SAFE: Security-Aware FlexRay Scheduling Engine

Gang Han*, Haibo Zeng†, Yaping Li‡, and Wenhua Dou*
*National University of Defense Technology, Changsha, Hunan, China
†McGill University, Montreal, Quebec, Canada ‡University of California, Berkeley, California, USA

***Abstract*—In this paper, we propose SAFE (Security Aware FlexRay scheduling Engine), to provide a problem definition and a design framework for FlexRay static segment schedule to address the new challenge on security. From a high level specification of the application, the architecture and communication middleware are synthesized to satisfy security requirements, in addition to extensibility, costs, and end-to-end latencies. The proposed design process is applied to two industrial case studies consisting of a set of active safety functions and an X-by-wire system respectively.**

I. Introduction

Modern automotive electrical/electronic (E/E) systems are implemented over networked Electronic Control Units (ECUs) communicating via serial buses and gateways. The current design processes, methods, and tools for automotive E/E systems focus on safety (e.g., the recently released ISO 26262 standard), reliability, and cost optimization. However, automotive E/E systems are inherently insecure as little or no security considerations are included in the development process. Communication networks are vulnerable as they enable unauthorized access in a relatively straightforward manner, as all the current communications among ECUs in the vehicle are developed with no authentication in place [14]. *The accessibility to today's cars is rapidly increasing due to cellular connectivity, internet access, remote diagnostics and car service entry points, and soon-to-be-available vehicle-to-vehicle and vehicle-to-infrastructure communications.* Therefore, a large scale cyber-attack on these systems is only a matter of time. Once an attack on one of the ECUs succeeds, it has the potential to propagate across the entire network using the communication infrastructure.

As shown by Koscher et al. [4], in many cases it may be impossible to trace the real malicious cause of malfunctioning. Hence, we are convinced that security must be taken into account in the early phases of the development cycle of automotive E/E systems. This requires enforcing software programming standards that prevent software defects, and, as in this work, implementing security mechanisms that enable the validation of the sender identity to avoid potentially harmful messages across the communication network. We first introduce FlexRay as it is the focus of the paper.

1) FlexRay Protocol: FlexRay is a modern communication protocol for the automotive domain. The bandwidth of FlexRay is assigned according to a time-triggered pattern, with a maximum communication speed of 10 Mbps. Time is divided into *communication cycles*, and each cycle consists of four segments - *Static*, *Dynamic*, *Symbol Window* and *Network Idle Time*. Clock synchronization, embedded in the standard, ensures deterministic communication at no additional cost. The *static* segment of the communication cycle enables the transmission of time-critical messages according to a periodic cycle, in which a time *slot*, of fixed length and in a given position in the cycle, is always reserved for the same node. Each node only needs to know the time slots for its outgoing and incoming communications, or its local scheduling tables. As long as the local tables are consistent, no timing conflicts or interferences arise. Slots that are left free in the (virtual) global table resulting from the composition of the local tables can be used for future extensions. Time protection and isolation from timing faults are guaranteed by bus guardians that prevent node from transmitting outside the allocated time window.

The impact of security vulnerabilities on safety may depend upon the scenarios a specific communication protocol is used for. *We focus on the static segment of FlexRay*, since it is most likely to be used to support safety-critical communications.

2) Related Works: Security is identified as an emerging requirement for embedded systems, and particularly, automotive systems [14]. [4] performs a set of experiments exploring what an attacker could do to a car if he/she can maliciously communicate on the car's internal network.

Several key management protocols have been proposed for time-triggered systems. Time-delayed release of keys (TESLA) [8] assumes loose time synchronization between the sender and the receivers, and uses delayed key disclosure to achieve asymmetric security properties. [10] provides multicast authentication on a per-message basis in time-triggered applications using one truncated message authentication code per receiver. This approach incurs overhead proportional to the number of receivers. [13] discusses the threat model and the authentication protocols for time-triggered architectures, as well as an implementation of the TESLA protocol on a prototype Time-Triggered Ethernet system. [2] incorporates the confidentiality service into Controller Area Network (CAN) based on a lightweight symmetric stream cipher algorithm. [5] proposes a message authentication mechanism for CAN-based automotive architectures by modifying the work in [10].

The design synthesis of the FlexRay static communication schedule has been well studied in the past, see [3] [15] [11] [7] and the references therein. However, *none of them considers the pressing concerns of cyber-security for automotive systems.*

3) Our Contributions: In this paper, we adopt the TESLA protocol, and present a problem definition that includes the execution and synchronization requirements from operations of security mechanisms. We propose divide-and-conquer techniques to divide the problems in two subproblems of manageable sizes and solve them in cascade. We apply the proposed two-step approach to two industrial case studies to demonstrate its effectiveness and efficiency.

II. Security Mechanisms for FlexRay

1) Threat Model: In-vehicle communication needs to be suitably secured as it is possible to directly mounting into a

978-3-9815370-2-4/DATE14/ © 2014 EDAA

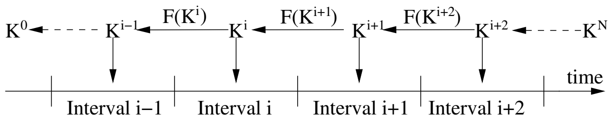


Fig. 1. Key generation and use in time intervals.

car's internal network via the on-broad diagnostics port, even if the car is immune to attacks from external communication. As demonstrated in [4], an attacker can leverage this ability to completely circumvent a broad array of safety-critical systems, e.g., braking individual wheels and stopping the engine. These vulnerabilities can propagate to the whole network due to the broadcast nature of FlexRay. To prevent such an attack, a receiver must be able to verify the message's *authenticity* (it comes from the claimed sender). We assume that the attacker can interrupt an ECU's service but has no access to the signing keys. We focus on *fabrication*, one of the four classes of threats discussed in [9]. Fabrication denotes unauthorized generation of data, e.g., the transmission of a new message with an ID that the attacker is not authorized to. The other three, modification, interception, and interruption, are out of our scope.

To prevent this attack, cryptographical authentication mechanisms are commonly used to verify the message authenticity. Among them, a Message Authentication Code (*MAC*) is a short piece of information appended to a message for verification purposes. A MAC is *symmetric key* based, where *signing* and *verification* use the same secret key shared between the sender and receiver. Both operations generate a fixed-length MAC using the shared key and the (arbitrarily long) message. Another option is *public/asymmetric key* cryptography (e.g., digital signatures), which uses a secret key to sign and a different public key to verify. However, as symmetric key cryptography is computationally more efficient than its public key counterpart, *it is more suitable for resource constrained embedded systems [13].*

2) TESLA Authentication Protocol: Authentication protocols for broadcast communication can adopt several key management strategies. We consider TESLA [8], which uses time to create asymmetry for a symmetric key based approach. Consequently, it enjoys the benefit of computational efficiency while having the asymmetric security property.

Similar to FlexRay, time is divided into intervals of equal duration T_{int}. Before protocol execution, the sender computes a one way key chain of length N as K^0, K^1,..., K^N, where $K^i = F(K^{i+1})$. The function $F(\cdot)$ is required to be *one way* (easy to compute but difficult to invert) and *pseudo-random* (loosely speaking, having random outputs). In interval i, the sender uses the same key K^i to sign all messages sent in this interval, and the key chain is used in reverse order starting with K^0. To bootstrap, a sender signs K^0 with a signature scheme to authenticate K^0, and in turn, all values in the key chain. Figure 1 depicts the assignment of time intervals to a key chain. Key K^i remains secret for the next d intervals (thus K^i will not be disclosed until all messages in interval $i + d - 1$ are transmitted). d is defined as the *key disclosure delay*. The *authentication delay* (between the start of message transmission and the receipt of the released key) is linearly increasing with d and the time interval T_{int} [8] [13].

In TESLA, there are three security operations: *key generation, MAC signature, and MAC verification*. Key generation is realized by iteratively calling the one-way hash function. MAC signature and verification are often done by iteratively applying a keyed hash function (i.e., HMAC, Hash-based MAC) [12]. As in the original TESLA implementation [8], we can use the same function (such as SHA-1 or MD5) for key generation and the underlying hash function for MAC signature/verification, to save code size (for software implementation) or share the same hardware module (for FPGA-based implementation).

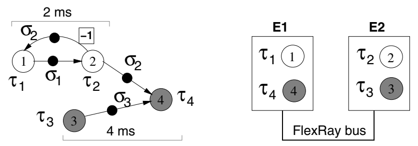


Fig. 2. Application graph and its supporting architecture.

III. Problem Definition

We consider a system application mapped to the supporting architecture, a set of ECUs interconnected by a FlexRay bus. The application is modeled as a *dataflow graph* $\mathcal{G}(\mathbb{V}, \mathbb{E})$. The vertices in the graph represent the *tasks* and the edges represent the data communicated among them (see Fig. 2). A task τ_i is characterized by the tuple $(E_i, T_i, \pi_i, o_i, C_i, D_i)$, where E_i is its *ECU*, T_i its *period*, π_i its priority, o_i its *initial phase*, C_i its *worst case execution time* (WCET), and $D_i \leq T_i$ its *deadline*.

Each task will run an infinite sequence of instances or *jobs*. The *application cycle* or *hyperperiod* H is defined as the least common multiple of the task periods. Each job is considered as an individual scheduling entity and denoted as Γ_i. The scheduling problem consists of planning the execution of jobs and the transmission of signals in the available slots inside H. Jobs can also be denoted with reference to their task, with $\Gamma_{k,j}$ denoting the j-th job of task τ_k. The set of jobs arrived in one application cycle defines the *application instance graph*, as in Fig. 3.

An edge from task τ_h to τ_k represents the input/output connections between them. Each edge may optionally carry a *k-unit delay*, e.g. $\tau_2 \xrightarrow{-1} \tau_1$ in Fig. 2 denotes a one-unit delay between τ_2 and τ_1. If the edge is delivered without delay, the reader must execute after the sender job activated immediately before it, but before the following one; otherwise, it will use the output from the k-th previous job of the sender (see Fig. 3). We consider a *signal* as the communication data between two jobs. Signal σ_i has a given bit width B_i produced by its source SR_i and received by its sinks SN_i.

Besides the application-level tasks, we also consider the set of security operations $\mathbb{A}$ (including the MAC signature as, verification av, and key generation ak). We assume on each ECU the security operations are either implemented in software or FPGA, with known WCETs. The key K can be transmitted along with a message or occupy a dedicated slot. The operations of security mechanisms impose precedence constraints: a message must be signed before sent out, and verified after the reception of itself (including its MAC) and the associated key (see Fig. 3).

The keyed hash functions will output a fixed length of MAC for an arbitrary-length message, e.g., 20 bytes in HMAC-SHA1. This can still add significant overheads to FlexRay communication (7.9% of the bandwidth even with the maximum payload of 254 bytes). We consider a truncation of MAC as long as the desired level of security is ensured. Without better estimation, the generated MAC can be assumed

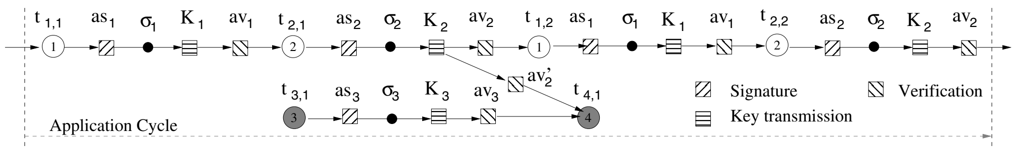


Fig. 3. Application instance graph with authentication operations.

to be completely random [10]. The probability of successfully forging a valid MAC is 2^{-n}, where n is the number of bits in the truncated MAC. Such a probability should be lower than a threshold imposed by, e.g., requirements from ISO 26262. For example, the most stringent safety level, Automotive Safety Integrity Level D, requires that the number of dangerous failures is no higher than 10^{-8} per hour. For a signal transmitted with period $1ms$, it translates to a MAC of no less than 48 bits (or 6 bytes). We assume σ_i is also characterized by a minimum length L_i of its attached MAC corresponding to its security level. Each message is attached with a MAC according to the highest security requirement among the signals packed in it.

A *path* p from τ_i to τ_j is a sequence $[\tau_i, \dots, \tau_j]$ of tasks such that there is a link between any two consecutive tasks. The *latency of path* p is defined as the time interval between the arrival of one job of $\tau_i = \text{src}(p)$ and the completion of the job of $\tau_j = \text{snk}(p)$ that produces a result dependent on it.

Because of the structure of the supply chain and the need to reuse ECUs on different platforms [15], a FlexRay bus configuration (L_{com}, N_{slot}, L_{slot}, B_{slot}) (or a limited list of them) is typically given as an input, where L_{com} is the length of the FlexRay communication cycle, N_{slot} the number of slots in the static segment, L_{slot} the length of the slot in time, and B_{slot} the size of the slot in bits. The system hyperperiod must be an integer multiple of L_{com}.

As part of the end-to-end latency, the authentication delay is linearly increasing with the key disclosure delay d and the time interval T_{int} [8] [13]. We thus use the smallest $d = 1$. Also, we set $T_{int} = L_{\text{com}}$ (as proposed in [13]), such that the time interval schedule of the authentication protocol is aligned with the FlexRay communication cycle. The key chain length N is selected to be an integer multiple of the number of communication cycle in the system hyperperiod.

With a software only implementation, the high computing cost incurred by security mechanism operations can impose a major upgrades of the current ECUs [13]. Alternatively, *we assume that an FPGA co-processor can be added*, which cooperates and shares memory with the main processor in the ECU. Such a solution is also suitable for the current *incremental* automotive design practice, where typically small, evolutionary changes are added without changing the existing platforms. It should be noted that standardized secure hardware extension will most likely be part of many next-generation automotive microcontrollers [1] (e.g., Freescale MPC564xB-C). However, if no such option is available, replacing the current ECUs would require a complete redesign and testing, thus very time-consuming and costly. We assume the hardware cost associated to the authentication modules in FPGA is known. For the software implementation, we assume the MAC signature/verification operations are scheduled as the highest priority jobs on each ECU for its urgency. The key generation can be run as a background task with the lowest priority.

The list of *decision variables* includes: task scheduling, including the task offset selection; signal scheduling, including signal-to-message packing, message-to-slot assignment, ECU-to-slot ownership; and the selection of software/hardware implementation of the security operations and their scheduling.

Algorithm 1: Heuristic for Signal-to-Message Packing

```
1  for each ECU $e$ do
2      MsgList($e$) = $\emptyset$;
3      for each subset $\mathcal{S}$ of signals with the same period from $e$ do
4          $m$={}; // create a new message;
5          Add $m$ to MsgList($e$);
6          for each signal $\sigma_i \in \mathcal{S}$ in decreasing size do
7              if $MsgSize(m) + B_i > B_{slot}$ then
8                  $m$={}; // create a new message;
9                  Add $m$ to MsgList($e$);
10             end
11             $m = m + \{\sigma_i\}$; // pack $\sigma_i$ to message $m$;
12         end
13     end
14 end
```

We propose to consider the following design concerns: the total cost of FPGA units to be added into the system; security level, using the minimum number of bits in the truncated MAC, to enforce the maximum probability of induced failure by attackers; timing performance, using application level end-to-end latency; and extensibility, using the number of unused slots in the FlexRay static segment. We use the total cost of FPGA units as our optimization objective, and assume the other aspects are constraints on the schedule.

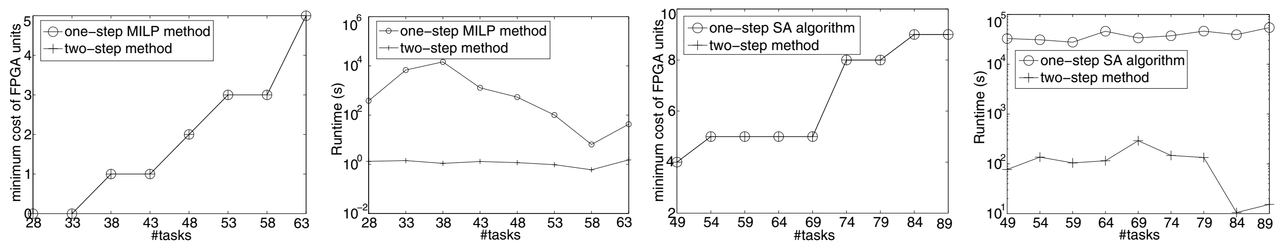
IV. A Divide-and-Conquer Heuristic

A subset of our problem, the packing of signals into slots, has been demonstrated to be NP-complete [6]. A one-step MILP formulation can be built for the FlexRay scheduling problem, but with very high complexity. In the following, we propose a divide-and-conquer heuristic, where signal-to-message packing is found using a heuristic algorithm, followed by an MILP formulation for the other design variables.

Step 1: Signal packing. In the first step, we propose a greedy bin-packing algorithm to pack signals on the same ECU into messages, each of which can be accommodated by one single FlexRay static slot. As presented in Algorithm 1, the signals are first grouped by their periods, i.e., only the signals sent by tasks with the same period can be packed into the same message. Since we are considering a synchronous communication model (to take into consideration the precedence constraints between normal tasks/signals and the ones for security mechanisms), this can possibly avoid the coupling of signal scheduling with different periods and reduce the number of required security operations.

For each subset of signals (signals of the same period), the signals are sorted in a descending order according to their sizes. Then, we pack the signals one by one into a message that is initialized to an empty one. If the size of the resulting message is larger than that of a FlexRay slot, we add message into the packed list and generate a new one. Such a signal packing is done for each ECU in the FlexRay network.

Step 2: Task scheduling, message to slot assignment, and MAC operations. Once the signals are packed into messages, the problem can be simply extended from [15], and we omit the details due to page limit.



(a) #FPGA units for active safety functions

(b) Runtime for active safety functions

(c) #FPGA units for x-by-wire system

(d) Runtime for x-by-wire system

Fig. 4. Minimum cost of required FPGA units and runtime for case studies: active safety functions ((a) and (b)) and x-by-wire ((c) and (d)).

V. Experimental Results

We perform our experiments on a machine with an Intel Core i5 2.4GHz CPU and 4GB memory. We consider two industrial case studies consisting of a set of active safety functions and an X-by-wire system respectively. We assume the FPGA implementation as in [12], where the maximum time to sign/verify a data block (512 bits) is $18.5\mu s$. For software implementation, we measure the execution times of security operations on Freescale MPC564xL, a representative automotive 32-bit microcontroller. A MAC signature operation takes 13749 CPU cycles to finish, or $115\mu s$ if MPC564xL runs at 120MHz, a MAC verification takes 16959 cycles (or $142\mu s$), and a key generation operation uses 3210 cycles (or $27\mu s$).

1) Case Study - Active Safety Functions: The first case study is a set of active safety functions [3], including adaptive cruise control, electric power steering, and traction control. The case study is relatively small, with 6 ECUs, 28 tasks, and 31 signals, thus can be solved with the one-step MILP formulation. Under both approaches, the minimum cost of FPGA units is 0, i.e, all MAC operations can be implemented as software tasks without violating any constraints.

In order to study the performance and the scalability of the proposed approaches, we increase the number of tasks and repeat the experiments. Each new experiment is generated by randomly duplicating a set of 5 tasks and adding them to the case study. Overall, we generate configurations with a number of tasks between 28 and 63. The set of signals and ECUs is the same as in the original case study. The minimum cost of required FPGA units and runtime of each experiment are shown in Figure 4(a) and (b). As in the figure, both approaches achieve the same results for all cases. However, the two-step approach requires a runtime that is 1 to 4 magnitudes smaller. In all experiments, the runtime of the two-step approach is less than 2 seconds, while the one-step approach can be very long (e.g., 14505 seconds for 38 tasks).

2) Case Study - an Automotive X-by-Wire System: The X-by-wire application configuration is obtained from a prototypical X-by-Wire application [15]. The application has 10 ECUs connected by one FlexRay bus. There are a total of 49 tasks, with periods of 1ms and 8 ms respectively, and 132 signals.

The one-step MILP formulation for this case study has roughly 10 million variables and constraints, and is not solvable in reasonable time. For the two-step approach, there are 57772 variables and 176871 constraints in the formulation of the second step. After about 70 seconds, the CPLEX solver finds an optimal solution with 4 FPGA units added.

To verify the quality of the two-step approach, we apply a one-step Simulated Annealing (SA) algorithm to the case study. The SA algorithm considers the following three transition operators: reschedule of a task, by randomly changing the initial phase of a randomly selected task; remapping of a signal, by randomly moving a signal to its neighborhood slot; and reschedule of the key. We take the results of the two-step approach as the initial solution to the SA algorithm. We study the scalability of the two approaches by introducing more tasks in the same way as the previous case study. The number of tasks ranges from 49 to 89. The optimization results and runtimes are shown in Figure 4(c) and (d). In all the cases, the runtime is less than 300 seconds, and more FPGA units are required when more tasks are added. For example, 9 ECUs should be equipped with an FPGA unit when there are a total of 84 tasks. The runtime of SA algorithm is 2 to 3 magnitudes larger than two-step method without finding better results.

VI. Conclusion

In this paper, we propose SAFE, a design framework for FlexRay static segment schedule to address the new challenge on security. We provide divide-and-conquer techniques to divide the problems in two subproblems of manageable sizes and solve them in cascade. We apply the proposed two-step approach to two industrial case studies and demonstrate the effectiveness and efficiency of the proposed two-step approach, as it can achieve the same results as either one-step MILP approach or simulated annealing in a much shorter time.

References

[1] Hersteller Initiative Software. *http://www.automotive-his.de*

[2] M. Chavez, C. Rosete, and F. Henriquez. Achieving Confidentiality Security Service for CAN. In *Proc. CONIELECOMP*, 2005.

[3] S. Ding, N. Murakami, H. Tomiyama, and H. Takada. A ga-based scheduling method for flexray systems. In *Proc. EMSOFT*, 2005.

[4] K. Koscher *et al*. Experimental security analysis of a modern automobile. In *Proc. IEEE Symposium on Security and Privacy*, 2010.

[5] C.-W. Lin *et al*. Cyber-security for the Controller Area Network (CAN) communication protocol. *ASE Science Journal*, 1(2):80–92, 2012.

[6] M. Lukasiewycz, M. Glaß, J. Teich, and P. Milbredt. Flexray schedule optimization of the static segment. In *Proc. CODES+ISSS*, 2009.

[7] M. Lukasiewycz *et al*. Modular scheduling of distributed heterogeneous time-triggered automotive systems. In *Proc. ASPDAC*, 2012.

[8] A. Perrig, R. Canetti, D. Tygar, and D. Song. The TESLA Broadcast Authentication Protocol. *RSA CryptoBytes*, 5:2–13, 2002.

[9] C. P. Pfleeger and S. L. Pfleeger. Security in Computing, 4th Edition. *Prentice Hall PTR*, 2006.

[10] C. Szilagyi and P. Koopman. Flexible multicast authentication for time-triggered embedded control network applications. In *Proc. DSN*, 2009.

[11] B. Tanasa, U. Bordoloi, P. Eles, and Z. Peng. Reliability-Aware Frame Packing for the Static Segment of FlexRay. In *Proc. EMSOFT*, 2011.

[12] M.-Y. Wang *et al*. An HMAC processor with integrated SHA-1 and MD5 algorithms. In *Proc. ASPDAC*, 2004.

[13] A. Wasicek *et al*. Authentication in Time-Triggered Systems Using Time-Delayed Release of Keys. In *Proc. ISORC*, 2011.

[14] M. Wolf, A. Weimerskirch, and C. Paar. Security in automotive bus systems. In *Workshop on Embedded Security in Cars*, 2004.

[15] H. Zeng *et al*. Schedule Optimization of Time-Triggered Systems Communicating Over the FlexRay Static Segment. *IEEE Trans. Industrial Informatics*, 7(1): 1–17, 2011.