Security-Aware Mapping for TDMA-Based Real-Time Distributed Systems

Chung-Wei Lin*, Qi Zhu†, and Alberto Sangiovanni-Vincentelli*
University of California, Berkeley, Berkeley, CA*, University of California, Riverside, Riverside, CA†
E-Mails: cwlin@eecs.berkeley.edu, qzhu@ece.ucr.edu, alberto@berkeley.edu

***Abstract*—Cyber-security has become a critical issue for real-time distributed embedded systems in domains such as automotive, avionics, and industrial automation. However, in many of such systems, tight resource constraints and strict timing requirements make it difficult or even impossible to add security mechanisms *after* the initial design stages. To produce secure and safe systems with desired performance, security must be considered *together* with other objectives at the system level and from the beginning of the design. In this paper, we focus on security-aware design for Time Division Multiple Access (TDMA) based real-time distributed systems. The TDMA-based protocol we consider is an abstraction of many time-triggered protocols that are being adopted in various safety-critical systems for their more predictable timing behavior, such as FlexRay, Time-Triggered Protocol, and Time-Triggered Ethernet. To protect against attacks on TDMA-based real-time distributed systems, we apply a message authentication mechanism with time-delayed release of keys, which provides a good balance between security and computational overhead but needs sophisticated network scheduling to ensure that the increased latencies due to delayed key releases will not violate timing requirements. We propose formulations and an algorithm to optimize the task allocation, priority assignment, network scheduling, and key-release interval length during the mapping process, while meeting *both security and timing requirements*. Experimental results of an automotive case study and a synthetic example show the effectiveness and efficiency of our approach.**

I. Introduction

Cyber-security has become increasingly critical for real-time distributed embedded systems, as they become more networked and more connected with the physical environment, surrounding infrastructures, and other systems. Having more external interfaces increases the possibility of systems being compromised by attackers. Furthermore, the internal architectures of such systems are often designed without sufficient (or any) security consideration. Once part of the system is compromised by attackers, other sub-systems or even the entire system might be affected. For instance, in automotive domain, researchers demonstrated that modern vehicles can be attacked from a variety of interfaces including physical access, short-range wireless, and long-range wireless channels [2], [4], [5]. They also showed that by compromising one Electronic Control Unit (ECU) through those interfaces, the attacker may gain access to other ECUs via communication buses such as Controller Area Network (CAN) buses, and attack safety critical sub-systems such as engine and brake systems [2].

To protect against attacks, not only the various interfaces of the real-time distributed embedded systems need to be secured, the internal architectures need to be hardened with security mechanisms as well. However, such systems usually have tight resource constraints, such as limited communication bandwidths and computational resources, and strict timing requirements for system safety and performance. This makes it very difficult or sometimes impossible to add security mechanisms after the initial design stages without violating the system constraints or impeding the system performance. It is therefore important to address security *together with other design objectives from the beginning of design process and at the system level*, where many important design choices are decided.

There are two main challenges when addressing security together with other design objectives for real-time distributed embedded systems. The first is to apply appropriate security mechanisms, which highly depends on the functional requirements (such as security levels and real-time deadlines) and the architecture specifications (such as system sizes, communication bandwidths, and computational resources). The second is to develop formulations and algorithms to effectively explore the design space for optimizing design objectives while meeting all design constraints (which are often in conflict and require careful trade-offs). In [6], a Mixed Integer Linear Programming (MILP) approach is proposed to address security together with other metrics during the design stage for CAN-based automotive systems.

In this paper, we focus on security-aware design for Time Division Multiple Access (TDMA) based real-time distributed systems. The TDMA-based protocol we consider is an abstraction of many existing protocols, such as the FlexRay [3], the Time-Triggered Protocol [9], and the Time-Triggered Ethernet [10]. It is critically important to address these protocols, as they are being increasingly adopted in various safety-critical systems such as automotive and avionics electronic systems for their more predictable timing behavior. Compared with priority-based networks such as CAN, TDMA-based systems have *fundamental differences* on system modeling (in particular for latency modeling), on security mechanism selection (global time is available for security reasons), on design space (network scheduling is the focus of this work but not a factor in [6]), and on algorithm design (MILP is not suitable for this work because of modeling complexity). Therefore, the approaches for priority-based systems such as [6] do not apply to TDMA-based systems. We need to rethink appropriate security mechanisms and develop a new set of formulations and algorithms to explore the design space.

There are many security mechanisms that can be applied to the TDMA-based protocol. For message authentication, legit senders and receivers usually share keys so that they can use the keys to compute Message Authentication Codes (MACs) and protect against masquerade attacks[1]. In [11], key management strategies are divided into several categories: one

[1]A masquerade attack is the case that an attacker sends a message in which it claims to be a node other than itself.

key for all nodes, one key for each node, private and public keys, and time-delayed release of keys. The one-key-for-all approach is simple but not suitable for distributed systems because it does not protect against masquerade attacks from a node in the group. The one-key-for-each approach protects against such masquerade attacks, but it has limited scalability because the message size increases quickly with the number of nodes in the network. The approach of private and public keys provides higher security level, but its computational overhead is much higher with the usage of asymmetric ciphers, which makes it difficult to be used in resource-limited real-time distributed systems. Compared with these three approaches, the approach of time-delayed release of keys is the most suitable for real-time distributed embedded systems because it provides a good balance between security level and computation and communication overhead [11]. As an example, the Timed Efficient Stream Loss-Tolerant Authentication (TESLA) [7], [8] security mechanism, is based on the time-delayed release of keys.

Despite being a more economical choice, using the time-delayed release of keys for TDMA-based systems still puts significant timing overheads on communication and computation in real-time embedded systems. In particular, the message latencies may significantly increase due to the waiting for key releases, and the end-to-end latencies may violate deadline requirements. For system safety and performance, it is critical to ensure that the usage of such security mechanism will not violate any timing constraint.

In this work, we apply message authentication with the time-delayed release of keys to protect against attacks on a TDMA-based protocol, and develop formulations and an algorithm to explore the design space while meeting *both the security and the timing requirements*. Specifically for the exploration, we optimize the task allocation, priority assignment, network scheduling, and key-release interval length during the mapping process from the functional model to the architectural model, while considering the overhead of the security mechanism and end-to-end deadline constraints. To the best of our knowledge, this is the first work to address both the security and safety constraints during the system level mapping process for the time-delayed release of keys for TDMA-based real-time distributed systems.

We develop an algorithm that combines simulated annealing with a set of efficient optimization heuristics for security-aware mapping. In particular, we propose a network scheduler and a transmission delay analyzer (which outputs exact solutions in a single-switch network) to optimize the network scheduling and analyze the worst-case transmission delay. Our network scheduler and latency analyzer can address synchronous and asynchronous message arrivals, both of which are common scenarios in real-time distributed systems, *e.g.*, they match the Time-Triggered (TT) messages and the Rate-Constrained (RC) messages in the Time-Triggered Ethernet protocol. Experimental results of an industrial case study and a synthetic example show the effectiveness and efficiency of our algorithm, and demonstrate that security must be considered with other metrics during the design stage.

The rest of the paper is organized as follows. Section II introduces the background. Section III presents the design space and our mapping algorithm. Section IV reports the experimental results, and Section V concludes the paper.

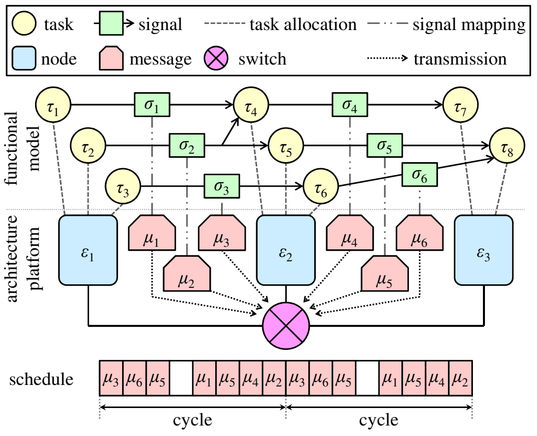

task
signal
task allocation
signal mapping
node
message
switch
transmission
functional model
architecture platform
τ_1 τ_2 τ_3 τ_4 τ_5 τ_6 τ_7 τ_8
σ_1 σ_2 σ_3 σ_4 σ_5 σ_6
ε_1 ε_2 ε_3
μ_1 μ_2 μ_3 μ_4 μ_5 μ_6
schedule
μ_3 μ_6 μ_5 μ_1 μ_5 μ_4 μ_2 μ_3 μ_6 μ_5 μ_1 μ_5 μ_4 μ_2
cycle
cycle

Fig. 1. System model.

II. Background

A. System Model

The mapping problem addressed in this paper is based on the Platform-Based Design paradigm [1], [12], where the functional model and the architecture (physical) platform are initially captured separately and defined through some abstractions. Then, they are brought together through a mapping process, meaning that the functional model is implemented on the architecture platform. Besides, there are objectives and constraints in the mapping process to be optimized and satisfied.

As shown in Figure 1, the functional model is a task graph that consists of a set of tasks, denoted by $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_{|\mathcal{T}|}\}$, and a set of signals, denoted by $\mathcal{S} = \{\sigma_1, \sigma_2, \ldots, \sigma_{|\mathcal{S}|}\}$. Each signal is between a source task and a destination task. Tasks are activated periodically and communicate with each other through signals.

The architecture model is a distributed platform that consists of a set of computation nodes, denoted by $\mathcal{E} = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_{|\mathcal{E}|}\}$, and nodes are assumed to support preemptive priority-based task scheduling. The nodes are connected through a TDMA-based switch (we focus on the single-switch case in this paper, and our formulation can be extended to multi-switches cases). A set of messages is communicated among nodes through the switch, denoted by $\mathcal{M} = \{\mu_1, \mu_2, \ldots, \mu_{|\mathcal{M}|}\}$. The switch uses a TDMA-based model for scheduling, in which each **time slot** in the schedule can be assigned to one message. Several time slots form a **cycle**, and the network switch repeats the same scheduling sequence after each cycle. It is possible that a time slot is empty (not assigned to any message) in a schedule, and it is also possible that there are more than one time slots assigned to the same message in a cycle.

During mapping, the functional model is mapped onto the architecture platform, as shown in Figure 1. Specifically, the tasks are allocated onto nodes with their priorities, and the signals are mapped onto messages and transmitted on the network. Messages are triggered periodically and each message contains the latest values of the signals that are mapped to the message.

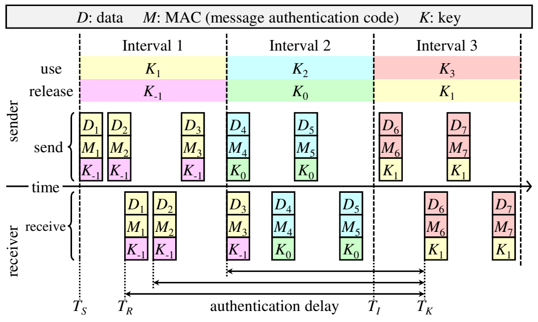


Fig. 2. Time-delayed release of keys. T_S, T_R, and T_K are the sending time, the receiving time, and the key-receiving time of the packet (D_1, M_1, K_{-1}), respectively. T_I is the starting time of Interval 3.

B. Attack Model

We consider three possible types of attacks as in [11], including tapping the port of an existing node, replacing an existing node, and connecting to an empty port of the switch. The time-delayed release of keys approach [7], [8], [11] as shown below is used to prevent these attacks.

C. Time-Delayed Release of Keys

Definition 1: A **packet** is an instance of a message.

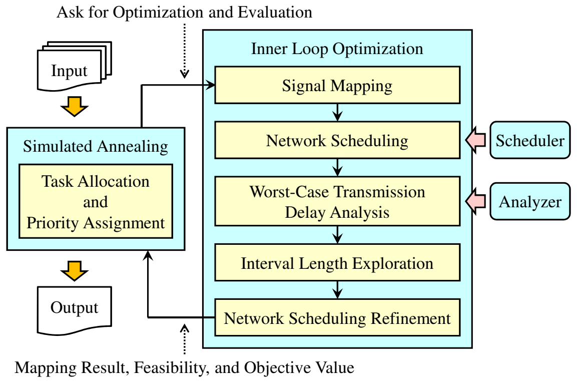
The time-delayed release of keys is a popular approach for message authentication. In this security mechanism, each sender maintains a key chain where the keys in the key chain are computed in a reversed order to provide fault tolerance. Usually, keys in the key chain are not used for computing MACs. Instead, they are used for computing other keys, and those keys are used for computing MACs [7], [8]. A sender maintains **intervals**, and it uses the same key to compute MACs in one interval. When the sender intends to send a packet in an interval, it uses the corresponding key in the interval to compute a MAC and sends the packet including the data, the MAC, and the key *used in several intervals before*. When a receiver receives the packet, it stores the data and the MAC first. Once the receiver receives the corresponding key (which will be released by the sender after several intervals), it can authenticate the packet.

An example of the time-delayed release of keys is shown in Figure 2, where we show the keys used for computing MACs and the released keys. When the sender intends to send data D_1 in Interval 1, it uses key K_1 to compute MAC M_1 and sends (D_1, M_1, K_{-1}), where K_{-1} is the key used in two intervals before. The receiver receives it and stores D_1 and M_1 first. In Interval 3, the sender sends (D_6, M_6, K_1). When the receiver receives the packet, it uses K_1 to authenticate (D_1, M_1, K_{-1}).

Definition 2: Given a packet P, the **sending time** T_S of the packet is the time that its sender sends it. The **receiving time** T_R of the packet is the time that its receiver receives it. The **key-receiving time** T_K of the packet is the time that its receiver receives its corresponding key (for the first time).

The **security requirement of the time-delayed release of keys** is stated in [7], [8]: a packet is safe if its receiving time is before the moment its corresponding key may be released (otherwise masquerade attacks can be conducted), *i.e.*, for each packet,

$$T_R < T_I, \quad (1)$$



Ask for Optimization and Evaluation
Input
Simulated Annealing
Task Allocation and Priority Assignment
Output
Inner Loop Optimization
Signal Mapping
Network Scheduling
Worst-Case Transmission Delay Analysis
Interval Length Exploration
Network Scheduling Refinement
Scheduler
Analyzer
Mapping Result, Feasibility, and Objective Value

Fig. 3. Algorithm flow.

where T_I is the starting time of the interval in which the corresponding key for the packet is released[2].

In the example in Figure 2, since the sender uses K_1 to compute M_3 for the packet (D_3, M_3, K_{-1}) and the packet arrives at the receiver in Interval 2, the sender has to wait until Interval 3 to release K_1.

Definition 3: Given a packet P, the **transmission delay** D_T of the packet is its receiving time minus its sending time, *i.e.*, $D_T = T_R - T_S$. The **authentication delay** D_A of the packet is its key-receiving time minus its receiving time, *i.e.*, $D_A = T_K - T_R$. The **latency** L of the packet is its key-receiving time minus its sending time, *i.e.*, $L = T_K - T_S = D_T + D_A$.

Compared with traditional symmetric ciphers, the time-delayed release of keys has a lower computational overhead because a sender only needs to compute one MAC for each packet. Compared with asymmetric ciphers which have more complex calculation, the time-delayed release of keys also has a much lower computational overhead [11]. However, as shown in Figure 2, it increases the latency of a packet due to the authentication delay. In Section III-D, we will show how the network scheduling plays an important role in reducing the latency of a packet, which is extremely critical for real-time distributed systems.

III. Algorithm

Given a system described in Section II-A, we explore the *design space* of task allocation, priority assignment, signal mapping, network scheduling, and interval length. The end-to-end deadline requirements are set on a set of time-critical functional paths. The worst-case latency of a time-critical path should not be larger than its deadline. The optimization objective is to minimize the summation of the worst-case latencies of all time-critical paths.

A. Overview

Figure 3 shows the flow of our algorithm. It combines simulated annealing with a set of optimization heuristics. In the simulated annealing, the task allocation and the task priority are randomly changed. Every time the task allocation and the task priority are changed, the algorithm calls the inner loop optimization to perform a set of optimization heuristics and

[2]If the synchronization precision is considered, we can add a small positive constant (the precision of time) to the left-hand side.

evaluate the feasibility and the objective value. The inner loop optimization consists of five steps: signal mapping, network scheduling, worst-case transmission delay analysis, interval length exploration, and network scheduling refinement. After these five steps, the inner loop optimization returns the mapping result (signal mapping, network schedule, and interval length). It also returns the feasibility and the objective value, which are used by the simulated annealing to decide whether to keep the changed task allocation and task priority or not. We decide the task allocation and the task priority first in the simulated annealing because they have significant impact on the design constraints and objectives, and also on the possible values for other design variables. The inner loop optimization is called every time the task allocation and the task priority are changed, so it must be very efficient and effective. The details are introduced in the following sections.

B. Task Allocation and Priority Assignment

The initial allocation of a task is assigned based on the task index modulo the number of nodes, *i.e.*, tasks are distributed as evenly as possible. The initial priority of a task is assigned in a greedy fashion—the tasks that appear in more time-critical paths are assigned with higher priorities. During simulated annealing, two random operations may be performed. The first one is to allocate a task to another node, and the second one is to switch the priorities of two tasks. We use a parameter P to control the probability that the first operation is selected in each iteration, while the probability that the second operation is selected is $1 - P$.

To explore the design space more efficiently, we also propose an accelerating method for the simulated annealing. With this accelerating method, tasks are divided into two groups, depending on whether tasks are in time-critical paths or not. Tasks in the first group are in at least one time-critical path, and tasks in the second group are not in any time-critical path. If the first operation is selected, there is another parameter Q to control the probability that a task in the first group is selected. This method can effectively accelerate the simulated annealing because those tasks in the first group play more important roles in the constraint satisfaction and the objective minimization.

C. Signal Mapping

Each signal needs to be mapped onto a message, and we assume each signal is packed into its own message in this work. Without loss of generality, we assume that signal σ_j is mapped to message μ_j, so the period, the length, the source node, and the destination node of a message can be directly decided. What we need to explore is whether a message should be **synchronous** or **asynchronous**.

For a synchronous message, the network knows the time that each packet of the message is sent. For an asynchronous message, the network does not know the time that each packet of the message is sent but knows the period (or pattern) of the message. In our algorithm, if a signal is time-critical (the signal is on at least one time-critical path), then its message is assigned as a synchronous message; otherwise, its message is assigned as an asynchronous message. If a message is synchronous, we also need to decide the time that the first packet of the message is sent. For message μ_j, we assign the time that the first packet of the message is sent as

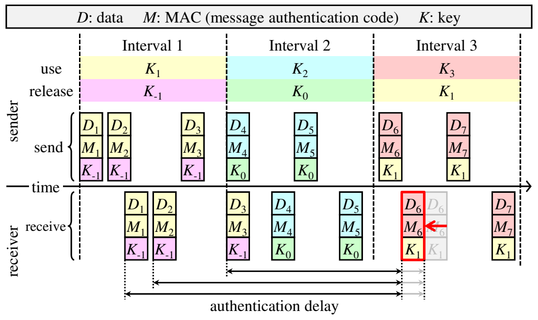


Fig. 4. An approach to reduce the authentication delay.

$j \times L$ modulo T_j^σ, where L is the time length of a time slot and T_j^σ is the period of σ_j. This assignment can lower the probability that the packets of two different messages are sent at the same time. If this happens, one packet will be delayed, and the transmission delay of its message may increase. It should be mentioned that, in Time-Triggered Ethernet [10], a synchronous message matches the Time-Triggered (TT) traffic, while an asynchronous message matches the Rate-Constrained (RC) traffic. If a network only supports synchronous messages or asynchronous messages, then all messages are assigned as synchronous messages or asynchronous messages.

D. Network Scheduling

To satisfy design constraints, the increased latency due to the authentication delays must be considered and reduced during the design stage. We observe that there are three approaches for the network scheduling that can reduce packet latency.

The first approach is that a scheduler may schedule each sender's first packet within an interval so that it can be received earlier. The key-receiving time of a packet P is the receiving time of the first packet P' carrying the corresponding key, so the latency of P is

$$L = T_K - T_S = T'_R - T_S = T'_S + D'_T - T_S, \tag{2}$$

where T'_S, T'_R, and D'_T are the sending time, the receiving time, and the transmission delay of packet P'. The first approach minimizes L by minimizing D'_T. As shown in Figure 4, M_1, M_2, and M_3 are computed by K_1. Because the receiver receives the packet (D_6, M_6, K_1) earlier, it can authenticate the packets (D_1, M_1, K_{-1}), (D_2, M_2, K_{-1}), and (D_3, M_3, K_{-1}) earlier, and their authentication delays and latencies become smaller, compared with the timing illustrated in Figure 2.

The second approach is that a scheduler may try to schedule a packet earlier to ensure that it is received before the end of the interval. As a result, the sender can release keys one interval earlier without violating the security requirement, and the authentication delays and latencies become smaller. The first packet P' carrying the corresponding key is sent after the starting time of the corresponding interval, so

$$T_I < T'_S. \tag{3}$$

Note that T_I is the starting time of the interval in which the corresponding key for the packet is released. The second approach minimizes T_I first so that T'_S can also be minimized (the corresponding key can be released earlier). From Equation (2), the latency of P becomes smaller. As shown in Figure 5,

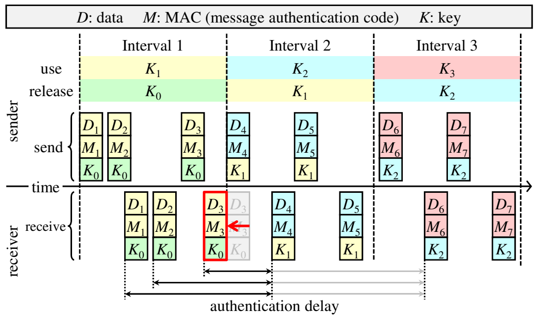


Fig. 5. A more effective approach to reduce the authentication delay.

because the receiver receivers the packet (D_3, M_3, K_0) before the end of Interval 1, the sender can release keys just one interval earlier without violating the security requirement. As a result, the authentication delays and latencies become smaller, compared with the timing illustrated in Figure 2.

The third approach is that a scheduler may minimize the worst-case transmission delay of packets. In some cases, if a scheduler cannot schedule a packet so that it is sent and received in the same interval (for example, a packet is sent just before the end of an interval), the above second approach will not work. However, it provides an insight that, if a scheduler can minimize the worst-case transmission delay of packets, keys can be released earlier. In the third approach, different from the traditional design of the time-delayed release of keys, the intervals of used keys and released keys are not aligned. As shown in Figure 6, given the worst-case transmission delay D_T, a key is released D_T time units after the end of the interval in which the key is used for computing MACs. As a result, the authentication delay and the latency are also reduced, compared with the timing illustrated in Figure 2. Combining Definition 2 and Equations (1) and (3), we get

$$T_R = T_S + D_T < T_I < T'_S. \tag{4}$$

The third approach minimizes D_T first so that T_I and then T'_S can also be minimized. From Equation (2), the latency of P becomes smaller.

We will reduce the latency of a packet through the above three approaches. In this step, we minimize the worst-case transmission delay of packets so that keys can be released earlier (the second and the third approaches[3]). We will then try to release keys earlier in the network scheduling refinement step in Section III-G (the first approach).

Specifically, in this step, we first assign priorities to messages. A message whose corresponding signal appears more times in the time-critical paths is given a higher priority. We then schedule messages one-by-one according to their priorities. If message μ_j is synchronous, we schedule time slots to it "as early as possible". In other words, we schedule the first time slot after the arrival of a packet to the message. If message μ_j is asynchronous, we first compute the number of time slots that we plan to allocate to the message in a cycle. For asynchronous message μ_j, the number of time slots in a cycle is $\left\lceil R \times \frac{N \times L}{T_j^\sigma} \right\rceil$, where R is a parameter larger than or equal to 1, N is the number of time slots in a cycle, L is the time length of a time slot, and T_j^σ is the period of σ_j.

[3]Note that the second approach can be regarded as a special case of the third approach. Both of them try to minimize transmission delays of packets.

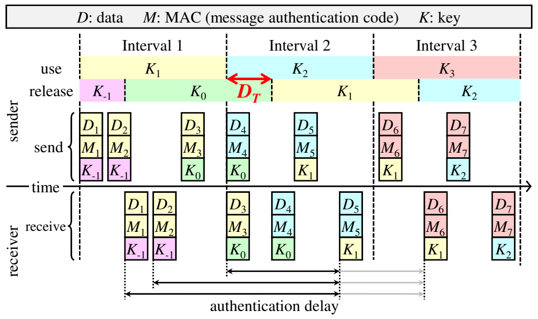


Fig. 6. Given the worst-case transmission delay D_T, an approach to reduce the authentication delay.

After computing the number of time slots, we schedule time slots to the asynchronous message "as evenly as possible". It is possible that a time slot has been used (occupied) by a higher-priority message. In this case, we schedule the next empty time slot to the message. It is very important that we schedule synchronous messages as early as possible and asynchronous messages as evenly as possible, as they are the optimal strategies for each of them (which will be further demonstrated in the next section).

One thing that should be emphasized is the choice of R value. A large R means that denser time slots are scheduled to an asynchronous message, and the worst-case transmission delay of the message may decrease. If we do not consider the time-delayed release of keys, the decreasing of the worst-case transmission delay of an asynchronous message has no effect on the objective value because only non-time-critical signals are mapped to an asynchronous message. This is also a reason that a traditional scheduler may not be suitable for this problem. On the contrary, when we consider the time-delayed release of keys in this case, the decreasing of the worst-case transmission delay of an asynchronous message enables its sender to release keys earlier, so the worst-case latencies of synchronous messages and the objective value can become smaller. Therefore, we increase the parameter R in our case. Specifically, if $R = 1$ and the network utilization rate (the ratio of the number of scheduled time slots to the number of total time slots) is smaller than a pre-assigned value U, we increase R so that the network utilization rate reaches U.

E. Worst-Case Transmission Delay Analysis

Besides network scheduling, an accurate analyzer for computing the worst-case transmission delay is also very important. Given the worst-case transmission delay D_T, a key can be released D_T time units after the end of the interval in which the key is used for computing MACs. If the analyzer underestimates the worst-case transmission delay, the security requirement may be violated because keys may be released too early. If the analyzer overestimates the worst-case transmission delay (*i.e.*, being too pessimistic), minimizing the worst-case transmission delay may not be effective. To compute the worst-case transmission delay, we first define the packet arrival pattern and the schedule pattern of a message as follows.

Definition 4: A **packet arrival pattern** A is defined by m, p and $a_1, a_2, \ldots, a_m$, where the arriving times of packets are $a_1, a_2, a_3, \ldots, a_m$, and the pattern repeats with a period p ($a_i < p$ for all i, $1 \le i \le m$).

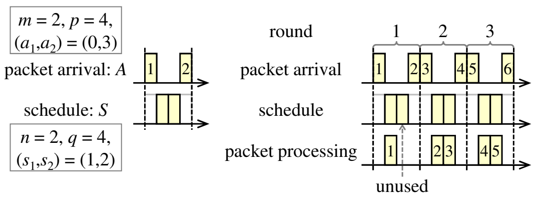


Fig. 7. An example for synchronous messages with tree rounds. The second packet (#2, #4, or #6) of each round is an unscheduled packet after its corresponding round, and the second time slot of the first round is an unused time slot. The second round and the third round have the same packet processing pattern.

Definition 5: A **schedule pattern** S is defined by n, q and $s_1, s_2, \ldots, s_n$, where the starting times of time slots are $s_1, s_2, s_3, \ldots, s_n$, and the pattern repeats with a period q ($s_i < q$ for all i, $1 \le i \le n$).

The problem here is: "given the packet arrival pattern A and the schedule pattern S of a message, what is the worst-case transmission delay of the packet arrival pattern?" We will discuss synchronous messages and asynchronous messages in the following sections.

1) Synchronous Message:

Definition 6: A **round** is a time period whose length is the least common multiple of p and q. A packet is **unscheduled** after a round if it is not assigned to any time slot after the round; otherwise, it is **scheduled**. A time slot is **unused** if no packet is assigned to it; otherwise, it is **used**.

In Figure 7, there are three rounds. The second packet of each round is an unscheduled packet after its corresponding round. The second time slot of the first round is an unused time slot. Given a synchronous packet arrival pattern A and its schedule pattern S, we only need to consider two rounds for the worst-case transmission delay of the packet arrival pattern. In the analysis, we start from the first packet and assign each packet to the first unused time slot after the arrival of the packet.

Theorem 1: We only need to consider two rounds for the worst-case transmission delay of the packet arrival pattern.

Proof: We claim that the numbers of unscheduled packets after the first round and the second round are the same. Therefore, the pattern of the second round is the same as that of any following round. We will prove that the number of unscheduled packets does not decrease or increase after the second round. For the non-decreasing part, it is because the second round is more difficult (with some unscheduled packets after the first round) than the first round. For the non-increasing part, we first assume that $\frac{m}{p} \le \frac{n}{q}$, *i.e.*, the number of packets in a round is never larger than the number of time slots in a round; otherwise, the algorithm returns infinity directly. Given this assumption, after the first round, the number of unscheduled packets is never larger than the number of unused time slots. Accordingly, in the second round, the repeated time slots of those unused time slots in the first round are sufficient for those unscheduled packets in the first round. Besides, the repeated time slots of those used time slots in the first round are still sufficient for the repeated packets of those scheduled packets in the first round. Therefore, the number of unscheduled packets does not increase. Combining the two parts, the numbers of unscheduled packets after the first round and the second round are the same, so we only need to consider the first two rounds. ∎

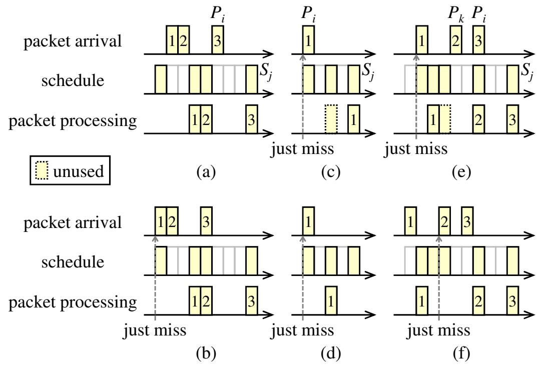


Fig. 8. For asynchronous messages, if the worst-case transmission delay happens when packet P_i is assigned to time slot S_j, then (a–b) one of P_i itself and the packets arriving before P_i must just miss a time slot, and (c–f) there must be no unused time slot between the arriving time of the packet just missing a time slot and the starting time of S_j.

The theorem also implies that an unscheduled packet after the second round does not affect the result. An example is shown in Figure 7, where the second round and the third round have the same packet processing pattern. It also shows that we need to consider at least two rounds for the worst-case transmission delay of the packet arrival pattern.

2) Asynchronous Message:

Definition 7: A packet **just misses** a time slot if the starting time of the time slot is ϵ time unit earlier than the arriving time of the packet where $\epsilon \to 0$.

Theorem 2: If the worst-case transmission delay happens when packet P_i is assigned to time slot S_j, then (1) one of P_i itself and the packets arriving before P_i must have just missed a time slot, and (2) there must be no unused time slot between the arriving time of the packet just missing a time slot and the starting time of S_j.

The proof is omitted due to the limitation of space. Figure 8(a) is an example of the first part of the theorem. We can shift all packets so that they arrive earlier as shown in Figure 8(b), and the transmission delays of them become larger. Figure 8(c) is an example of the first case of the second part of the theorem, where P_i should be assigned to the unused time slot as shown in Figure 8(d). Figure 8(e) is an example of the second case of the second part of the theorem. We can shift all packets so that they arrive earlier as shown in Figure 8(f), and the transmission delays of the packets arriving between P_k and P_i become larger.

Given Theorem 2, we only need to consider a finite number of different alignments of the packet arrival pattern and the schedule pattern—they are the cases that at least one packet just misses a time slot. Here, we assume that the patterns have been repeated (duplicated) enough to A' and S' with lengths equal to the least common multiple of p and q. The worst-case transmission delay of the packet arrival pattern is

$$\max_{1 \le i \le m, 1 \le j \le n, 1 \le k \le m} \left((s'_{j+k} - s'_j) - (a'_{i+k-1} - a'_i) \right), \quad (5)$$

where $a'_1, a'_2, \ldots, a'_m$ are arriving times of packets and $s'_1, s'_2, \ldots, s'_n$ are the starting times of time slots. For each (i, j, k), the equation computes the transmission delay of the $(i+k-1)$-th packet under the case that the i-th packet just misses the j-th time slot[4]. The equation can be written as

$$\max_{1 \leq k \leq m} \left(\max_{1 \leq j \leq n} \left(s'_{j+k} - s'_j \right) - \min_{1 \leq i \leq m} \left(a'_{i+k-1} - a'_i \right) \right). \quad (6)$$

As a result, we can reduce the complexity of the computation from $O(m^2 n)$ to $O(mn + m^2)$.

F. Interval Length Exploration

The latency of a packet highly depends on the length of an interval. A shorter interval results in a smaller latency of a packet, but, if the number of keys in a key chain is a constant (the memory size for storing keys), a shorter interval means that a sender needs to recompute a key chain more frequently, which increases the computational overhead. After we decide the network scheduling and compute the worst-case transmission delay, we explore the interval length of each node. For each node, there is a list of possible interval lengths, we start from the shortest one and check if the task computing a new key chain can meet its deadline which is the number of keys times the length of an interval. If it cannot meet its deadline, we will check the next possible interval length.

G. Network Scheduling Refinement

To further minimize the latency of a packet, we want keys to be released as early as possible without violating security requirement. After the interval length of each node is decided, a key can be released with the first packet in an interval. Therefore, for each sender, we check if there is any empty time slot between the starting time of a releasing interval of a sender and the first time slot assigned to a message sent by the sender. If there is such a time slot, we assign the time slot to the sender so that the sender can use the time slot to release a key. It is after the starting time of a releasing interval of a sender to satisfy the security requirement; and it is before the first time slot assigned to a message sent by the sender, so the key is released earlier, and the latency of a packet can be reduced. Furthermore, because the time slot is originally empty, it will not increase the latencies of other packets.

After this step, we can compute the worst-case latencies of time-critical paths and the objective value and check the feasibility. The mapping result, the feasibility and the objective value are returned to the simulated annealing in the outer loop.

IV. Experimental Results

We obtained the industrial test case used in [6]. The test case is an automotive system which supports advanced distributed functions with end-to-end computations collecting data from 360-degree sensors to the actuators, consisting of the throttle, brake and steering subsystems and of advanced Human-Machine Interface devices. The architecture platform consists of 9 nodes (ECUs) which are assumed to be connected through a TDMA network (an abstraction of the Time-Triggered Ethernet or the FlexRay). The network parameters are set according to [10], while the computation time for a MAC or a key chain is scaled from [11]. The functional model consists of 41 tasks and 83 signals, and 171 paths are selected with deadlines 300 *msec* or 100 *msec*. The algorithm is implemented in C/C++. The experiments on the mapping problem were run on a 2.5-GHz processor with 4GB RAM.

We compare the results of a non-security-aware mapping approach and our security-aware algorithm. The non-security-aware mapping approach is based on the same simulated annealing core, but it does not consider any effect from the time-delayed release of keys during the mapping, *i.e.*, the latency of a packet is exactly its transmission delay because it does not need to wait a key. *After* the mapping is decided, we then apply the time-delayed release of keys on the design (in this step, we explore the interval of each node). On the contrary, our security-aware algorithm considers the overheads of the time-delayed release of keys *from the beginning* and solve the security-aware mapping problem as mentioned in Section III. There are two optional optimization techniques which result in four possible combinations of the two. The first optimization technique (OPT1) is to increase R ($R > 1$) in the network scheduling, as mentioned in Section III-D. The second optimization technique (OPT2) is to use empty time slots and release keys earlier in the network refinement, as mentioned in Section III-G. For the simulated annealing (SA), the parameters P and Q are both set to 0.9, where Q is for the accelerated method (Accelerated SA) as mentioned in Section III-B.

The results are listed in Table I where all of them are the averages of 10 runs. To have a fair comparison, we let all settings run with the same number (15,000) of iterations in the simulated annealing. The objective is the summation of the worst-case latencies of all time-critical paths. The non-security-aware mapping approach cannot find a feasible solution because applying the time-delayed release of keys *after* the mapping is decided makes some time-critical paths miss their deadlines. On the contrary, our security-aware mapping algorithm can find a feasible solution with the objective value 22,256 *msec*. If we consider the objective value of the non-security-aware mapping approach (although it is not feasible), it is 25,007 *msec*, larger than that of our security-aware mapping algorithm. This is because our security-aware mapping algorithm tends to minimize the transmission delays of asynchronous messages, which enable their senders to release keys earlier and lead to a smaller objective value. Besides, our security-aware mapping algorithm has smaller runtimes because it has a stronger force to allocate the source and target tasks of a signal to the same node and leave fewer messages on the network, which makes the runtime of network scheduling smaller.

With the two optimization techniques, the objective values are reduced to 21,415 *msec* and 21,329 *msec*, respectively. This is because, the OPT1 tries to further minimize the transmission delay of an asynchronous message, and the OPT2 tries to use empty time slots and release keys earlier. These two techniques are exactly the third and the first approaches in Section III-D. By combining both of them, the objective value is further reduced to 20,853 *msec*. If the accelerated SA is applied, the design space can be explored more effectively. With the same number of iterations, the accelerated SA can find smaller objective values, compared with the basic SA. Especially, it can find a feasible solution earlier, which is because it focuses more on those tasks which play more important roles in the constraint satisfaction and the objective minimization. The

[4]The concept here is that the densest part of the packet arrival pattern is served by the least dense part of the schedule pattern.

TABLE I. The comparison between a non-security-aware mapping approach (its objectives are reported, but its solutions are infeasible) and our security-aware mapping algorithm for the industrial test case, where there are two optional optimization techniques resulting in four combinations. The objective is the summation of the worst-case latencies of all time-critical paths. The feasible time is the time it takes to find the first feasible solution.

		Non-Security-Aware Mapping	Security-Aware Mapping			
			No OPT1/OPT2	OPT1 Only	OPT2 Only	OPT1+OPT2
Basic SA	Objective (*msec*)	25,006.665	22,256.048	21,414.690	21,329.322	20,853.017
	Runtime (*s*)	56.435	47.046	50.725	47.767	47.862
	Feasible Time (*s*)	×	3.576	3.652	3.439	4.818
Accelerated SA	Objective (*msec*)	23,475.727	21,156.529	20,581.321	21,010.984	20,236.140
	Runtime (*s*)	55.695	50.441	47.963	44.070	48.065
	Feasible Time (*s*)	×	2.959	2.910	1.733	1.826

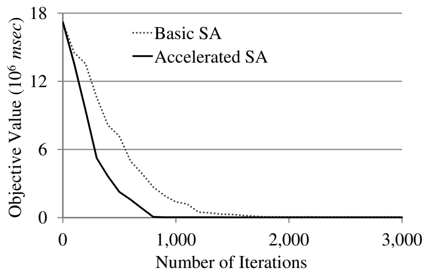


Fig. 9. The converging behaviors of the basic SA and the accelerated SA for the industrial test case. The x-axis represents the number of iterations of the simulated annealing, and the y-axis represents the objective value (10^6 *msec*) where each constraint violation contributes 100,000 to the objective value.

converging behaviors of the basic SA and the accelerated SA are illustrated in Figure 9. The accelerated SA converges faster than the basic SA. From the experimental results, it is difficult to tell whether OPT1 or OPT2 is more effective, but having both of them outperforms each individual. Besides, all experiments with our algorithm (using different combinations) are done within one minute. Even only considering task allocation and task priority assignment, an MILP-based approach similar to that in [6] cannot find a feasible solution in one hour. This shows the efficiency of our algorithm.

We also generate a large random test case including 500 tasks, 1,000 signals, 50 nodes, and 100 time-critical paths. We apply our security-aware mapping algorithm with the accelerated SA and the two optimization techniques (OPT1 and OPT2). In addition to algorithm scalability, we are also interested in the impact of resource availability (specifically the network utilization[5]) on the system performance and feasibility. Table II lists the objectives and runtimes under different signal periods and therefore different network utilizations (all settings are run with the same number of iterations). First of all, we can see that the algorithm scales well with the problem size because of the efficient inner loop optimization heuristics. Furthermore, as the utilization increases, the objectives and the runtimes increases dramatically. This shows the significant impact of resource availability on the system performance and feasibility when security is taken into consideration, and therefore further demonstrates the need to address security together with other metrics in an integrated formulation.

V. Conclusion

In this paper, we present formulations and an algorithm to address security together with other design objectives during the mapping stage for TDMA-based real-time distributed systems. The algorithm optimizes the task allocation, priority assignment, network scheduling, and key-release interval length, with the consideration of the overhead and constraints from a time-delayed release of keys security mechanism. An industrial case study and a synthetic example demonstrate that our approach can effectively and efficiently explore the design space to meet all design requirements, and demonstrate the importance of considering security together with other metrics during the design stage.

[5]Here, utilization is defined as the ratio of the number of used (not scheduled) time slots to the number of total time slots.

TABLE II. Results of a large random test case.

	Security-Aware Mapping (Accelerated SA + OPT1 + OPT2)		
Signal Period Setting	1X	0.75X	0.5X
Network Utilization	0.464	0.597	0.859
Objective (*msec*)	20,403.161	24,714.822	45,513.222
Runtime (*s*)	280.823	334.415	440.395
Feasible Time (*s*)	21.384	29.395	60.946

Acknowledgment

This work was supported in part by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP), a Semiconductor Research Corporation program sponsored by MARCO and DARPA. This work was supported in part by the Industrial Cyber-Physical Systems Center (iCyPhy).

References

[1] L. Carloni, F. D. Bernardinis, C. Pinello, A. Sangiovanni-Vincentelli, and M. Sgroi, “Platform-based design for embedded systems,” Embedded Systems Handbook, CRC Press, 2005.

[2] 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,” USENIX Conf. on Security, 2011.

[3] FlexRay Consortium, “FlexRay communications system protocol specification,” v3.0.1, 2010.

[4] P. Kleberger, T. Olovsson, and E. Jonsson, “Security aspects of the in-vehicle network in the connected car,” IEEE Intelligent Vehicles Symp., pp. 528–533, 2011.

[5] F. Koushanfar, A.-R. Sadeghi, and H. Seudie, “EDA for secure and dependable cybercars: challenges and opportunities,” ACM/IEEE Design Automation Conf., pp. 220–228, 2012.

[6] C.-W. Lin, Q. Zhu, C. Phung, and A. Sangiovanni-Vincentelli, “Security-aware mapping for CAN-based real-time distributed automotive systems,” IEEE/ACM International Conf. on Computer-Aided Design, pp. 115–121, 2013.

[7] A. Perrig, R. Canetti, D. Tygar, and D. Song, “Efficient authentication and signing of multicast streams over lossy channels,” IEEE Symp. on Security and Privacy, pp. 56–73, 2000.

[8] A. Perrig, R. Canetti, D. Song, and D. Tygar, “Efficient and secure source authentication for multicast,” Network and Distributed System Security Symp., pp. 35–46, 2001.

[9] SAE Aerospace, “TTP communication protocol,” SAE AS6003 Standard, 2011.

[10] SAE Aerospace, “Time-Triggered Ethernet,” SAE AS6802 Standard, 2011.

[11] A. Wasicek, C. El-Salloum, and H. Kopetz, “Authentication in time-triggered systems using time-delayed release of keys,” IEEE International Symp. on Object/Component/Service-Oriented Real-Time Distributed Computing, pp. 31–39, 2011.

[12] Q. Zhu, Y. Yang, M. Di Natale, E. Scholte, and A. Sangiovanni-Vincentelli, “Optimizing the software architecture for extensibility in hard real-time distributed systems,” IEEE Trans. on Industrial Informatics, vol. 6, no. 4, pp. 621–636, 2010.