durations. Even though protecting the system from side-channel attacks is not within the scope of this work, this simple step could be the first step in preventing information leakage. A cryptographic hash function (MD5) takes the output cipher and the key used for

encryption to produce a hash, which is XORed with a nonce to generate the MAC. The output MAC size is truncated if needed and set to be at least the size computed in Sect. 4.3. The generated MAC is then transmitted with the encrypted message data in the payload section of the FlexRay frame.

The authentication decryption at the receiver ECU begins by authenticating the sender ECU of a received message. The received cipher and the selected key are given to the same cryptographic hash function whose result is XORed with a nonce to generate a local MAC. The authentication of the sender ECU is successful only when the local MAC matches the received MAC. Otherwise, the authentication process fails, and the received message is discarded. After successful authentication of a sender ECU, AES decryption is initiated, and the output is XORed with the nonce to extract the original message data as plain text.

As discussed in Sect. 3.3, we mainly focus on protecting the system from masquerade and replay attacks as they are the most common, hard to detect, and severely impact system safety and performance. The system is protected against masquerade or impersonation attacks by authenticating the ECUs in the system using the STS protocol, which establishes the session keys used for encryption and decryption only after successful authentication. The attacker fails to authenticate due to the lack of trusted certificates and cannot masquerade as a legitimate ECU. Moreover, the MAC generated in the authenticated encryption protects the system from replay attacks. During the MAC generation, it is essential to XOR the output of the hash function with the nonce as it makes the messages resilient to replay attacks. During a replay attack, the authenticity of the replayed message fails as the nonce used in computing the local MAC at the receiver is different from the nonce used in generating the received MAC at the sender. This mismatch in MAC will result in discarding the message sent by the attacker. Moreover, in the event of a man-in-the-middle attack, where the attacker tries to modify the message payload, the MAC comparison fails, resulting in protecting the integrity of the messages. Also, if an attacker eavesdrops on the network, the attacker would still be unable to decrypt the encrypted messages, as no keys are exchanged on the network. In this manner, we achieve confidentiality of the message data. Hence, using the proposed *SEDAN* framework, we were able to achieve all the security objectives, namely confidentiality, integrity, and ECU authenticity (as discussed in Sect. 3.4).

4.7 Runtime Message Scheduler

Runtime message scheduling is the last step in the *SEDAN* framework. It takes the unique values of the cipher and MAC generated in the previous step and packs them into FlexRay frames generated during the frame packing step (Sect. 4.2). Other control fields, such as the fields in the header and trailer segments that are required for the transmission of FlexRay frames, are also added by the scheduler. The runtime scheduler uses the design time generated message schedule and interacts with the FlexRay protocol engine to schedule messages on to the FlexRay bus at runtime.

5 Experiments

5.1 Experimental Setup

We evaluated the performance of our proposed *SEDAN* framework by comparing it with the best-known prior work [21]. In [21], the authors proposed a technique that uses simulated annealing to minimize the end-to-end latencies of all in-vehicle network messages and uses symmetric key encryption with the time-delayed release of keys to improve security in a vehicle system. Since [21] does not support variable key sizes, three different variants of [21] are implemented using AES encryption with fixed key sizes of 128, 192, and 256 bits, which are referred to as 'Lin et al. AES-128', 'Lin et al. AES-192', and 'Lin et al. AES-256' respectively in the experimental results. We generated several test cases based on automotive network and ECU computation data extracted from a real-world vehicle (2016 Chevrolet Camaro) that we have access to. We modeled the network and ECU computation data as directed acyclic graphs (DAGs), which were generated using TGFF [50]. We developed multiple synthetic test cases by scaling this data based on different combinations of the number of ECUs, number of applications, number of tasks in each application, and the range of periods. Moreover, we assume that the deadline for both tasks and messages are equal to their period. Lastly, we considered the FlexRay 3.0.1 [51] protocol with the following network parameters for all experiments: cycle duration of 5 ms with 62 static segment slots, with a slot size of 42 bytes, and 64 communication cycles.

5.2 Benchmarking Encryption Algorithms

To accurately capture the runtime behavior of session key generation and authenticated encryption/decryption steps, we implemented various encryption algorithms in the software. We implemented AES-CBC with key sizes of 128, 192, and 256 bits, RSA with key sizes of 512, 1024, 2048, and 4096 bits, and the ECC with key sizes of 256 and 384-bits using OpenSSL [52]. All these algorithms were executed on an ARM Cortex-A9 CPU on a ZedBoard, which has similar specifications as state-of-the-art ECUs [53, 54].

Table 1 shows the average AES encryption/decryption times with different standard key sizes for one block of data (16 Bytes) on an ARM Cortex A9 CPU. These values are used to model the latency overhead on each message due to the added security mechanisms at design time. They are also used in scheduling decisions and computing the response time of the messages. The encryption and decryption times of RSA with 512, 1024, 2048, and 4096-bit keys and ECC with 256 and 384-bit keys are also shown in Table 1. These values are considered in choosing between RSA and ECC as the cryptographic scheme in the STS protocol. The NIST recommends a key size of 2048-bits for RSA [55], while

Table 1 Execution time (ms) of AES, RSA, and ECC on ARM Cortex A9

Cryptographic scheme	Key size	Encryption / Decryption	
AES	128	0.35	
	192	0.393	
	256	0.415	
Cryptographic scheme	**Key size**	**Public key operation**	**Private key operation**
RSA	512	2.01	19.89
	1024	6.48	139.15
	2048	23.65	911.8
	4096	91.52	6283.2
ECC	256	59.8	17.1
	384	182.4	50.4

NSA recommends a 256-bit key size for SECRET level and a 384-bit key size for TOP SECRET level using ECC [56]. Moreover, ECC with 224, 256, and 384-bit key sizes provides similar security as RSA with 2048, 3072, and 7680 key sizes, respectively [57]. In this work, we consider the minimum key sizes based on the above-mentioned recommendations. From Table 1, it can be seen that RSA is faster for verifying signatures (operation performed using the public key) and much slower for generating signatures (operation performed using the private key). On the other hand, ECC is much faster for generating signatures while relatively slower for verifying signatures. It is important to note that the security (provided by RSA using the equivalent key size) doubles when the ECC key size is increased from 256 to 384. However, since the automotive systems are highly resource-constrained, we choose to employ ECC with a 256-bit key size (which still provides higher security than the minimum recommended key size for RSA) for cryptographic operations in the STS key agreement protocol. Moreover, the ECC execution time values are used in estimating the worst-case time required for setting up a session key, which is 0.24 s for a 256-bit key, while an equivalent RSA 2048 takes 3.72 s. Thus, it is evident that ECC is much faster than RSA for a similar level of security. Moreover, ECC can provide a similar level of security compared to RSA, with a much shorter key size. Lastly, when we profiled the MD5 hashing algorithm used in the authenticated encryption step, we observed that processing one block of data takes about 2.68 μs.

Moreover, with the increasing complexity of automotive applications, designing security mechanisms that result in minimal power consumption is crucial. Hence, we profiled the security mechanisms studied in this work and presented the power consumption results in Table 2. Other overheads, such as memory consumption, are not explicitly modeled as most modern-day ECUs have sufficient memory to store the small keys needed for secure transfers. Additionally, the designer can limit the number of pre-computed session keys that can be stored to minimize the memory overhead. Based on the results in Tables 1 and 2, it is evident that ECC has lower computation and memory overhead than RSA for the same level of security. Hence, in *SEDAN,* we authenticate the ECUs in the system and setup session keys using the

Table 2 Power consumption of AES, RSA and ECC on ARM Cortex A9

Cryptographic scheme	Key size	Encryption / Decryption	
AES (mW)	128	57.76	
	192	58.04	
	256	60.19	
Cryptographic scheme	**Key size**	**Public key operation**	**Private key operation**
RSA (W)	512	0.28	0.65
	1024	0.34	1.22
	2048	0.72	1.91
	4096	1.08	2.58
ECC (W)	256	0.62	0.33
	384	0.93	0.58

STS protocol using the ECC. Additionally, we use AES to encrypt and decrypt the messages in the system using the keys computed from the session key.

5.3 GRASP Parameter Selection

To get an efficient solution using the GRASP, it is essential to select the appropriate values for the threshold parameters α and β_{max}. We ran a series of simulations by changing the value of α from 0 to 1 with an increment of 0.2, and the greedy randomized construction phase was run 1000 times using different input test cases. We observed that the mean solution approached a greedy solution, while the variance approached zero as α tends to 1. On the other hand, when α is small and close to zero, the mean solution approaches a random solution with high variance. Therefore, we selected $\alpha = 0.8$, which provided a good quality solution to the local search phase that resulted in a near greedy solution in the presence of a relatively large variance.

Moreover, we observed that $\beta_{max} = 3$ provided enough randomness to look for other solutions in each iteration of the local search phase. A higher value of β_{max} could result in an exhaustive local search leading to unreasonably long computation times. Also, the minimum value of β needs to be 2 to increase the key size of at least one message when the key size is reduced in the event of a destroy operation. This prevents the generation of a solution that results in lower ASV compared to the solutions in previous iterations. Lastly, a relatively small value for $p_{kd} = 0.3$ is chosen to avoid frequent key size decrements.

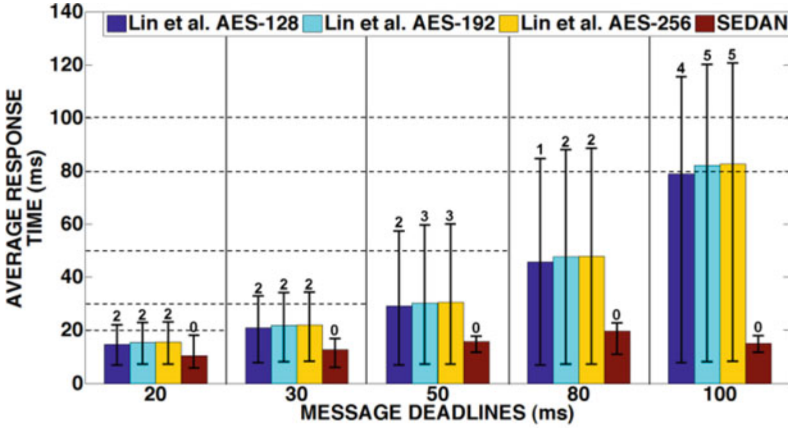
5.4 Response Time Analysis

In this subsection, we present the response time analysis by comparing our proposed *SEDAN* framework with the three variants of [21]. Response time of a message is defined as the end-to-end latency, which is the aggregate of the time for encryption and MAC generation, and queuing delay at the sender ECU; transmission time on the Flexray bus, and the time for MAC verification and decryption at the receiver ECU. We evaluated our proposed *SEDAN* framework, and the comparison works using three different test cases: (1) low input load- system with 5 ECUs (3 single-core and 2 dual-core) and 77 tasks that produced 57 (time-triggered) signals; (2) medium input load- system with 12 ECUs (9 single-core and 3 dual-core) and 126 tasks with 93 signals; and (3) high input load- system with 16 ECUs (12 single-core and 4 dual-core) and 243 tasks with 196 signals. The average message response time for the low, medium, and high input load cases with their deadlines on the x-axis is illustrated in Figs. 8(a–c). The confidence interval on each bar represents the minimum and maximum average response time of messages. The dashed horizontal lines represent different message deadlines. The number on top of each bar is the number of deadlines misses.

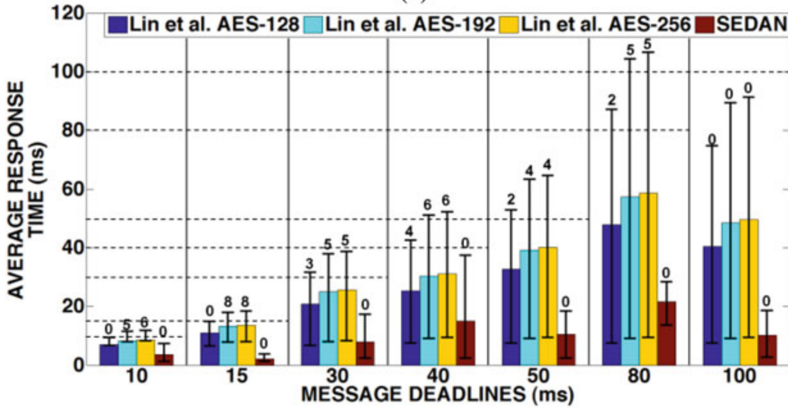
From Figs. 8(a–c), it is clear that *SEDAN* outperforms the three variants of [21] and achieves significantly lower average response times for all the messages under all input load cases. *SEDAN* achieves this by balancing security and real-time performance goals by optimizing key sizes while meeting message security requirements and ensuring that all ECU utilizations are below 100%. This prevents the messages from experiencing additional delays on top of the latency caused by the encryption-decryption processes. Moreover, all three variants of [21] experience significant authentication delays (time taken from the transmission of the message to decryption of the message) compared to *SEDAN*, which increases the response time of the messages when using [21]. These high authentication delays in [21] are because of the time-delayed release of keys, which is employed in all three variants of [21]. Also, the periodic computation of keys in every session at each ECU in all three variants of [21] results in high ECU utilization overhead resulting in increased response time and power consumption. Lastly, the requirement of large message buffers to hold multiple messages for longer durations in [21] (due to the time-delayed release of keys) further increases power consumption and response time.

5.5 Security Analysis

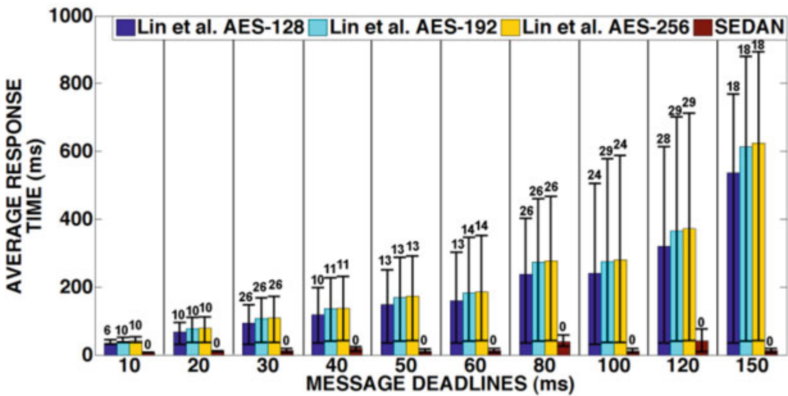
Table 3 shows the number of security violations in each technique under three different input load cases (as discussed in the previous sub-section). A security violation is defined as an instance when the derived security constraints (defined in Sect. 4.3) for a message are not met. From Table 3, it can be seen that the *SEDAN* and Lin et al. AES-256 are the only techniques that do not violate any security



(a)



(b)



(c)

Fig. 8 Comparison of the average response time of all messages under (**a**) low; (**b**) medium, and (**c**) high input application load conditions for Lin et al. AES-128, AES-192, AES-256 [21], and *SEDAN* (with the number of missed deadlines shown on the top of the bars)

Table 3 Total number of security violations for each input load configuration

Framework	Lin et al. 128	Lin et al. 192	Lin et al. 256	SEDAN
Low load	28	12	0	0
Medium load	45	16	0	0
High load	96	31	0	0

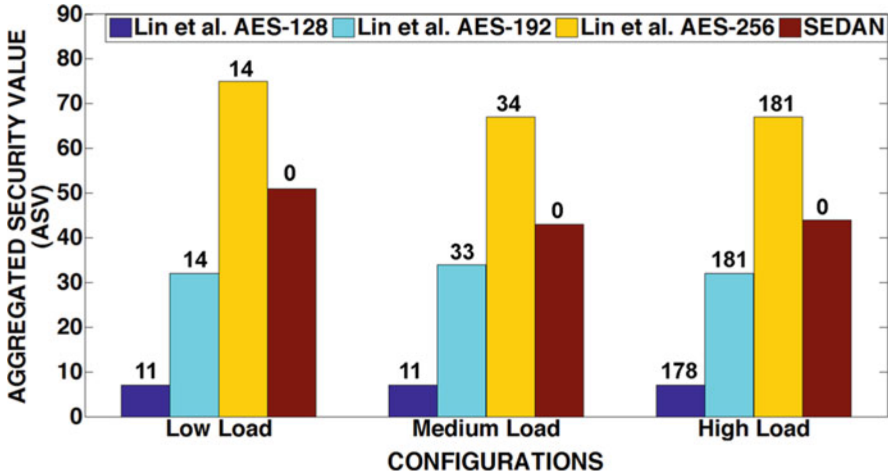


Fig. 9 Comparison of aggregate Security Value (ASV) under each input load configuration for Lin et al. AES-128, AES-192, AES-256 [21], and *SEDAN* (with the number of missed deadlines on top of bars)

requirements. However, it is essential to note that, unlike *SEDAN*, Lin et al. AES-256 has no intelligent key size assignment scheme and assigns all the messages with 256-bit keys irrespective of their ASIL group, which helps in meeting the message security requirements. But this results in increased ECU utilization, which in turn incurs additional latency overheads for messages. Moreover, unlike all three variants of [21], *SEDAN* does not exchange or release keys on an unsecured communication bus. This helps prevent an attacker from gaining knowledge about the current and previously used keys, which provides additional security to the systems. *SEDAN* also does not require frequent key computation at each ECU within a single session, as done in [21], which helps reduce utilization overheads in ECUs when *SEDAN* is employed.

Lastly, the ASV for the three input load cases, with numbers on top of each bar showing the number of messages that missed deadlines, is illustrated in Fig. 9. It can be seen that Lin et al. AES-256 achieves the highest ASV. However, this comes at the cost of multiple missed deadlines. Thus, *SEDAN* is able to satisfy minimum security requirements (i.e., all messages have at least the minimum key size required by the designer) and all real-time deadlines for all messages while providing an ASV value that is higher than that for Lin et al. AES-128 and Lin et al. AES-192.

Thus, *SEDAN* represents a promising framework that can intelligently manage the limited computing resources in vehicles while improving the overall security of the system. Moreover, from Fig. 9 and Table 3, it is evident that *SEDAN* is able to do a better job of balancing security and real-time performance goals by intelligently optimizing key sizes and accurately integrating overheads of security primitives while making task and message scheduling decisions.

6 Conclusions

In this chapter, we presented a novel security framework called *SEDAN* that combines design time schedule optimization with runtime symmetric key management to improve security in time-critical automotive systems without utilizing any additional hardware. We demonstrated the feasibility of our *SEDAN* framework by implementing cryptographic algorithms on real-world processors. Moreover, the experimental results indicate that *SEDAN* is able to reason about security overheads to intelligently adapt security primitives during the message and task scheduling, ultimately ensuring that both security and real-time constraints are met. Such a framework promises to be extremely useful as we move towards connected autonomous vehicles with large attack surfaces by *enabling security to be a first-class design objective* without sacrificing real-time performance objectives.

References

1. Kukkala, V.K., Tunnell, J., Pasricha, S., Bradley, T.: Advanced driver-assistance systems: a path toward autonomous vehicles. IEEE Consumer Electronics Magazine. **7**(5) (2018)
2. Kukkala, V.K., Pasricha, S., Bradley, T.: JAMS: Jitter-Aware Message Scheduling for FlexRay Automotive Networks. In: IEEE/ACM International Symposium on Network-on-Chip (2017)
3. DiDomenico, C., Bair, J., Kukkala, V.K., Tunnell, J., Peyfuss, M., Kraus, M., Ax, J., Lazarri, J., Munin, M., Cooke, C., Christensen, E.: Colorado State University EcoCAR 3 Final Technical Report. In: SAE World Congress Experience (WCX) (April 2019)
4. Kukkala, V.K., Bradley, T., Pasricha, S.: Priority-based Multi-level Monitoring of Signal Integrity in a Distributed Powertrain Control System. in: Proceedingsof IFAC Workshop on Engine and Powertrain Control, Simulation and Modeling (July 2015)
5. Kukkala, V.K., Bradley, T., Pasricha, S.: Uncertainty Analysis and Propagation for an Auxiliary Power Module. In Proceedings of IEEE Transportation Electrification Conference (TEC) (June 2017)
6. Kukkala, V.K., Pasricha, S., Bradley, T.: JAMS-SG: a framework for jitter-aware message scheduling for time-triggered automotive networks. ACM Transactions on Design Automation of Electronic Systems (TODAES). **24**(6) (2019)
7. Tunnell, J., Asher, Z., Pasricha, S., Bradley, T.H.: Towards improving vehicle fuel economy with ADAS. SAE International Journal of Connected and Automated Vehicles. **1**(2) (2018)
8. Tunnell, J., Asher, Z., Pasricha, S., Bradley, T.H.: Towards Improving Vehicle Fuel Economy with ADAS. In Proceedings of SAE World Congress Experience (WCX) (2018)
9. Asher, Z., Tunnell, J., Baker, D.A., Fitzgerald, R.J, Banaei-Kashani, F., Pasricha, S., Bradley, T.H.: Enabling Prediction for Optimal Fuel Economy Vehicle Control. In: Proceedings of SAE World Congress Experience (WCX) (2018)

10. Dey, J., Taylor, W., Pasricha, S.: VESPA: a framework for optimizing heterogeneous sensor placement and orientation for autonomous vehicles. IEEE Consumer Electronics Magazine (CEM). **10**(2) (2021)
11. Kukkala, V., Pasricha, S., Bradley, T.: SEDAN: Security-Aware Design of Time-Critical Automotive Networks. IEEE Transaction on Vehicular Technology (TVT). **69**(8) (2020)
12. Kukkala, V.K., Thiruloga, S.V., Pasricha, S.: INDRA: Intrusion Detection using Recurrent Autoencoders in Automotive Embedded Systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD). **39**(11) (2020)
13. Kukkala, V.K., Thiruloga, S.V., Pasricha, S.: LATTE: LSTM Self-Attention based Anomaly Detection in Embedded Automotive Platforms. ACM Transactions on Embedded Computing Systems (TECS). **20**(5s), Article 67 (2021)
14. Thiruloga, S.V., Kukkala, V.K., Pasricha, S.: TENET: Temporal CNN with Attention for Anomaly Detection in Automotive Cyber-Physical Systems. In: Proceedings of IEEE/ACM Asia & South Pacific Design Automation Conference (ASPDAC) (January 2022)
15. Kukkala, V.K., Thiruloga, S.V., Pasricha, S.: Roadmap for Cybersecurity in Autonomous Vehicles. In IEEE Consumer Electronics Magazine (CEM) (2022)
16. Studnia, I., Nicomette, V., Alata, E., Deswarte, Y., Kaâniche, M., Laarouchi, Y.: Survey of security threats and protection mechanisms in embedded automotive systems. In: IEEE Dependable Systems and Networks Workshop (2013)
17. Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S.: Experimental security analysis of a modern automobile. In: IEEE Symposium on Security and Privacy (2010)
18. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T.: Comprehensive Experimental Analyses of Automotive Attack Surfaces. USENIX Security Symposium (2011)
19. Miller, C., Valasek, C.: Remote Exploitation of an Unaltered Passenger Vehicle. In: Black Hat USA (2015)
20. Izosimov, V., Asvestopoulos, A., Blomkvist, O., Törngren, M.. Security-Aware Development of Cyber-Physical Systems Illustrated with Automotive Case Study. In: IEEE/ACM Design, Automation & Test in Europe (2016)
21. Lin, C.W., Zhu, Q., Sangiovanni-Vincentelli, A.: Security-Aware Mapping for TDMA-Based Real-Time Distributed Systems. In: IEEE/ACM International Conference on Computer-Aided Design (2014)
22. Zalman, R., Mayer, A.: A secure but still safe and low cost automotive communication technique. In: IEEE/ACM Design Automation Conference (2014)
23. Lin, C.W., Zhu, Q., Phung, C., Sangiovanni-Vincentelli, A.: Security-Aware Mapping for CAN-Based Real-Time Distributed Automotive Systems. In: IEEE International Conference on Computer-Aided Design (2013)
24. Lin, C.W., Zhu, Q., Sangiovanni-Vincentelli, A.: Security-aware modeling and efficient mapping for CAN-based real-time distributed automotive systems. IEEE Embedded Systems Letter. **7**(1) (2015)
25. Han, G., Zeng, H., Li, Y., Dou, W.: SAFE: Security Aware FlexRay Scheduling Engine. In: IEEE/ACM Design, Automation & Test in Europe (2014)
26. Hazem, A., Fahmy, H.A.: LCAP- A Lightweight CAN Authentication Protocol for Scheduling in-Vehicle Networks. In: Embedded Security in Cars Conference (2012)
27. Chavez, M.L., Rosete, C.H., Henriquez, F.R.: Achieving Confidentiality Security Service for CAN. In: IEEE International Conference on Electronics, Communications and Computers (2005)
28. Yoshikawa, M., Sugioka, K., Nozaki, Y., Asahi, K.: Secure in-Vehicle Systems against Trojan Attacks. In: IEEE International Conference on Computers and Information Science (2015)
29. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: International Workshop on Cryptographic Hardware and Embedded Systems (2007)

30. Lukasiewycz, M., Mundhenk, P., Steinhorst, S.: Security-aware obfuscated priority assignment for automotive CAN platforms. ACM Transactions on Design Automation on Electrical Systems. **21**(2) (2016)
31. Shreejith, S., Fahmy, S.A.: Security Aware Network Controllers for Next Generation Automotive Embedded Systems. In: IEEE/ACM Design Automation Conference (2015)
32. Perrig, A., Canetti, R., Tygar, J.D., Song, D.: The TESLA Broadcast Authentication Protocol. In: RSA Cryptobytes (2005)
33. Lin, C.W., Yu, H.: Coexistence of Safety and Security in Next-Generation Ethernet-Based Automotive Networks. In: IEEE/ACM Design Automation Conference (2016)
34. Zheng, B., Deng, P., Anguluri, R., Zhu, Q., Pasqualetti, F.: Cross-layer codesign for secure cyber-physical systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. **35**(5) (2016)
35. Lin, C.W., Zheng, B., Zhu, Q., Sangiovanni-Vincentelli, A.: Security-aware design methodology and optimization for automotive systems. ACM Trans. Des. Autom. Electron. Syst. **21**(1) (2015)
36. Mundhenk, P., Steinhorst, S., Lukasiewycz, M., Fahmy, S.A, Chakraborty, S.: Lightweight Authentication for Secure Automotive Networks. In: IEEE/ACM Design, Automation & Test in Europe (2015)
37. Mundhenk, P., Paverd, A., Mrowca, A., Steinhorst, S., Lukasiewycz, M., Fahmy, S.A., Chakraborty, S.: Security in automotive networks: lightweight authentication and authorization. ACM Trans. Des. Autom. Electron. Syst. **22**(2) (2017)
38. Xie, T., Qin, X.: Improving security for periodic tasks in embedded systems through scheduling. ACM Trans. Embed. Comput. Syst. **6**(3) (2007)
39. Lin, M., Xu, L., Yang, L.T., Qin, X., Zheng, N., Wu, Z., Qiu, M.: Static security optimization for real-time systems. IEEE Transaction on Industrial Informatics. **5**(1) (2009)
40. Liang, H., Jagielski, M., Zheng, B., Lin, C.W., Kang, E., Shiraishi, S., Nita-Rotaru, C., Zhu, Q.: Network and System Level Security in Connected Vehicle Applications. In: IEEE/ACM International Conference on Computer-Aided Design (2018)
41. Waszecki, P., Mundhenk, P., Steinhorst, S., Lukasiewycz, M., Karri, R., Chakraborty, S.: Automotive electrical and electronic architecture security via distributed in-vehicle traffic monitoring. IEEE Trans. Comput. Aided Design Integrated Circuits Syst. **36**(11) (2017)
42. Wu, M., Zeng, H., Wang, C., Yu, H.: Safety Guard: Runtime Enforcement for Safety-Critical Cyber-Physical Systems. In: IEEE/ACM Design Automation Conference (2017)
43. Dutta, R.G., Guo, X., Zhang, T., Kwiat, K., Kamhoua, C., Njilla, L., Jin, Y.: Estimation of Safe Sensor Measurements of Autonomous System under Attack. In: IEEE/ACM Design Automation Conference (2017)
44. ISO 26262: Road Vehicles- Functional Safety, ISO Standard (2011)
45. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theory. **22**(6) (1976)
46. Feo, T.A., Resende, M.G.: Greedy randomized adaptive search procedures. J. Glob. Optim. **6**(2) (1995)
47. O'Higgins, B., Diffie, W., Strawczynski, L., De Hoog, R.: Encryption and ISDN- a Natural fit. In: International Switching Symposium (1987)
48. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theory. **31**(4) (1985)
49. Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J.A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B.: Imperfect Forward Secrecy: how Diffie-Hellman Fails in Practice. In: ACM SIGSAC Conference on Computer and Communications Security (2015)
50. Dick, R.P., Rhodes, D.L., Wolf, W.: TGFF: Task Graphs for Free. In: IEEE/ACM International Workshop on Hardware/Software Codesign (1998)
51. FlexRay. FlexRay Communications System Protocol Specification, ver.3.0.1. [Online]. Available: http://www.flexray.com
52. OpenSSL: Cryptography and SSL/TLS toolkit [Online]. Available: http://www.openssl.org/

53. NXP, MPC5775K [Online] www.nxp.com/docs/en/data-sheet/MPC5775KDS.pdf
54. NXP, i.MX 6 [Online] www.nxp.com/docs/en/fact-sheet/IMX6SRSFS.pdf
55. Barker, E., Dang, Q.: Recommendation for Key Management: Application-Specific Key Management Guidance. NIST special publication 800–57 Part 3, Revision 1 (2015)
56. Barker, E., Chen, L., Roginsky, A., Vassilev, A., Davis, R.: Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography. NIST Special Publication 800-56A Revision 3 (2018)
57. National Security Agency. The Case for Elliptic Curve Cryptography. www.nsa.gov/business/programs/elliptic_curve.shtml