SEDAN: Security-Aware Design of Time-Critical Automotive Networks

Vipin Kumar Kukkala, *Student Member, IEEE*, Sudeep Pasricha, *Senior Member, IEEE*, and Thomas Bradley, *Member, IEEE*

***Abstract*— The increasing number of Electronic Control Units (ECUs) in today's vehicles and their greater connectivity with the outside environment has made vehicles more vulnerable to security attacks. Integrating security mechanisms in ECUs has become essential, but incurs overheads, which can delay safety-critical task execution and message transfers. In this work, we introduce a methodology to derive security requirements for tasks and messages in automotive systems based on the ISO 26262 standard. We then propose a framework (*SEDAN*) to increase the security of the system without violating the real-time constraints and security requirements of messages, or ECU utilization limits.**

***Index Terms*—Automotive networks, real-time systems, security**

I. Introduction

MODERN vehicles are examples of complex cyber-physical systems with tens of interconnected Electronic Control Units (ECUs) that control various operations. The advent of Advanced Driver Assistance Systems (ADAS) in vehicles has resulted in an increase in the number of ECUs, which in turn has increased the complexity of the in-vehicle network and the entire automotive system. It is projected that in the near future, improving ADAS effectiveness will require connecting to external systems using vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) protocols [1]. This increased connectivity will make vehicles more vulnerable to sophisticated security attacks. *Ensuring the security of automotive systems will thus become crucial as smart vehicles and self-driving cars become more ubiquitous.*

Some of the most common attacks on vehicles include masquerade, replay, and denial of service (DoS) attacks [2]. In a *masquerade* attack, an attacker ECU 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 network with random messages, thereby preventing the normal operation of valid ECUs. Most of these attacks require access to the in-vehicle network either physically e.g., using on-board diagnostics (OBD-II), or remotely e.g., using LTE or Bluetooth. Some efforts, e.g., [3]-[6], have demonstrated different ways to gain access to the in-vehicle network and send malicious messages to take control of the vehicle. As wireless V2V/V2I transfers become common, vehicle security will be further compromised.

Traditional in-vehicle network protocols such as CAN, FlexRay, etc., do not have any inherent security features to address security concerns such as confidentiality, authentication, and authorization. Hence, preventing unauthorized access to the in-vehicle network requires implementing additional security mechanisms in ECUs. The two most widely used techniques involve symmetric key and asymmetric key encryption. The former uses the same key for both encryption and decryption, while the latter uses a public-private key pair that has a strong mathematical relation. *Both mechanisms incur computational overhead, 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 serious injuries for vehicle occupants. Thus, security mechanisms must be introduced very carefully in vehicles.

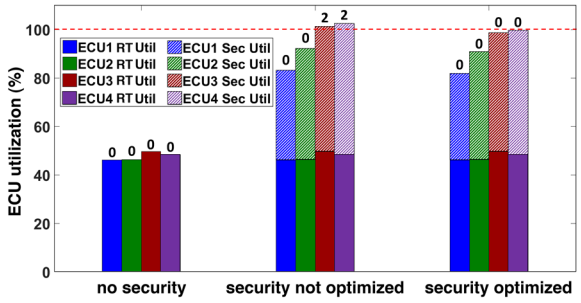


Fig. 1. Motivation for an in-vehicle security framework with low overhead; numbers on top of bars indicate missed application real-time deadlines

Figure 1 illustrates the individual ECU utilizations of a FlexRay-based automotive system consisting of four ECUs running 12 different time-critical automotive applications (each with multiple tasks). Each ECU has a utilization because of the execution of real-time automotive tasks (*RT Util*) and a security utilization because of the execution of security-specific tasks (*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* case has no security mechanism implemented, while the *security not optimized* case uses AES-256 in the ECUs for encryption and decryption of messages. In the latter case, the total utilization for ECUs 3 and 4 (sum of real-time and security task

Manuscript submitted: 19 Jun, 2019; revised October 22, 2019 and February 13, 2020; accepted May 24, 2020. This work was supported in part by grants from the U.S. Department of Energy (DOE) and General Motors.

V. K. Kukkala and S. Pasricha are with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523 USA (e-mail: vipin.kukkala@colostate.edu; sudeep@colostate.edu).

T. Bradley is with the Dept. of Mechanical Engineering, Colorado State University, Fort Collins, CO 80523 USA (e-mail: thb@colostate.edu).

utilizations) exceeds 100% because of the overhead of security-specific encryption/decryption task execution, resulting in missed deadlines for four applications. The *security-optimized* case represents our goal in this work, to integrate all required security mechanisms while keeping utilization below 100% for all ECUs, without any deadline violations.

In this article, we propose a novel framework called *SEDAN* to improve security in time-triggered automotive systems with a minimal overhead on the in-vehicle ECUs. As symmetric key cryptography is less computationally intense than asymmetric key cryptography, we adapt it as part of *SEDAN* to enhance vehicle security; *however note that the use of symmetric encryption is not the main novelty of our work*. *SEDAN* aims to maximize overall system security without violating real-time deadline constraints and per-message security constraints in the system. Our novel contributions in this work are:

- We introduce a novel methodology to derive the security requirements for the different messages in an automotive system based on ISO 26262 and an empirically derived metric to quantify the security of a system;
- We devise a meta-heuristic based key management technique to provide effective security for various message types and ensure ECU utilizations do not exceed 100%;
- We develop an approach for the joint exploration and synthesis of message schedules and security characteristics in TDMA-based automotive systems, and also propose a technique to efficiently map tasks to ECUs while meeting real-time message deadlines and ECU utilization goals;
- We extract network traffic and ECU execution data from a real-world 2016 Chevrolet Camaro vehicle, to compare *SEDAN* with [20] which is the best-known prior work in the area, and also demonstrate the scalability of our work.

II. Related Work

Security in automotive systems was not a major concern until recently. The first full vehicle hack in 2010 was demonstrated in [3] where the authors 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 via the CAN bus. They were also able to perform the same attacks remotely [4]. The researchers in [5] hacked the radio in a 2014 Jeep Cherokee which was connected to both the CAN buses in that vehicle. They used the telematics system in the radio to send remote messages to the vehicle, which were injected into the CAN bus. Most recently in [6], the authors 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 vehicle network. All of these attacks have raised serious concerns about security in automotive systems.

It is hard to prevent unauthorized access to the vehicle bus as the traditional in-vehicle network protocols do not provide any security features. However, one of the popular solutions in the literature to prevent unauthorized access is by authenticating the sender ECU using message authentication codes (MACs). Several works [7]-[11], [13], [14] advocate the use of MACs to improve security in automotive systems. A mixed integer linear programming (MILP) formulation to minimize the overhead for MAC computation and end-to-end application latency in a CAN-based system was proposed in [8], where the same MAC was used for a group of ECUs. This work was extended in [9] to minimize the security risks associated with grouping of different ECUs. In [11] an authentication protocol called LCAP was presented to encrypt messages, with hash functions generating hashed MACs to authenticate ECUs. An RC4 encryption based authentication is implemented in [13] to improve security in CAN-based systems. Another lightweight authentication scheme based on PRESENT [12] is introduced in [14] and evaluated on FPGAs. However, cryptanalysts have demonstrated successful attacks on both RC4 and PRESENT. In [15] a technique is presented to protect a fleet of vehicles by obfuscating CAN bus message identifiers (IDs). *However, all of the above mentioned techniques are designed for event-triggered protocols (such as CAN), and are not applicable to more scalable and sophisticated time-triggered protocols.*

In [7], a lightweight authentication technique is proposed which 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. A device level technique is presented in [17] that uses an enhanced network interface (NI) to authenticate ECUs in the system by making use of hardware-based security modules (HSMs). In [10], FPGAs are used as co-processors for ECUs to handle all the security operations implemented based on the TESLA [16] protocol. But both techniques in [10] and [17] require additional computing resources and many modifications to the existing automotive systems, which is not cost efficient. The authors in [18] proposed a virtual local network (VLAN) based solution for improving security in Ethernet-based automotive systems. They proposed 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 also lacks scalability. In [19] a co-design framework is proposed to improve message response times while meeting the security concerns. However, the authors only consider encrypting a small subset of messages to guarantee control performance, which makes the system vulnerable.

An interesting framework is proposed in [20] that uses a time delayed release of keys approach (adapted from the TESLA protocol [16]) in conjunction with simulated annealing 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 integrates the concept of security with real-time system design from the beginning of the system design phase. This work is extended in [21] by including V2V communication, using dedicated short-range communication (DSRC). A lightweight authentication technique for vehicles called LASAN is proposed in [22] that is based on the Kerberos protocol which is a popular network authentication protocol in the client-server environment. The authors extended this work

in [23] by performing a detailed analysis and comparison with the TESLA [16] protocol. Though the LASAN technique demonstrated superior performance over others, it has stringent requirements for a trusted centralized ECU, which creates a single point of failure. In [24], a security mechanism using different authentication methods was proposed for real time systems. In [25] a group-based security service model is presented with a goal to maximize the combined security of the system. However, as the model ignores time-critical constraints, it cannot be implemented in automotive systems.

An intrusion detection system based on principal component analysis (PCA) is proposed in [26]. An in-vehicle network monitoring system that detects the presence of an attacker by monitoring the increased transmission rates of the messages is proposed in [27]. In [28] the usage of reactive runtime enforcers called safety guards is proposed, to detect the discrepancies between the input data from sensors and output of the controllers. A challenge response authentication approach was proposed in [29] to detect the presence of attackers and estimate the values of the attacked signals. However, this technique requires prior (sometimes proprietary) information about the sensors and also cannot be used for passive safety sensors.

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 leads to over-optimistic results; (ii) they use only one key size for all messages, which ignores heterogeneous security goals in real systems; (iii) they ignore precedence constraints between tasks and messages and; (iv) they consider homogenous single core ECUs which do not accurately represent today's vehicles. In this work, we present the *SEDAN* framework that addresses these limitations of prior work. *SEDAN* improves security in vehicles with time-triggered network protocols (we demonstrate it for the FlexRay protocol, but it can be easily extended to other time-triggered protocols as well, e.g., TTEthernet), while satisfying all designer-imposed security, utilization, and message timing constraints.

III. Problem Definition

A. System and Application Model

We consider a general automotive system where multiple ECUs execute different time-critical applications and are connected using a FlexRay bus-based network, as shown in Fig. 2. Each ECU consists of two major components: a host processor (HP) and a communication controller (CC). The HP is responsible for running automotive and security applications, whereas a CC acts as an interface between the HP and the FlexRay bus, and is responsible for packing message data into frames, sending and receiving messages, and filtering unwanted messages. We consider heterogeneous HPs that have different numbers of cores, which aligns with the state-of-the-art. Note that the heterogeneity is limited to varying the number of homogeneous cores per HP (i.e., multicore parallelism).

Every automotive application consists of both 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 can contain control or data signal values generated by an ECU as a result of task execution. Signals are packed into messages by the HP and are given to the CC to transmit as FlexRay frames on the bus. The automotive applications can be classified as one of two types: *(i)* time-triggered (periodic), or *(ii)* event-triggered (aperiodic). Most safety-critical applications, e.g., anti-lock braking, collision detection, etc., are time-triggered and generate time-triggered messages. Event-triggered messages are generated by maintenance and diagnostic applications. Much like real-time applications across other domains, the execution characteristics of these applications are known at design time. In this work, we focus on time-triggered applications as they have a significant impact on 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 also meeting their real-time deadline constraints. We adapt state-of-the-art standards, 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.

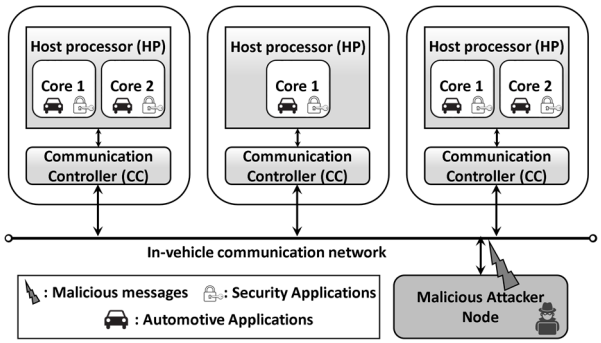


Fig. 2. Overview of our assumed automotive system model

B. FlexRay Communication Protocol

FlexRay is an in-vehicle network protocol designed to support 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 10Mbps. The structure of the FlexRay protocol is shown in Fig. 3. A communication cycle is one complete instance of a communication structure that repeats periodically. Each 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 transmit time-triggered messages. Each static segment slot consists of a header, payload (up to 254 bytes), and trailer segments. The static segment enforces a TDMA media access scheme where each ECU is assigned a particular static segment slot and a cycle number to transmit messages. In contrast, the dynamic segment consists of variable

sized dynamic segment slots that are used to transmit event-triggered messages. A Flexible-TDMA media access scheme is enforced in the dynamic segment where the highest priority ECU gains access to the bus. The symbol window is used for signaling the start of the first communication cycle network and network maintenance. The network idle time segment helps maintain inter-ECU synchronization.

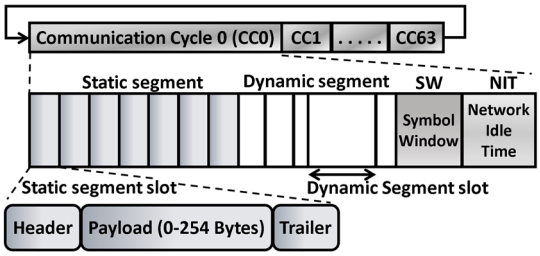


Fig. 3. Structure of the FlexRay in-vehicle network protocol

C. Attack Model

We focus on protecting a vehicle from masquerade and replay attacks as they are most common, hard to detect, and can have a severe impact. The increased external connectivity of modern vehicles creates multiple pathways (attack vectors) to gain access to the in-vehicle network and ECUs. An attacker can employ available attack vectors to gain access to the in-vehicle network and masquerade as an existing ECU or replay valid message transmissions to achieve their goal. In our study, we considered the most common and feasible attack vectors in vehicles, which include connecting to systems that communicate with the outside world (such as infotainment systems), connecting to the OBD-II port, 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. However, handling some of the advanced attack vectors such as core tampering and hardware Trojans requires additional resources and is beyond the scope of our work.

D. Security Model

We aim to achieve the following security objectives in vehicles: *(i)* confidentiality of message data, and *(ii)* the authentication of ECUs. Meeting these objectives can 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. We use AES to achieve confidentiality by encrypting message data using a shared secret key. 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 message encryption 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 IV-F presents the entire encryption/decryption flow in detail. The key size for encrypting/decrypting messages is assigned based on security requirements of a message, which is discussed in section IV-C.

E. Definitions

Our system model has the following inputs:

- Set of heterogeneous (1 or 2 core) ECUs N= {*1, 2, …, N*};
- Set of applications A = {1, 2, …, λ} 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 application $a \in A$;
- Every 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 $\tilde{a}_{q,n}, \tilde{p}_{q,n}, \tilde{d}_{q,n}, \tilde{e}_{q,n}$ denote 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 $\{\bar{a}_{i,n}, \bar{p}_{i,n}, \bar{b}_{i,n}, \bar{d}_{i,n}\}$, where $\bar{a}_{i,n}, \bar{p}_{i,n}, \bar{b}_{i,n}, \bar{d}_{i,n}$ are the arrival time, period, deadline, and data size (in bytes) of signal $s_{i,n}$ respectively;
- After frame packing, every 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 $a_{j,n}, p_{j,n}, d_{j,n}, b_{j,n}, \Delta_{j,n}, \psi_{j,n}$ are the arrival time, period, deadline, data size (in bytes), and minimum security requirement of the message $m_{j,n}$ (see Section IV-C), 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: Our goal is to maximize security (aggregate security value, described in Section IV-D) while synthesizing a design time schedule for time-triggered tasks and messages that meet three types of constraints: *(i)* real-time deadline constraints for tasks and messages in all applications; *(ii)* minimum security constraints for each message in the system, *(iii)* ensure utilization of an ECU does not exceed 100%.

IV. SEDAN Framework: Overview

A high level illustration of our *SEDAN* framework is shown in Fig. 4 with all of the design time steps shown with white boxes and the runtime steps shown with gray boxes. The steps involved in *SEDAN* can be classified into two categories: *(i)* security-related operations to improve system security, and *(ii)* real-time operations to satisfy the application 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 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* first sets up the session keys that will be used for generating keys to perform authenticated encryption

and decryption of messages. A runtime scheduler then makes use of the previously generated keys and the optimal design time schedule to schedule messages online. Each of these steps is discussed in detail in the following subsections.

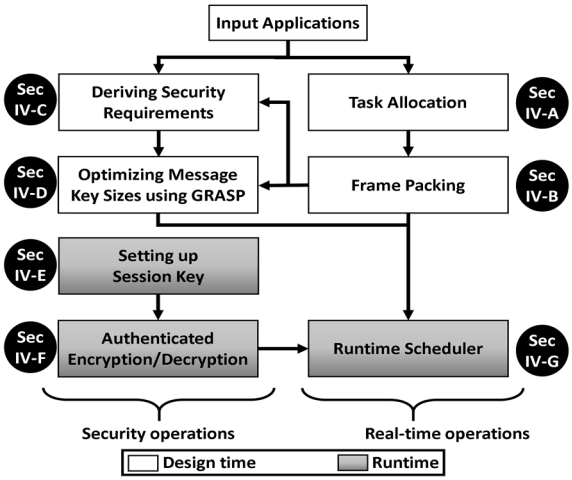


Fig. 4. Overview of the proposed *SEDAN* framework

A. Task Allocation

This is the first step of the *SEDAN* framework and occurs at design time. The goal here is to quickly allocate each task in the system to an available ECU resulting in a balanced real-time utilization across ECUs, which makes the *load-balancing task allocation scheme* a good choice for this step. Note that 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) below. The real-time utilization of any given ECU ($\widetilde{U}_n$) is the sum of the real-time utilizations of the tasks ($\widetilde{U}_{t_{q,n}}$) allocated to that ECU, as shown in (2):

$$\widetilde{U}_{t_q} = \frac{\tilde{e}_q}{\tilde{p}_q} \quad (1)$$

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

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' ($\widetilde{U}_{t_q}$) 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; 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, each task in the system is allocated to an available ECU. After the task allocation step, it is trivial to generate the set of signals S_n for each ECU, based on the precedence constraints of tasks in the application.

Note: As an alternative to a load-balancing task allocation, we also explored an allocation approach with a goal of minimizing total communication volume between ECUs. However, it resulted in non-uniform load allocation across ECUs, which led to violations in ECU utilization constraints after implementing security mechanisms.

B. Frame Packing

Frame packing refers to the grouping of generated signals at each ECU into messages. This is done to maximize bus bandwidth utilization. The set of signals generated by the mapped tasks on each ECU are given as the input to this step and are packed into messages based on three conditions: *(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 cipher is independent of the key size used*. But it is dependent on the input size to 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 section IV-C; a designer can also use a value greater than 49). We adapted a fast *greedy frame packing* heuristic proposed in [31] by integrating the computed payload size definition to generate a set of messages to be transmitted and received, for each ECU.

C. Deriving Security Requirements

We now present a methodology to derive security requirements for each message obtained from the output of the frame packing step. A risk classification scheme defined in ISO 26262 [37] known as the Automotive Safety Integrity Level (ASIL) is adapted to derive security requirements in our work. Four different ASILs: ASIL-A, ASIL-B, ASIL-C, and ASIL-D, are defined in the 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 malicious attackers (hence, higher the safety requirement, higher the security requirement).

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 messages in the system are assigned ASIL groups as follows. Every application is associated with an ASIL group depending on the criticality and tolerance to failure. Each task in that application inherits the same ASIL group 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 assign a security score ($m_{j,n}^{SS}$) to each safety-critical message depending on its assigned key size, as follows: 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* ($m_{j,n}^{AW}$). 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 [33]. Using these metrics that we have defined above, a *security value* ($m_{j,n}^{SV}$) is derived for each message as shown in (3) below. The overall security of the system can then be quantified using an empirically derived metric called *Aggregate Security Value (ASV)*, as shown in (4):

$$m_{j,n}^{SV} = m_{j,n}^{AW} * m_{j,n}^{SS} \quad (3)$$

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

where $\psi_{j,n}$ and R_n are defined in Section III.E. ASV is essentially 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 multiple systems using the same encryption standard. A system with a higher ASV value is more secure than a system with a lower value*.

The second 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 usually expressed as FIT (Failure in Time), which denotes the maximum number of acceptable failures per 1 billion hours of usage. According to specifications, 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 to have less than 10^{-8} failures per hour while ASIL-A applications can have up to 10^{-5} failures per hour. The security requirements for each message in the system are then derived as follows:

- Consider a message $m_{j,n}$ with period $p_{j,n}$ (in milliseconds)
- Number of transmissions of $m_{j,n}$ per second are $(10^3/p_{j,n})$
- 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;
- Thus 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:

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

where constant Q 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 corresponding to that ASIL. Thus, for an ASIL-D message, for the most stringent (smallest) period we observed (=1ms), $\Delta_{j,n}(D)$ = 49 bits (thus this is used in frame packing).

D. Optimizing Message Key Sizes using GRASP

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

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

where b_{size} denotes the block size in bytes and $T_{encr/decr}$ represents the time to encrypt or decrypt one block of data.

As AES is the encryption algorithm used in this work, the above equation can be re-written as:

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

where $T_{AES(X)}$ is the time 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 given ECU ($\overline{U}_n$) is the sum of the security-induced utilizations due to all transmitted and received messages ($\overline{U}_{m_j}$) for that ECU, as shown in (8) below. Thus, for an ECU n, its total utilization (U_n) is the sum of the real-time utilization ($\widetilde{U}_n$) and security-induced utilization ($\overline{U}_n$) as shown in (9):

$$\overline{U}_n = \sum_{j=1}^{R_n} \overline{U}_{m_{j,n}} \quad (8)$$

$$U_n = \widetilde{U}_n + \overline{U}_n \quad (9)$$

To avoid undesirable latency overheads and uncertainty, we always constrain the utilization for any ECU to be below 100%.

A *greedy randomized adaptive search procedure* (*GRASP*) metaheuristic [38] is developed and utilized to achieve this goal. An overview of this approach is illustrated in Fig. 5. The optimal message key size allocation step takes the set of messages from the output of frame packing (section IV-B) and the derived security requirements (section IV-C) as inputs. An *initial solution* is generated by assigning the minimum required key sizes for all the messages based on the derived security requirements. A feasibility check is performed later to determine the feasibility of the initial solution. The feasibility check investigates the *(i)* total ECU utilization (U_n) for all ECUs and *(ii)* number of missed deadlines using a design time scheduler. In this work we adapt the fast design time scheduling heuristic proposed in [31] to synthesize an optimal design time schedule. If there are no utilization violations at any ECU ($U_n \leq 100\%\ \forall$ ECUs) and deadline misses for any message, the initial solution is given as the input to the GRASP metaheuristic. 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 explores various design time schedule configurations (assigning messages and ECUs to FlexRay static segment slots) and message key sizes (that are greater than or equal to the minimum key size requirement for a message) to select a solution that maximizes ASV, with no security violations, real-time deadline misses, and without exceeding 100% utilization for any ECU.

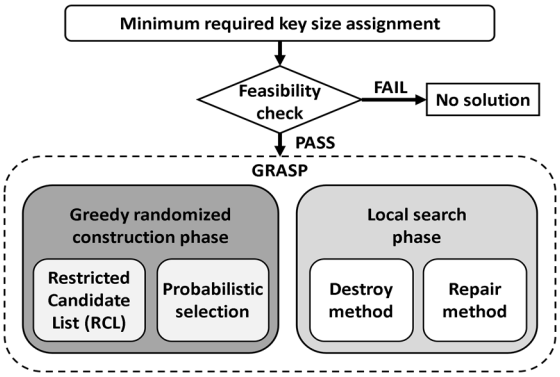


Fig. 5. Overview of optimal message key size allocation step using GRASP

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. At the end, the best overall solution is chosen as the final solution. Two important aspects of the greedy randomized construction phase are the *greedy aspect* and *probabilistic aspect*. The greedy aspect involves generating a Restricted Candidate List (RCL), which consists of best elements that will improve the partial solution (solution within the greedy randomized construction phase). The random selection of an element from the RCL, to be incorporated into the partial solution, is the probabilistic aspect. The solutions generated during the greedy randomized construction phase are not necessarily optimal. Hence a local search phase is used to improve the constructed solution. The local search is an iterative process that uses destroy and repair mechanisms to search for local optimum within a defined neighborhood. If an improved solution is found, then the best solution is updated.

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 where the inputs are: set of nodes (*N*), set of all the messages in the system (*M*), ASV of the system and minimum required message key size assignment (*init_solution*) which is the initial solution given to GRASP to reduce the search space, maximum iterations (*max_iterations*), RCL threshold (α), and a destroy-repair threshold (β). The algorithm begins by assigning the *init_solution* to the *best_solution* (step **1**). GRASP iteratively tries to find a better solution (steps **2**-**8**) until *max_iterations* is reached. In each iteration *greedy_randomized_construction()* (step **3**) generates a local feasible solution (*current_solution*) which is updated using *local_search()* (step **4**). If a better solution is found at the end of local search phase, the *best_solution* is updated (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 with utilization for each ECU in the system below 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. 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.

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 with an aim to maximize the ASV of the system without violating deadline, security, and ECU utilization constraints. 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:  M̃ ← {m ∈ M | m^AG ≠ ASIL-D}
2:  Increment m^SS by 1 ∀ m ∈ M̃ and compute m^SV
3:  Sort M̃ in the increasing order of m^SV
4:  while M̃ ≠ { } do
5:      SV_min = min ({m^SV ∀ m ∈ M̃})
6:      SV_max = max ({m^SV ∀ m ∈ M̃})
7:      RCL ← {m ∈ M̃ | m^SV ≥ SV_min + α * (SV_max − SV_min)}
8:      m̄ ← random element from RCL
9:      Increment the key size of m̄ to the next higher key size
10:     if feasibility_check( ) == fasle do
11:         Revert the key size of m̄ back to its previous key size
12:         Decrement m^SS by 1 for m̄ and compute m^SV
13:     end if
14:     Remove m̄ from M̃
15: end while
16: current_solution ← {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 ($\tilde{M}$) is created in step **1**. The security score of each message (m^{SS}) in $\tilde{M}$ is incremented by one and the security values of the messages (m^{SV}) are updated using (3) (step **2**). In step **3**, $\tilde{M}$ is sorted in the increasing order of m^{SV} and the ties are resolved based on the message period. In steps **4**-**15**, the algorithm tries to find a local solution by incrementing key sizes for some of the messages that would result in no deadline,

security, and ECU utilization violations. In steps **5**, **6** the minimum (SV_{min}) and maximum (SV_{max}) security values are computed respectively. The *RCL* consists of messages in $\widetilde{M}$, that will result in increased ASV when their key size is incremented. The messages whose security value (m^{SV}) is within the interval [SV_{min} + α (SV_{max} - SV_{min}), SV_{max}] are added to the *RCL* in step **7** (this is the greedy aspect of greedy randomized construction). The quality of *RCL* is regulated using an RCL threshold ($\alpha \in [0, 1]$). The threshold (α) controls the amount of greediness and randomness in the algorithm. The case $\alpha = 0$ corresponds to a pure random approach, while $\alpha = 1$ is equivalent to pure greedy approach. In step **8**, a random message ($\overline{m}$) is selected from the *RCL* (probabilistic selection) and its key size is incremented to the next higher key size in step **9** (i.e., 128 → 192 or 192 → 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 [31]. If any of them fails, the *feasibility_check()* returns *false*, and reverts the key size of ($\overline{m}$) back to its previous key size (step **11**) and re-computes m^{SV} of $\overline{m}$ after decrementing the m^{SS} by one (step **12**). Otherwise, the key size increment is left unchanged. The message ($\overline{m}$) is removed from $\widetilde{M}$ and the steps **5-14** are repeated until there are no messages left in $\widetilde{M}$. Lastly, in step **16**, the ASV of the system and the current message key size assignment are assigned to the *current_solution*. The function *calculate_ASV()* is implemented using (4).

2) Local search phase

The local search phase iteratively improves the solution found in the greedy randomized construction phase by investigating a defined neighborhood in the problem space. This is achieved by using *destroy* and *repair* methods which remove a part of the solution and recreate a feasible solution, respectively. In this work, 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 known as the destroy-repair threshold specifies 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.

Algorithm 3 shows the pseudocode of the local search procedure. The *destroy()* function (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. *min_score()* returns the minimal security score demanded by the assigned ASIL group. *repair()* (steps **5-18**) aims to increase the key size for β non ASIL-D messages and computes *local_solution()*. Every time a message needs to be selected for incrementing the key size in *repair()*, the one message that results in maximum increase in the ASV of the system is selected (step **8**). Ties are resolved based on the ASIL group and if multiple messages have the same ASIL group, one message is selected at random.

The local search algorithm iteratively explores the neighborhood around the *current_solution* using *destroy()* and *repair()* to find a better solution (steps **19-29**). In each iteration, β is chosen randomly from [2, β_{max}] (step **20**). *destroy()* is modeled as a stochastic process which is controlled by the key decrease probability (p_{kd}) (steps **21-24**). Lastly, the *current_solution* is updated if a better *local_solution* is found in the repair method (steps **25-28**). In each iteration of GRASP, at the end of local search phase a local optimum is found if there exits one. Otherwise, the solution remains unchanged from the greedy randomized construction phase. Note: 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 and the security induced utilization of the sender and receiver ECUs. The real-time task set induced utilization of the ECUs also remains unchanged, as the time-triggered task execution times do not change with changing message key sizes.

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 ∈ M | m^SS > min_score(m^AG)}
3:      Decrement the key size of a random message (m̄) in M_d
4:  end function

5:  function repair (β, M, N)
6:      M_r = {m ∈ M | m^AG ≠ ASIL-D}
7:      while (β > 0) or (M_r ≠ { }) do
8:          m̄ = {m ∈ M_r | ΔASV is maximum}
9:          Increment the key size of message (m̄)
10:         if feasiblity_check( ) == false do
11:             Revert the key size of (m̄) back to previous key size
12:         else do
13             β = β – 1
14:         end if
15:         Remove (m̄) 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:         β = β – 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, same solution as greedy_randomized_construction( ).

E. Setting up Session Key

This is the first step at runtime in the *SEDAN* framework. It involves settings up session keys required for generating keys that will be used for the encryption and decryption of messages. In general, *using the same key every time for encryption and decryption for the entirety of the vehicle lifetime makes the system highly vulnerable*. Therefore, during runtime, we generate a new key for every session (called session key).

A session is defined as the time duration between the start of a vehicle to turning off the vehicle. As we use symmetric key encryption, all ECUs in the system need the same secret key to function properly. As traditional automotive networks do not have any inbuilt security features, the major challenge here is in exchanging the session keys between ECUs over an unsecure

channel. We adapt the Station-to-Station (STS) key agreement protocol [32] which is based on the famous Diffie-Hellman key exchange method [33] 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 utilize elliptic curve cryptography (ECC) operations as the basis for key agreement instead of RSA, as the former is faster and has lower memory footprint for the same level of security compared to the latter (as discussed in section V-B). The STS protocol with ECC is illustrated in Fig. 6, and discussed below.

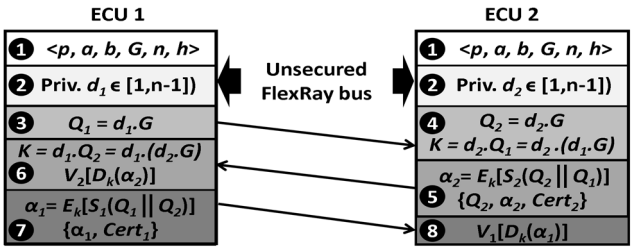


Fig. 6. Steps involved in setting up a session key using the STS protocol using ECC operations over an unsecure FlexRay bus

The approach begins with two ECUs agreeing upon a set of parameters known as domain parameters that define the elliptic curve. In the first step in fig. 6, 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 has an asymmetric key pair used for authentication (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 the next step (step 3), one of the ECUs (e.g., 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. A similar scalar multiplication (between d_2 and G) is performed at ECU2 but the output Q_2 is not sent to ECU1 (step 4). Additionally, ECU2 also computes the scalar multiplication of the private number d_2 and the received output Q_1 resulting in the common secret key K (session key). In the next step (step 5), ECU2 computes the signature ($S_2()$) of the concatenation of Q_2 and Q_1 (represented as $Q_2 \| Q_1$) 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 resulting in the cipher α_2. The scalar multiplication output (Q_2), output cipher (α_2), and the certificate ($Cert_2$) are all transmitted to ECU1 over the unsecured FlexRay bus. The certificate is used to prove the ownership of a public key, which is issued by a trusted certificate authority (CA) and programmed in the ECUs by the manufacturer. It consists of the public key of the owner and signature of the CA. The public key of the CA is used to verify the certificate and extract the public key of the owner. When the ECU1 receives them (in step 6), 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 ($Cert_2$). ECU1 agrees to use the key K for a session only when the verification is successful thereby authenticating ECU2. In the next step (step 7), ECU1 computes the signature ($S_1()$) of the concatenation of Q_1 and Q_2 (represented as $Q_1 \| Q_2$) using its private key of the asymmetric key pair. The output is encrypted using the key K resulting in 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 certificate of ECU1 ($Cert_1$). The 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 key over the bus. Additionally, the STS protocol uses no timestamps and provides a perfect forward secrecy. This session key is used to generate 128-bit, 192-bit and 256-bit keys using a standard AES key schedule at every ECU. 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.

Note: Even if there was an attacker already in the system during the key setup phase, the attacker cannot compute the secret key with the publicly available results due to the discrete logarithm problem [34]. Moreover, the common type of man-in-the-middle attack that has been performed on the simple Diffie-Hellman approach [35] fails with STS as the attacker cannot authenticate successfully. 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.

F. Authenticated Encryption/Decryption

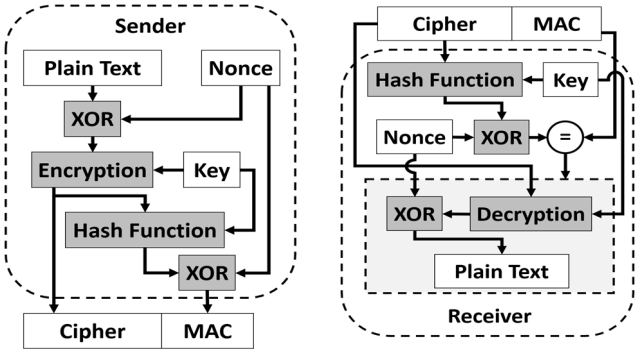
Authenticated encryption refers to simultaneously providing a message with confidentiality and authenticity. This is a well-known technique in the literature and is not a novelty of our work. However, we discuss it here to highlight how *SEDAN* leverages this process to achieve a more secure runtime system.

The keys computed using the session key (section IV.E) and the message to be encrypted are given as the inputs to this step. The authenticated encryption and decryption phases are illustrated in Fig. 7(a) and Fig. 7(b) and discussed next.

The authenticated encryption at the sender ECU begins with an XOR operation between the message data (plain text) and a nonce (random number), and the result is encrypted using AES with the key size allocated (as discussed in section IV.D). The XOR operation is performed to avoid generating the same cipher every time in cases where input data remains unchanged for long durations. Even though protecting the system from side channel attacks is not within the scope of this paper, this simple step could be the first step in preventing information leakage.

The output cipher and the key used for encryption are given to a cryptographic hash function (MD5). The result 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 section IV.C. It is then transmitted with the encrypted message data in the payload section of the FlexRay frame.

At the receiver ECU, the first step is 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 sender ECU is successfully authenticated when the local MAC matches with 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 (plain text).



(a) (b)

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

As discussed in section III-C, we mainly focus on protecting the system from masquerade and replay attacks as they are the most common, hard to detect, and have a severe impact on the system. The system is protected against masquerade or impersonation attacks by authenticating the ECUs in the system using the STS protocol, which also establishes the session keys used for encryption and decryption only after a successful authentication. The attacker fails to authenticate due to the lack of trusted certificates, and hence cannot masquerade as a legitimate ECU. The MAC generated in the authenticated encryption protects the system from replay attacks. During the MAC generation, it is important to XOR the output of the hash function with the nonce as it makes the messages resilient to replay attacks. Whenever an attacker tries to perform 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 results in a MAC mismatch leading to discarding of the message sent by the attacker. Moreover, in the event of a man-in-the middle attack, where the attacker tries to make any changes to the payload content, the MAC comparison fails, thereby protecting the integrity of the messages. Due to the broadcast nature of communication in automotive systems, an ECU or attacker can eavesdrop on the network using any of the attack vectors mentioned in section III-C. However, the attacker would not be able to decrypt the encrypted messages in 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- message confidentiality and integrity, and ECU authenticity (as discussed in section III-D). The other common type of attack in automotive systems is distributed denial of service (*DDoS*) attack. However, we do not focus on this attack as they can be easily detected using a rule based intrusion detection systems (IDS) and can be prevented by designing the network with appropriate gateways and proper bus isolation. Some of the other complex attacks such as side channel attacks, core tampering and hardware trojans require additional resources and are outside the scope of our current work. We plan to investigate these attacks in our future work.

G. Runtime Message Scheduler

Runtime message scheduling is the last step in our framework that takes the unique values of the cipher and MAC generated in the previous step and inserts them into FlexRay frames generated during the frame packing step (section IV-B). Other controls 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 at this point. 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.

V. Experiments

A. Experimental Setup

We evaluated the performance of our proposed *SEDAN* framework by comparing it with [20], which uses simulated annealing to minimize the end-to-end latencies of all in-vehicle messages and uses symmetric key encryption and time-delayed release of keys to improve security in a vehicle system. As [20] does not support variable key sizes, three different variants of [20] are implemented using AES encryption with fixed key sizes of 128, 192 and 256 bits and referred to as 'Lin et al. AES-128', 'Lin et al. AES-192', and 'Lin et al. AES-256' respectively in the results. We generated test cases based on automotive network and ECU computation data extracted from a real-world 2016 Chevrolet Camaro vehicle that we have access to. Directed acyclic graphs (DAGs) were generated using TGFF [30] and annotated with this data. We generated multiple 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. Also, we assume that the deadline for both tasks and messages are equal to their period. For all experiments, a FlexRay 3.0.1 [36] protocol is used with the following network parameters: cycle duration of 5ms with 62 static segment slots, with a slot size of 42 bytes, and 64 communication cycles.

B. Benchmarking Encryption Algorithms

To accurately model the runtime behavior of session key generation and authenticated encryption/decryption we implemented these algorithms in 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 [39]. The algorithms were executed on an ARM Cortex-A9 CPU on a ZedBoard, which has similar specifications compared to many of the state-of-the art ECUs [40], [41].

TABLE I
AES, RSA AND ECC EXECUTION TIMES (ms) 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

The average AES encryption/decryption times with different standard key sizes for one block of data (16 Bytes) are shown in Table I. These values are used at design time to model the latency overhead on each message due to the added security mechanisms. They are also used in computing the response time of the messages and when making scheduling decisions. 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 used to decide between RSA or ECC as the scheme for cryptographic operations in STS protocol. The NIST recommends a keys size of 2048 bits for RSA [42], while NSA recommends a 256 bit key size for SECRET level and 384 bit key size for TOP SECRET level using ECC [43]. Moreover, ECC with 224, 256 and 384 bit key sizes provides a similar security as RSA with 2048, 3072 and 7680 key sizes respectively [44]. In this work, we consider the minimum key sizes based on the above mentioned recommendations. From table I, it can be observed that RSA is faster for verifying signatures (operation performed using public key) and much slower for generating signatures (operation performed using private key). However, on the other hand, ECC is much faster for generating signatures while being relatively slower for verifying signatures. It is important to observe 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 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. Additionally, the ECC values are used in estimating the worst-case time required for setting up a session key, which is 0.24s for a 256-bit key, while an equivalent RSA 2048 takes 3.72s. Thus, ECC is much faster compared to RSA to achieve a similar level of security. Moreover, the key size required to achieve a similar security is much shorter in ECC compared to RSA. The latency associated with computing the hash value using MD5 for one block of data is 2.68µs. Moreover, with the increasing complexity of automotive applications it is important to design the security mechanisms that result in minimal power overhead. Hence, we profiled the security mechanisms used in this paper and presented the power consumption results in Table II. 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. However, the designer can place an upper limit on the number of pre-computed session keys that can be stored to minimize memory overhead. Based on the results shown in Tables I and II, ECC has lower computation and memory overhead compared to RSA for the same level of security. Therefore, in *SEDAN* we authenticate the ECUs in the system and setup session keys using the STS protocol with ECC, instead of RSA. Additionally, we use AES to encrypt and decrypt the messages in the system using the keys derived from the session key.

TABLE II
AES, RSA AND ECC POWER CONSUMPTION 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

C. GRASP parameter selection

To get an efficient solution using the GRASP metaheuristic, it is important to choose 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 for 1000 times using different input test cases. We noticed that the mean solution approached a greedy solution, while the variance approached zero as α tends to 1. In contrast, when α is small and close to zero, the mean solution approaches a random solution with high variance. In order to provide a good quality solution to the local search phase, we chose $\alpha = 0.8$ which leads to a near greedy solution in the presence of relatively large variance. Also, 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, in order 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 with lower ASV compared to the solutions in previous iterations. Moreover, a relative small value for $p_{kd} = 0.3$ is chosen to avoid frequent key size decrements.

D. Response Time Analysis

We tested the *SEDAN* framework and the three variants from

Lin et al. [20] with three different test cases: (1) low input load, in a system with 5 ECUs (3 single-core and 2 dual-core) and 77 tasks that produced 57 (time-triggered) signals; (2) medium input load, in a system with 12 ECUs (9 single-core and 3 dual-core) and 126 tasks with 93 signals; and (3) high input load, in a system with 16 ECUs (12 single-core and 4 dual-core) and 243 tasks with 196 signals. Figures 8(a)-(c) show the average message response time for the low, medium and high input load cases with their deadlines on the x-axis. Response time of a message is its 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. The confidence interval on each bar shows 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.

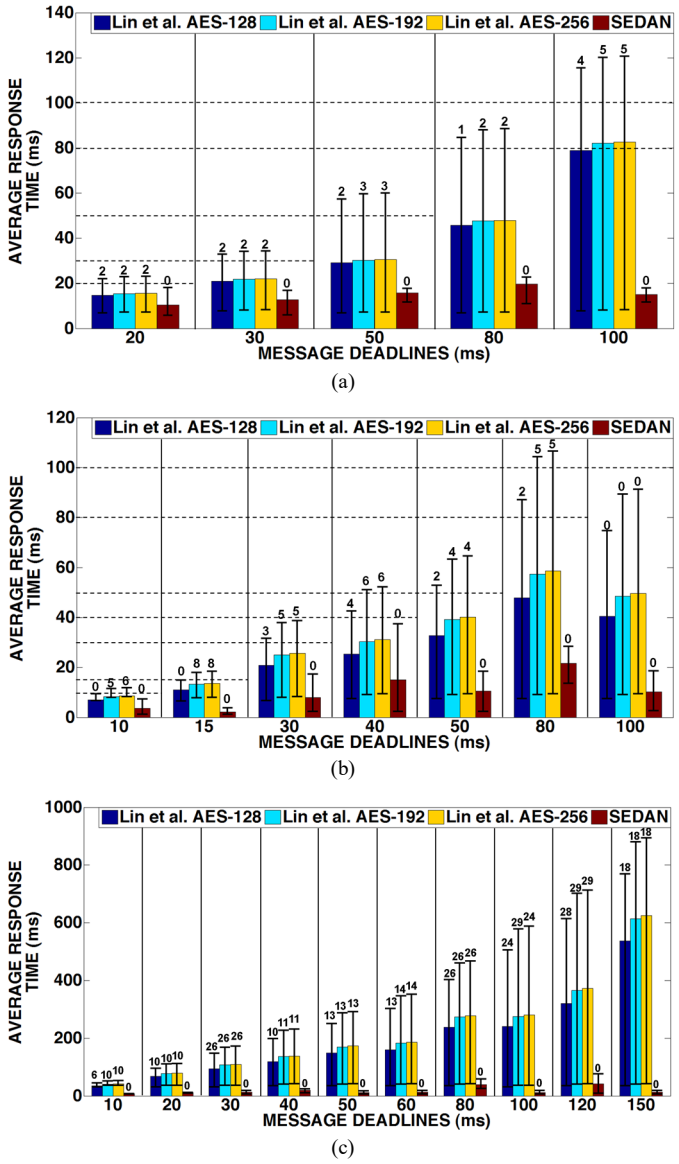


Fig. 8. Average response time of all messages (with number of missed deadlines shown on top of bars) for (a) low; (b) medium and (c) high input application load conditions, for Lin et al. AES-128, AES-192, AES-256 [20]; and *SEDAN*.

It is evident that *SEDAN* outperforms the three variants of [20] and achieves lower average response times for all of the messages in the different 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, compared to *SEDAN*, all the three variants of [20] experience significant authentication delays (time taken from the transmission of the message to decryption of the message), which results in increased response time of the messages. These high authentication delays are because of the time delayed release of keys in all three variants of [20]. Also, the periodic computation of keys in every session at each ECU in all three variants of [20] results in high ECU utilization overhead resulting in increased response time and power consumption. Moreover, the requirement of large message buffers to hold multiple messages for longer durations (due to time-delayed release of keys) further increases the power consumption and response time.

Note that time-triggered safety-critical applications, such as those found in vehicles, are designed to operate at a fixed period under all driving scenarios (e.g., rural, urban city, highway driving) to have a stable vehicular control system. Thus, the bus load is not impacted by the driving scenario and remains practically unchanged as the messages operate at a fixed period. Hence, the results shown in figure 8(a)-(c) would hold true for all driving scenarios.

TABLE III
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

E. Security Analysis

Table III shows the number of security violations in each technique, for the 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 (Section IV-C) for a message are not met. It can be seen that the *SEDAN* and Lin et al. AES-256 are the only techniques that do not violate any security requirements. It is important to note that unlike *SEDAN*, Lin et al. AES-256 has no smart 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 the three variants of [20], *SEDAN* does not exchange or release keys on an unsecured communication bus, thereby preventing an attacker from gaining knowledge about the current and previously used keys. *SEDAN* also does not require frequent key computation at each ECU within a single session, as done in [20], which helps reduce utilization overheads in ECUs when *SEDAN* is used.

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

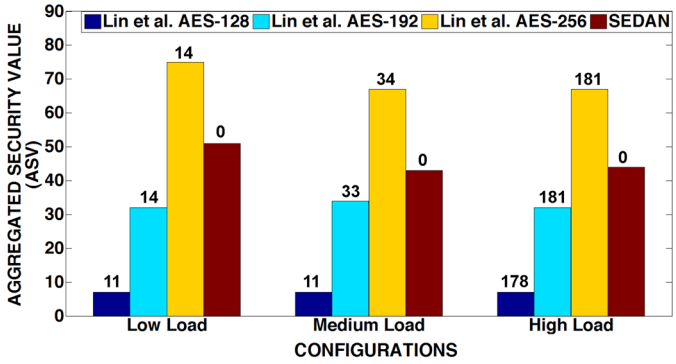


Fig. 9. Aggregate Security Value (ASV) for each input load configuration (with number of missed deadlines on top of bars)

In summary, *SEDAN* represents a promising framework that can intelligently manage the limited computing resources in vehicles while improving the security of the overall system. *SEDAN* is able to do a better job of balancing security and real-time performance goals than [20] as shown in Fig. 9 and Table III. It does so by intelligently optimizing key sizes and accurately integrating overheads of security primitives while making task and message scheduling decisions.

VI. Conclusions

In this work, we presented a novel security framework (*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 the cryptographic algorithms on real processors. Moreover, our experimental results indicate that *SEDAN* is able to reason about security overheads to intelligently adapt security primitives during message and task scheduling, ultimately ensuring that both security and real-time performance goals 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] V. K. Kukkala, J. Tunnell, S. Pasricha, and T. Bradley, "Advanced Driver-Assistance Systems: A Path Toward Autonomous Vehicles," *in IEEE Consumer Electronics Magazine*, vol. 7, no. 5, 2018.

[2] I. Studnia, V. Nicomette, E.Alata, Y. Deswarte, M. Kaâniche, and Y. Laarouchi, "Survey of security threats and protection mechanisms in embedded automotive systems," *in IEEE Dependable Systems and Networks Workshop*, 2013.

[3] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," *in IEEE Symposium on Security and Privacy*, 2010.

[4] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," in *USENIX Security Symposium*, 2011.

[5] C. Miller and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle," in *Black Hat USA*, 2015.

[6] V. Izosimov, A. Asvestopoulos, O. Blomkvist, and M. Törngren, "Security-aware development of cyber-physical systems illustrated with automotive case study," *in IEEE/ACM Design, Automation & Test in Europe*, 2016.

[7] R. Zalman, and A. Mayer, "A secure but still safe and low cost automotive communication technique," *in IEEE/ACM Design Automation Conference*, 2014.

[8] C. W. Lin, Q. Zhu, C. Phung, and A. Sangiovanni-Vincentelli, "Security-aware mapping for CAN-based real-time distributed automotive systems," *in IEEE International Conference on Computer-Aided Design*, 2013.

[9] C. W. Lin, Q. Zhu, and A. Sangiovanni-Vincentelli, "Security-aware modeling and efficient mapping for CAN-based real-time distributed automotive systems," *in IEEE Embedded Systems Letter*, vol. 7, no. 1, 2015.

[10] G. Han, H. Zeng, Y. Li, and W. Dou, "SAFE: Security Aware FlexRay Scheduling Engine," *in IEEE/ACM Design, Automation & Test in Europe*, 2014.

[11] A. Hazem, and H. A. Fahmy, "LCAP- A Lightweight CAN Authentication Protocol for Scheduling In-Vehicle Networks," *in Embedded Security in Cars Conference*, 2012.

[12] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An Ultra-lightweight block cipher," *in International Workshop on Cryptographic Hardware and Embedded Systems*, 2007.

[13] M. L. Chavez, C. H. Rosete, and F. R. Henriquez, "Achieving Confidentiality Security Service for CAN," *in IEEE International Conference on Electronics, Communications and Computers*, 2005.

[14] M. Yoshikawa, K. Sugioka, Y. Nozaki, and K. Asahi, "Secure In-vehicle Systems against Trojan Attacks," *in IEEE International Conference on Computers and Information Science*, 2015.

[15] M. Lukasiewycz, P. Mundhenk, and S. Steinhorst, "Security-aware obfuscated priority assignment for automotive CAN platforms," *in ACM Transactions on Design Automation on Electrical Systems*, vol. 21, no. 2, 2016.

[16] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "The TESLA broadcast authentication protocol," *in RSA Cryptobytes*, 2005.

[17] S. Shreejith, and S. A. Fahmy, "Security aware network controllers for next generation automotive embedded systems," *in IEEE/ACM Design Automation Conference*, 2015.

[18] C. W. Lin, and H. Yu, "Coexistence of safety and security in next-generation ethernet-based automotive networks," *in IEEE/ACM Design Automation Conference*, 2016.

[19] B. Zheng, P. Deng, R. Anguluri, Q. Zhu, and F. Pasqualetti, "Cross-layer codesign for secure cyber-physical systems," *in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, 2016.

[20] C. W. Lin, Q. Zhu, and A. Sangiovanni-Vincentelli, "Security-aware mapping for TDMA-based real-time distributed systems," *in IEEE/ACM International Conference on Computer-Aided Design*, 2014.

[21] C. W. Lin, B. Zheng, Q. Zhu, and A. Sangiovanni-Vincentelli, "Security-aware design methodology and optimization for automotive systems," *in ACM Transactions on Design Automation of Electronic Systems*, vol. 21, no. 1, 2015.

[22] P. Mundhenk, S. Steinhorst, M. Lukasiewycz, S. A. Fahmy, and S. Chakraborty, "Lightweight authentication for secure automotive networks," *in IEEE/ACM Design, Automation & Test in Europe*, 2015.

[23] P. Mundhenk, A. Paverd, A. Mrowca, S. Steinhorst, M. Lukasiewycz, S. A. Fahmy, and S. Chakraborty, "Security in Automotive Networks: Lightweight Authentication and Authorization," *in ACM Transactions on Design Automation of Electronic Systems*, vol. 22, no. 2, 2017.

[24] T. Xie and X. Qin, "Improving security for periodic tasks in embedded systems through scheduling," in *ACM Transactions on Embedded Computing Systems*, vol. 6, no. 3, 2007.

[25] M. Lin, L. Xu, L. T. Yang, X. Qin, N. Zheng, Z. Wu, and M. Qiu, "Static security optimization for real-time systems," in *IEEE Transaction on Industrial Informatics*, vol. 5, no. 1, 2009.

[26] H. Liang, M. Jagielski, B. Zheng, C. W. Lin, E. Kang, S. Shiraishi, C. Nita-Rotaru, and Q. Zhu, "Network and system level security in

connected vehicle applications," in IEEE/ACM *International Conference on Computer-Aided Design*, 2018.
[27] P. Waszecki, P. Mundhenk, S. Steinhorst, M. Lukasiewycz, R. Karri, and S. Chakraborty, "Automotive electrical and electronic architecture security via distributed in-vehicle traffic monitoring," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 11, 2017.
[28] M. Wu, H. Zeng, C. Wang, and H. Yu, "Safety guard: Runtime enforcement for safety-critical cyber-physical systems," in IEEE/ACM *Design Automation Conference*, 2017.
[29] R. G. Dutta, X. Guo, T. Zhang, K. Kwiat, C. Kamhoua, L. Njilla, and Y. Jin, "Estimation of safe sensor measurements of autonomous system under attack," in IEEE/ACM *Design Automation Conference*, 2017.
[30] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," *in IEEE/ACM International Workshop on Hardware/Software Codesign, 1998.*
[31] V.K. Kukkala, S. Pasricha, and T. Bradley, "JAMS: Jitter-Aware Message Scheduling for FlexRay Automotive Networks," *in IEEE/ACM International Symposium on Network-on-Chip*, 2017.
[32] B. O'Higgins, W. Diffie, L. Strawczynski, and R. De Hoog, "Encryption and ISDN- A Natural Fit," *in International Switching Symposium*, 1987.
[33] W. Diffie, and M. Hellman, "New directions in cryptography," in *IEEE Transactions on Information Theory*, vol. 22, no. 6, 1976.
[34] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *in IEEE Transactions on Information Theory*, vol. 31, no. 4, 1985.
[35] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, and B. VanderSloot, "Imperfect forward secrecy: How Diffie-Hellman fails in practice," *in ACM SIGSAC Conference on Computer and Communications Security*, 2015.
[36] FlexRay. FlexRay Communications System Protocol Specification, ver.3.0.1. [Online]. *Available: http://www.flexray.com*
[37] ISO 26262: Road Vehicles- Functional Safety, ISO Standard, 2011.
[38] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *in Journal of Global Optimization*, vol. 6, no. 2, 1995.
[39] OpenSSL: Cryptography and SSL/TLS toolkit [Online]. Available: http://www.openssl.org/
[40] NXP, MPC5775K [Online] www.nxp.com/docs/en/data-sheet/MPC5775KDS.pdf
[41] NXP, i.MX 6 [Online] www.nxp.com/docs/en/fact-sheet/IMX6SRSFS.pdf
[42] E. Barker and Q. Dang, "Recommendation for Key Management: Application-Specific Key Management Guidance." NIST special publication 800-57 Part 3, Revision 1, 2015.
[43] E. Barker, L. Chen, A. Roginsky, A. Vassilev, and R. Davis, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography," NIST special publication 800-56A Revision 3, 2018.
[44] National Security Agency. "The Case for Elliptic Curve Cryptography," www.nsa.gov/business/programs/elliptic_curve.shtml

Vipin Kumar Kukkala (M'13) received his B. Tech degree in electronics and communications engineering from Jawaharlal Nehru Technological University, Hyderabad, India in 2013. He is currently pursuing the Ph.D. degree in electrical engineering at Colorado State University, USA. His current research interests include design of next generation automotive systems, automotive networks, and security in real-time embedded systems.

Sudeep Pasricha (M'02–SM'13) received the B.E. degree in Electronics and Communication Engineering from the Delhi Institute of Technology, India, in 2000 and the Ph.D. degree in computer science from the University of California at Irvine, USA, in 2008. He is currently a Professor and Chair of Computer Engineering in the Department of Electrical and Computer Engineering at Colorado State University. His current research interests include hardware-software co-design for cyber-physical systems, and optimizations for energy, reliability, and security in manycore embedded systems.

Thomas H. Bradley received the B.S. and M.S. degrees from the University of California at Davis, USA, in 2000 and 2003, and the Ph.D. degree from Georgia Institute of Technology, USA in 2008. He is currently a Professor of Mechanical Engineering at Colorado State University, where he is also an Associate Director of Systems Engineering. His current research interests include systems engineering applied to energy problems with an emphasis on environmental assessment, transportation systems, and policy making.