2019 IEEE 58th Conference on Decision and Control (CDC)
Palais des Congrès et des Expositions Nice Acropolis
Nice, France, December 11-13, 2019

Minimizing Side-Channel Attack Vulnerability via Schedule Randomization

Nils Vreman*, Richard Pates*, Kristin Krüger†, Gerhard Fohler†, Martina Maggio*
*Department of Automatic Control, Lund University, Sweden
{nils.vreman, richard.pates, martina.maggio}@control.lth.se
†Technische Universität Kaiserslautern, Germany
{krueger, fohler}@eit.uni-kl.de

***Abstract*—Control systems can be vulnerable to security threats where an attacker gathers information about the execution of the system. In particular, side-channel attacks exploit the predictability of real-time control systems and of their schedules. To counteract their action, a scheduler can randomize the temporal execution of tasks and limit the amount of information the attacker can gather. Schedule randomization is aimed at achieving the highest possible schedule diversity (measured using the *upper-approximated entropy*) during the real-time execution of the controller. This paper investigates fundamental limitations of schedule randomization for a generic taskset. The constructed schedule set has minimal size and achieves the highest possible upper-approximated entropy.**

I. Introduction

Security is a major concern for real-time and control systems [4], [5], [18]–[21]. Modern embedded systems are vulnerable to many different security threats [7], [13], one of them being side-channel attacks [17]. Side-channel attacks are based on attackers gathering knowledge about a system, and exploiting this knowledge to influence its behavior [1], [11], [15]–[17]. For example, recently, a team of researchers showed that it is possible to retrieve the engine speed from the frequency of execution of its control task [10]. In general an attacker knowing the schedule of an embedded control system can infer that the controller is sending a control signal to a plant periodically in predictable time slots. It can then use this knowledge to jam the network only when the control signal is being transmitted. Reducing the need for the attacker to be active also reduces the possibility of detecting the ongoing attack.

One of the logical countermeasures against this type of attacks is to impede the information gathering phase. In particular, an attacker that observes the execution of the real-time system should not be able to infer the tasks' schedule. However, classical scheduling algorithms are designed exactly for predictability and repeatability. Schedules (for periodic tasksets) usually repeat after a predetermined amount of time. This is precisely what gives an attacker the ability to observe the system and infer knowledge. An observer collecting information for long enough can then infer the pattern. Schedule randomization was proposed [22] to defend real-time systems against side-channel attacks. During the execution of the system, at the end of each hyperperiod, tasks' execution slots are shuffled, generating new schedules. The schedule is either generated online [22], or selected from a set of pre-generated schedules [8], [9]. For embedded systems, the overhead of online generation can be avoided if it is possible to compute and store a schedule set with acceptable diversity [8].

This paper studies the problem of generating a set of schedules that achieves maximum *diversity*, while having minimal *cardinality*. The minimal cardinality requirement ensures that the schedule set can be stored in the device limited memory and a simple random uniform choice can be implemented in the scheduler. The contribution of the paper is twofold.

- On the *theoretical* side, we identify analytical bounds and limitations for the problem of generating a set of schedules ensuring the maximum possible schedule diversity. We measure schedule diversity using the so called *upper-approximated entropy* [22]. We improve on [22] by determining an upper bound of the upper-approximated entropy value. We also compute the fewest schedules necessary to achieve the maximum diversity. These bounds are valid for a taskset with periodic tasks and constrained, implicit deadlines.
- On the *practical* side, we develop an efficient algorithm to generate the schedule set with minimal cardinality that achieves the maximum upper-approximated entropy possible, for a taskset consisting of tasks with implicit deadlines. We evaluate our algorithm with synthetic tasksets.

In the following, Section II contains a precise formalization of the problem addressed in the paper. Section III introduces our theoretical contribution: the definition of bounds for the solution of the given problem. Section IV describes our algorithm for schedule set generation and shows our experimental results. Section V discusses related research efforts.

II. Problem

In this section we formalize the problem of randomizing the execution of a taskset (composed of periodic tasks) in a platform with limited storage space. The problem comes from the desire to avoid side-channel attacks, i.e., attacks in which an attacker can learn (and exploit) the behavior of the

978-1-7281-1398-2/19/$31.00 ©2019 IEEE

system by observing its execution for a certain amount of time.

A. Attacker Model

The attack model used in this paper is inspired by previous contributions [22]. We assume that the attacker can observe execution traces and determine which task is running at any point in time [3]. This information can then be used to launch a targeted attack – for example impeding the transmission of control signals to actuators. We consider the attack to be successful if the attacker can predict which task ran at which point in time.

B. Notation

We are given a set of m periodically executed tasks $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_m\}$. Each task τ_i is defined as the tuple $\tau_i = \{e_i, t_i, d_i\}$, where e_i is the execution time to be given to the task, t_i is the task activation period, and d_i is the task deadline. For the remainder of the paper, only implicit deadline tasksets are treated; ergo, $t_i = d_i,\ i \in \{1, \ldots, m\}$. For the task to be scheduled correctly, the scheduling algorithm should give it access to the single-core CPU for e_i time units every activation, i.e., every t_i time units.

We denote with U the utilization of the set of tasks, i.e., $U = \sum_{i=1}^{m} {}^{e_i}/_{t_i}$. We assume that the taskset, $\mathcal{T}$, is schedulable; i.e., there exists a resource distribution such that each task meets its corresponding deadline. We denote with ℓ the hyperperiod of the taskset, i.e., the least common multiple $\text{lcm}(\cdot)$ of the task periods, $\ell = \text{lcm}(t_1, \ldots, t_m)$.

A schedule s for the tasks is a sequence of ℓ elements, that contains numbers in the set $\{0, 1, \ldots, m\}$, where 0 represents the execution of the idle task, and a number $i \in \{1, \ldots, m\}$ denotes the execution of task τ_i. Formally,

$$s = (s_1, s_2, \ldots, s_\ell)\,;\ s_j \in \{0, 1, \ldots, m\}. \tag{1}$$

We denote with s_j the value of the element in position j, i.e., the task that is executed according to the schedule s in the j-th time unit. Given that the set $\mathcal{T}$ is schedulable, we can safely assume that s respects all the constraints, such that for each task and each activation, the schedule assigns the prescribed execution time before the corresponding deadline.

The main objective is to determine a set $\mathcal{K}$ of k schedules that is as diverse as possible, keeping k as small as possible. We can decompose this problem into two different problems. The first part of the problem is determining the set of all the schedules that satisfy the constraints of execution times and periods. This set can have high cardinality, and it may not be possible to store all the schedules in the memory of the execution platform, e.g., an embedded system may have limited storage for alternative schedules. Suppose we obtain a set of n valid schedules $\mathcal{S}$ for the given taskset (each of length ℓ), $\mathcal{S} = \{s^{(1)}, s^{(2)}, \ldots, s^{(n)}\}$. The second part of the problem is then to select k schedules (from the set $\mathcal{S}$) to include in the set $\mathcal{K}$, to maximize the *diversity* of the schedules and avoid the possibility that the attacker gathers information about the tasks execution.

The diversity of the set of selected schedules $\mathcal{K}$ can be measured in different ways. The authors of [22] propose the use of entropy-like functions to measure the diversity of the schedule set, the *slot entropy* and the *upper-approximated entropy*. In the following, we are going to formalize these definitions to evaluate the schedule set diversity. However, to begin with, we define the slot count $C_{j,i,\mathcal{K}}$, as the number of occurrences of a task i in a given scheduling slot j in the set $\mathcal{K}$ (Definition 1). The slot count is then used to formally define the slot entropy $H_j(\mathcal{K})$ and the upper-approximated entropy $\tilde{H}(\mathcal{K})$ from [22]. We denote with $\phi(x)$ the function

$$\phi(x) = \begin{cases} 0 & x \leq 0 \\ -x \cdot \log_2(x) & x > 0 \end{cases}.$$

Definition 1. *Given a set of k valid schedules $\mathcal{K} = \{s^{(1)}, s^{(2)}, \ldots, s^{(k)}\}$ for the taskset $\mathcal{T}$, the j-th time unit, and the i-th task τ_i, we define the slot count $C_{j,i,\mathcal{K}}$ as a function that counts the occurrences of the task i in the given position j in the set $\mathcal{K}$. Using the square brackets as the Iverson brackets — that evaluates to 1 if the proposition inside the bracket is true, and to 0 otherwise — we can then write $C_{j,i,\mathcal{K}}$ as*

$$C_{j,i,\mathcal{K}} = \sum_{s^{(q)} \in \mathcal{K}} \left[s_j^{(q)} = i \right] = \sum_{q=1}^{k} \left[s_j^{(q)} = i \right]. \tag{2}$$

Using Definition 1, we can now formulate the slot entropy and the upper-approximated entropy according to the definitions given in [22].

Definition 2. *The slot entropy $H_j(\mathcal{K})$ can be written as a function of the tasks found in slot j, i.e.,*

$$H_j(\mathcal{K}) = \sum_{i=0}^{m} \phi\left(\frac{C_{j,i,\mathcal{K}}}{k}\right) = \sum_{i=0}^{m} -\frac{C_{j,i,\mathcal{K}}}{k} \cdot \log_2 \frac{C_{j,i,\mathcal{K}}}{k}. \tag{3}$$

Definition 3. *The upper-approximated entropy is the sum of all the slot entropies in the hyperperiod, i.e.,*

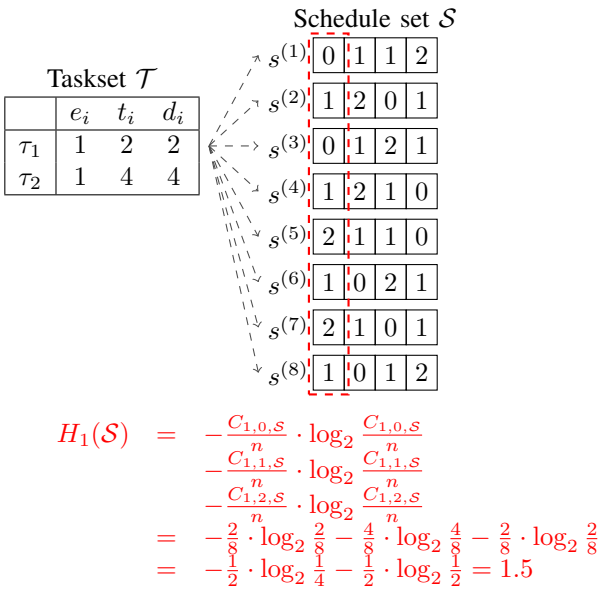
$$\tilde{H}(\mathcal{K}) = \sum_{j=1}^{\ell} H_j(\mathcal{K}) = \sum_{j=1}^{\ell} \sum_{i=0}^{m} -\frac{C_{j,i,\mathcal{K}}}{k} \cdot \log_2 \frac{C_{j,i,\mathcal{K}}}{k}. \tag{4}$$

C. Example

We here present a very simple example to help illustrate the quantities just introduced. We will use this as a running example in the remainder of the paper. We choose a simple taskset for this example. Its simplicity allows us to enumerate all the possible schedules and compute the upper-approximated entropy of all the possible schedule sets.

A taskset $\mathcal{T}$ is composed of two different tasks, $\mathcal{T} = \{\tau_1, \tau_2\}$. The first task τ_1 has execution time $e_1 = 1$, and period and deadline $t_1 = d_1 = 2$. The second task has execution time $e_2 = 1$ and period and deadline $t_2 = d_2 = 4$. The hyperperiod is then $\ell = 4$. The list of valid schedules for this taskset comprises 8 schedules, $\mathcal{S} = \{s^{(1)}, \ldots, s^{(8)}\}$.

Figure 1 shows the sequences and the computation of the slot entropy for the entire set $\mathcal{S}$ and the first slot $j = 1$, $H_1(\mathcal{S}) = 1.5$. Similarly, it can be shown that the slot entropy for the other slots is the same. The upper-approximated entropy for the set $\mathcal{S}$ is then $\tilde{H}(\mathcal{S}) = 4 \cdot 1.5 = 6$. Computing the power set of $\mathcal{S}$ and the upper-approximated entropy for



$$\begin{aligned} H_1(\mathcal{S}) &= -\tfrac{C_{1,0,\mathcal{S}}}{n}\cdot\log_2\tfrac{C_{1,0,\mathcal{S}}}{n} \\ &\quad -\tfrac{C_{1,1,\mathcal{S}}}{n}\cdot\log_2\tfrac{C_{1,1,\mathcal{S}}}{n} \\ &\quad -\tfrac{C_{1,2,\mathcal{S}}}{n}\cdot\log_2\tfrac{C_{1,2,\mathcal{S}}}{n} \\ &= -\tfrac{2}{8}\cdot\log_2\tfrac{2}{8}-\tfrac{4}{8}\cdot\log_2\tfrac{4}{8}-\tfrac{2}{8}\cdot\log_2\tfrac{2}{8} \\ &= -\tfrac{1}{2}\cdot\log_2\tfrac{1}{4}-\tfrac{1}{2}\cdot\log_2\tfrac{1}{2}=1.5 \end{aligned}$$

Fig. 1. Example of slot entropy computation for a given taskset $\mathcal{T}$ and schedule set $\mathcal{S}$.

each of the elements of the power set, it is possible to show that for $k=2$, the maximum upper-approximated entropy achievable is 4. With $k=3$ it is possible to reach an upper-approximated entropy ≈ 5. For $k=4$ (and $k=8$) the maximum of 6 is reached. With $k=5$, $k=6$, and $k=7$ it is possible to respectively reach values ≈ 5.786, ≈ 5.837, and ≈ 5.871.

In this case then, the minimal set of schedules that achieves the highest upper-approximated entropy has cardinality $k=4$. One possible optimal set is $\mathcal{K}=\{s^{(1)},s^{(2)},s^{(5)},s^{(6)}\}=\{(0\,1\,1\,2),(1\,2\,0\,1),(2\,1\,1\,0),(1\,0\,2\,1)\}$. The 4-elements set $\mathcal{K}$ that gives the maximum upper-approximated entropy is not unique. Another alternative would be $\mathcal{K}=\{s^{(3)},s^{(4)},s^{(7)},s^{(8)}\}$. If the example is as simple as this one, it is possible to enumerate all the alternatives and determine optimal solutions with a brute force approach. However, for tasksets with large hyperperiods, it is infeasible to enumerate all the valid schedules and to compute the power set of these schedules to determine an optimal solution.

D. Problem Formulation

Now it is possible to formally state the objective of minimizing the risk that an attacker would be able to extract and effectively use information to compromise the system. Given a taskset $\mathcal{T}$ and the set of all valid schedules $\mathcal{S}$, what is the smallest subset of $\mathcal{S}$ that maximizes the upper-approximated entropy?

Mathematically, this problem can be written as the following optimization problem.

Problem 1. *Given a taskset $\mathcal{T}$ and a valid set of schedules $\mathcal{S}$, solve*

$$\begin{aligned} \mathcal{K}^* &= \arg\min_{\mathcal{K}\subseteq\mathcal{S}} |\mathcal{K}| \\ \text{s.t. } \tilde{H}(\mathcal{K}) &= \max_{\mathcal{L}\subseteq\mathcal{S}} \tilde{H}(\mathcal{L}). \end{aligned}$$

Here $\mathcal{L}$ is any generic subset of $\mathcal{S}$, and we search for the set $\mathcal{K}$ with the minimum cardinality that achieves the maximum upper-approximated entropy. This problem consists of two main aspects. First we must determine the maximum upper-approximated entropy; that is we must solve

$$\tilde{H}^* = \max_{\mathcal{L}\subseteq\mathcal{S}} \tilde{H}(\mathcal{L}).$$

Then we must find the smallest subset $\mathcal{K}^*\subseteq\mathcal{S}$ with upper-approximated entropy $\tilde{H}(\mathcal{K}^*)=\tilde{H}^*$.

III. Fundamental Limits

Problem 1 could be approached by an exhaustive search. If one evaluated the upper-approximated entropy for every element in the power set of $\mathcal{S}$, the optimal solution to Problem 1 could be obtained by choosing the smallest subset that achieves $\tilde{H}^*$. However since the cardinality of $\mathcal{S}$ is typically large, this will be infeasible in practice.

One question then arises: Are there any fundamental limits on the achievable maximum upper-approximated entropy or the cardinality of $\mathcal{K}$? The remainder of this section is devoted to answering this question. We show that the properties of the taskset $\mathcal{T}$ impose fundamental limits both on $\tilde{H}^*$, and on the cardinality of the subsets that can achieve this bound.

The fact that each task in the taskset $\mathcal{T}$ must be executed with a certain frequency imposes a fundamental limit on the maximum upper-approximated entropy. The intuitive explanation for this is as follows. It is our objective to select a set of schedules that minimizes the information that an attacker can obtain by observing the execution of any individual task. We therefore want it to seem as if each task was allocated randomly to each given slot. However a random allocation with equal probability is not possible, because we are required to execute tasks for given time units a certain number times in each hyperperiod. Therefore the best we can do is make each task appear random with probability specified by its relative frequency in the hyperperiod. The entropy of the corresponding random variable then specifies an upper bound on the upper-approximated entropy of any set of schedules $\mathcal{K}\subseteq\mathcal{S}$.

Theorem 1. *Given any schedule set $\mathcal{K}\subseteq\mathcal{S}$,*

$$\tilde{H}(\mathcal{K}) \le \ell\cdot\sum_{i=0}^{m}\phi\left({}^{e_i}/_{t_i}\right).$$

Proof. When $x>0$, the function $\phi(x)=-x\cdot\log_2(x)$ is continuous and concave. This implies that ${}^1/_p\sum_{i=1}^{p}\phi(x_i)\le\phi\left({}^1/_p\sum_{i=1}^{p}x_i\right)$.

Applying this inequality to the expression in Equation (4), we obtain

$$\frac{1}{\ell}\sum_{j=1}^{\ell}\sum_{i=0}^{m}\phi\left(\frac{C_{j,i,\mathcal{K}}}{k}\right)\le\sum_{i=0}^{m}\phi\left(\frac{1}{\ell}\sum_{j=1}^{\ell}\frac{C_{j,i,\mathcal{K}}}{k}\right),$$

where $k = |\mathcal{K}|$ and the expression $\sum_{j=1}^{\ell} \frac{C_{j,i,\mathcal{K}}}{k}$ counts – for all the schedules in $\mathcal{K}$ – the amount of occurrences of each task in the hyperperiod, and can be written as

$$\sum_{j=1}^{\ell} \frac{C_{j,i,\mathcal{K}}}{k} = \frac{k}{k} \cdot \frac{e_i}{t_i} \cdot \ell,$$

thus leading to $\tilde{H}(\mathcal{K})/\ell$ being

$$\frac{1}{\ell} \sum_{j=1}^{\ell} \sum_{i=0}^{m} \phi\left(\frac{C_{j,i,\mathcal{K}}}{k}\right) \leq \sum_{i=0}^{m} \phi\left(\frac{\ell}{\ell} \frac{e_i}{t_i}\right) = \sum_{i=0}^{m} \phi\left(\frac{e_i}{t_i}\right).$$

Knowing that $\ell > 0$, we then derive the upper bound

$$\tilde{H}(\mathcal{K}) \leq \ell \cdot \sum_{i=0}^{m} \phi\left({}^{e_i}/_{t_i}\right) = \tilde{H}^{ub}, \tag{5}$$

for the achievable upper-approximated entropy. □

We denote the bound in Equation (5) with $\tilde{H}^{ub}$. We now state two corollaries that are more relaxed versions of the upper bound given in Theorem 1 (i.e., quantities that upper bound $\tilde{H}^{ub}$). The first one requires the least amount of information about the taskset and is applicable to all tasksets containing m tasks. The second one tightens the bound even further given that we know the tasksets total utilization U as well as the number of tasks in the taskset m.

Corollary 1. *Given any schedule set $\mathcal{K} \subseteq \mathcal{S}$,*

$$\tilde{H}(\mathcal{K}) \leq \ell \cdot \sum_{i=0}^{m} \phi\left({}^{e_i}/_{t_i}\right) \leq -\ell \cdot \log_2\left({}^{1}/_{(1+m)}\right). \tag{6}$$

This bound comes directly from applying the same principle used to prove Theorem 1 to Equation (5). The expression in Equation (6) is not always reachable, depending on the characteristics of the taskset. This leads us to consider the taskset characteristics. In particular, we can compute a bound that takes into account the utilization U of the taskset, leading to the following corollary.

Corollary 2. *Given any schedule set $\mathcal{K} \subseteq \mathcal{S}$,*

$$\tilde{H}(\mathcal{K}) \leq \ell \cdot \{-(1-U) \cdot \log_2(1-U) - U \cdot \log_2({}^{U}/_{m})\}. \tag{7}$$

The contribution of the idle task to the upper approximated entropy is equal to $\ell \cdot \{-(1-U) \cdot \log_2(1-U)\}$. The maximum value for the upper approximated entropy is reached when the utilizations of the tasks allow them to be evenly distributed. The contribution of each of them is then $-\ell \cdot {}^{U}/_{m} \cdot \log_2({}^{U}/_{m})$. Deriving the expression in Equation (7) allows us also to study when we can be closer to the bound in Equation (6) — and when we can expect to reach the maximum upper-approximated entropy for a set of m tasks, depending on the task characteristics. With respect to the utilization $U = \sum_{i=1}^{m} {}^{e_i}/_{t_i}$, the upper approximated entropy can reach its maximum when the utilization of the system is equal to $U = {}^{m}/_{(1+m)}$.

We will now show that a given schedule set $\mathcal{K} \subseteq \mathcal{S}$ can only achieve an upper-approximated entropy of $\tilde{H}^{ub}$ if it has cardinality of at least

$$\frac{\ell}{\gcd({}^{e_i}/_{t_i} \cdot \ell)}.$$

This is important, since it shows that if we want to achieve the upper bound on the upper-approximated entropy from Theorem 1, we must use a schedule set of at least the size given above.

Theorem 2. *Given any schedule set $\mathcal{K} \subseteq \mathcal{S}$, if*

$$|\mathcal{K}| < \frac{\ell}{\gcd\left({}^{e_i}/_{t_i} \cdot \ell\right)},$$

then the inequality in Equation (5) *is strict.*

Proof. To achieve the upper bound $\tilde{H}^{ub}$, the contribution to the upper-approximated entropy of each slot should be maximized. This happens when the slot entropy H_j is equal to $H_j = \sum_{i=0}^{m} \phi({}^{e_i}/_{t_i})$ for each slot. The contribution of each task to the slot entropy should then be equal to the task utilization ${}^{e_i}/_{t_i}$. To find a lower bound for k, we look at a single slot j. In fact, additional slots will only potentially increase the number of schedules needed to achieve higher upper-approximated entropy. We formulate the problem of finding $|\mathcal{K}^*| = k^*$ as an optimization problem.

$$\begin{aligned} \min_{\substack{k \in \mathbb{Z}^+ \\ C_{j,i,\mathcal{K}} \in \mathbb{Z}^+}} \quad & k \\ \text{s.t.} \quad & C_{j,i,\mathcal{K}} = \frac{e_i}{t_i} \cdot k, \quad \forall i \in \{0, \ldots, m\}. \end{aligned} \tag{8}$$

This means that we have a positive integer number of schedules in the set $\mathcal{K}$ that allows $C_{j,i,\mathcal{K}}$ (the number of times task i appears in slot j in set $\mathcal{K}$) to be a positive integer number for each task i. We perform a variable substitution and define $y = \ell/k$. Minimizing k now becomes equivalent to maximizing y. Relaxing temporarily the requirement that k be an integer, the problem in Equation (8) is reformulated as

$$\begin{aligned} \max_{C_{j,i,\mathcal{K}} \in \mathbb{Z}^+} \quad & y \\ \text{s.t.} \quad & C_{j,i,\mathcal{K}} = \frac{e_i}{t_i} \cdot \frac{\ell}{y}, \quad \forall i \in \{0, \ldots, m\}. \end{aligned} \tag{9}$$

The solution to the problem in Equation (9) is that y should be equal to the greatest common divisor of the utilizations multiplied by the hyperperiod, $y = \gcd({}^{e_i}/_{t_i} \cdot \ell)$, which yields to

$$k^* = \frac{\ell}{\gcd\left({}^{e_i}/_{t_i} \cdot \ell\right)}. \tag{10}$$

For this to be the solution of the optimization problem in Equation (8), k^* must be an integer number. Because the utilizations of the taskset (including the idle task) sum to one, $\sum_{i=0}^{m} {}^{e_i}/_{t_i} = 1$, the constraint for the idle task in the optimization problem of Equation (9) can also be written as

$$C_{j,0,\mathcal{K}} = \left(1 - \sum_{i=1}^{m} \frac{e_i}{t_i}\right) \cdot \frac{\ell}{y} = \left(\frac{\ell}{y} - \frac{\ell}{y} \cdot \sum_{i=1}^{m} \frac{e_i}{t_i}\right).$$

The solution of problem (9) ensures that $\frac{\ell}{y} \cdot \sum_{i=1}^{m} {}^{e_i}/_{t_i} \in \mathbb{Z}^+$ due to the m constraints for the (non idle tasks in the) taskset. The value of ℓ/y (equal to k^*) must then be a positive integer number. This implies that the solution of the problem in

Equation (9) is also a solution of the problem in Equation (8) and $k^* \in \mathbb{Z}^+$. □

Remark 1. *A simple modification of the proof of Theorem 2 shows that for any $\mathcal{K} \subseteq \mathcal{S}$, if*

$$|\mathcal{K}| \mod \frac{\ell}{\gcd(e_i/t_i \cdot \ell)} \neq 0,$$

then $\tilde{H}(\mathcal{K}) < \tilde{H}^{ub}$. This means that the cardinality of $\mathcal{K}$ must be a multiple of $\frac{\ell}{\gcd(e_i/t_i \cdot \ell)}$ in order to achieve the bound in Theorem 2.

A. Example

Applying Theorems 1 and 2 to the example presented in Section II-C gives us:

$$\tilde{H}^{ub} = \ell \cdot \sum_{i=0}^{m} \phi\left(e_i/t_i\right) = 4 \cdot \{\phi(1/2) + \phi(1/4) + \phi(1/4)\} = 6,$$

and

$$k^* = \frac{\ell}{\gcd\left(e_i/t_i \cdot \ell\right)} = \frac{4}{\gcd\left(4/2, 4/4, 4/4\right)} = 4.$$

IV. Evaluation

In order to investigate the limits presented above, we implemented a search algorithm based on constraint optimization that generates a set of schedules maximizing the upper-approximated entropy for a given taskset [1].

We tested the algorithm with synthetic tasksets, composed of n tasks with $n \in \{2, \ldots, 10\}$ and maximum hyperperiod $\ell \in \{100, 200, 300, 400, 500\}$. For each of the possible combinations, we generated 1000 tasksets using an extension of the UUniFast algorithm [2], that allows us to control the maximum hyperperiod. We randomized the periods t_i of each task, and imposed an implicit deadline constraint $t_i = d_i, \forall i \in \{1, \ldots, n\}$. We computed the upper-approximated entropy $\tilde{H}^{ub}$ for the taskset according to Theorem 1. We then computed the average contribution of each slot $\tilde{H}^{ub}/\ell$ to be able to compare tasksets with different hyperperiods. Finally, we ran our algorithm to generate the schedule set $\mathcal{K}$ with the lowest cardinality k^* given by Theorem 2. The algorithm was executed using an Intel Xeon E5-2620 v3 @ 2.40 GHz server with 24 CPUs and 132 Gb RAM.

In principle, the algorithm could run for a long time to find the maximum, thus we limited the duration and allowed a budget of 60 minutes to compute the schedule set. For practical reasons a maximum computation time was set. However, since the schedule set generation is performed offline, the time to generate a schedule set is inconsequential. We define the algorithm *accuracy* as the percentage of tasksets for which the algorithm found an optimal schedule set within the computation time budget. In Figure 2, we report the accuracy for the different combinations of maximum hyperperiod and number of tasks. On the right side of the figure we show the average contribution of each slot to the upper-approximated entropy $\tilde{H}^{ub}/\ell$ for the combination of $\ell = 300, n = 2$ as well as $\ell = 300, n = 10$.

The dots in the figures show the tight bounds and the achievable entropy, while the dashed line represents the (non-tight) bound discussed in Corollary 2, Equation (7). As can be seen, for some of the tasksets, the constraints imposed by the execution times and the periods allow the upper-approximated entropy to reach this bound, but in other cases the bound presented in Theorem 1 is tighter. One can also notice that the variance of the achieved upper-approximated entropy increases when the utilization increases. This is because a higher utilization introduces tighter constraints on the achievable entropy.

Another important result is the fact that the upper-approximated entropy reaches the maximum of the bound discussed in Corollary 2 for $m/(m+1)$, m being the number of tasks. The implication of this is that from a security perspective the optimal system utilization is $m/(m+1)$. This is very important for determining critical system paramters, like the taskset utilization, for security-aware embedded and real-time systems.

Finally, the red lines represent the (non-tight) bound introduced by Corollary 1. In most cases, this bound is unreachable, due to the constraints introduced by the tasks characteristics.

V. Related Work

Security is an important concern for modern real-time and control systems. Recently, Jiang *et al.* [6] analyzed the robustness against differential power analysis attacks of Earliest Deadline First and Fixed Priority with Rate Monotonic priority assignment. Recent research on real-time security demonstrated that it is possible to extract information about tasksets running on real-time systems with fixed-priority scheduling [3], [12], [14] also for control systems [10]. This constitutes a side-channel attack [17]. After inferring the schedule, a targeted attack can exploit its characteristics to disturb the real-time system's operation, for example interfering with the correct execution of a control task. An alternative is to covertly interfere with data transmission during precise time intervals in which the task is transmitting data, generating network interference or preventing packet transmissions.

There has been extensive research regarding schedule randomization methods utilizing the upper-approximated entropy. For instance, to counter side-channel attacks, TaskShuffler [22] generates new schedules using task randomization to obfuscate attackers. The upper-approximated entropy of the schedules generated at runtime is used to quantify the diversity of the execution, and to then capture the probability of learning the real-time system's schedule. The authors propose an algorithm for randomization which is guided by this quantity and shuffles the tasks, trying to maximize upper-approximated entropy while enforcing the scheduling constraints at runtime. Contrary to TaskShuffler, our solution generates the list of schedules offline, and ensures optimality in both storage needs and upper-approximated entropy.

[1]The code for our experiments is available online, together with additional experiments not shown in the paper due to space constraints: *https://gitlab.control.lth.se/NilsVreman/rand-sched*.

$n \backslash \ell$	100	200	300	400	500
2	100.0	99.9	100.0	100.0	100.0
3	100.0	96.9	92.1	85.9	82.4
4	99.7	90.5	80.3	70.8	66.7
5	97.4	81.0	69.7	61.7	52.5
6	95.2	76.6	58.4	52.6	44.6
7	92.8	70.9	56.9	45.2	38.4
8	89.5	67.3	47.2	39.3	31.8
9	89.0	62.5	45.5	37.1	28.7
10	88.0	59.4	38.7	32.4	24.7

accuracy with varying number of tasks and hyperperiods

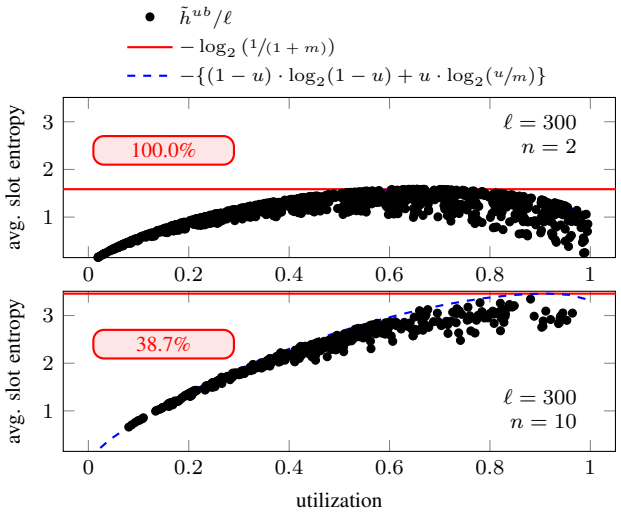


Fig. 2. Accuracy of the schedule set generator for various set of parameters n, ℓ (left) and example of average slot entropy over total system utilization, U, and corresponding bounds for tasksets with $\ell = 300, n = 2$ and $\ell = 300, n = 10$ (right).

Indeed, the additional overhead of generating new schedules may not be acceptable for embedded systems [8]. In this case, a schedule set can be generated offline. Online, at the end of each hyperperiod a new schedule is picked from the set, according to a uniform probability distribution [9]. Krüger *et al.* proposes this strategy, and also evaluates the schedule set based on its upper-approximated entropy. Our work proposes an alternative method for the offline generation of the schedule set. Our choice is based on analytical bounds on the achievable upper-approximated entropy, which have not been previously explored. We therefore can ensure the optimality of our solution with respect to both the size of the schedule set and its diversity.

VI. Conclusion and Future Work

This paper describes recent research advances in the security of real-time and control systems. Its contribution is to establish analytical bounds for the diversity of a schedule set. First, we derive a bound on the upper-approximated entropy of the set. Then we derive bounds on the minimum number of schedules needed to achieve said upper-approximated entropy's upper bound. We also design and implement an algorithm to generate the optimal schedule set for randomization of tasksets composed of periodic tasks with implicit deadlines.

References

[1] D. Agrawal, B. Archambeault, J. Rao, and P. Rohatgi. The EM side-channel(s). In *4th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES, 2002.

[2] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2), May 2005.

[3] C. Chen, A. Ghassami, S. Mohan, N. Kiyavash, R. B. Bobba, R. Pellizzoni, and M. Yoon. A reconnaissance attack mechanism for fixed-priority real-time systems. arXiv:1705.02561, 2017.

[4] M. Hasan, S. Mohan, R. Pellizzoni, and R. B. Bobba. A design-space exploration for allocating security tasks in multicore real-time systems. In *Design, Automation & Test in Europe*, DATE, 2018.

[5] J. Hendrickx, K. Johansson, R. Jungers, H. Sandberg, and K. Sou. Efficient computations of a security index for false data attacks in power networks. *IEEE TAC*, 59(12):3194–3208, 2014.

[6] K. Jiang, L. Batina, P. Eles, and Z. Peng. Robustness analysis of real-time scheduling against differential power analysis attacks. In *IEEE Computer Society Annual Symposium on VLSI*, ISVLSI, 2014.

[7] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre attacks: Exploiting speculative execution. meltdownattack.com, 2018.

[8] K. Krüger, M. Völp, and G. Fohler. Improving security for time-triggered real-time systems against timing inference based attacks by schedule obfuscation. In *Work-in-Progress Proceedings of the 29th Euromicro Conference on Real-Time Systems*, ECRTS, 2017.

[9] K. Kruger, M. Völp, and G. Fohler. Vulnerability analysis and mitigation of directed timing inference based attacks on time-triggered systems. In *Euromicro Conference on Real-Time Systems*, ECRTS, 2018.

[10] S. Liu, N. Guan, D. Ji, W. Liu, X. Liu, and W. Yi. Leaking your engine speed by spectrum analysis of real-time scheduling sequences. *Journal of Systems Architecture*, 2019.

[11] E. Mateos and C. Gebotys. A new correlation frequency analysis of the side channel. In *Workshop on Embedded Systems Security*, 2010.

[12] S. Mohan, M. Yoon, R. Pellizzoni, and R. B. Bobba. Real-time systems security through scheduler constraints. In *26th Euromicro Conference on Real-Time Systems*, 2014.

[13] D. Papp, Z. Ma, and L. Buttyan. Embedded systems security: Threats, vulnerabilities, and attack taxonomy. In *13th Annual Conference on Privacy, Security and Trust*, PST, 2015.

[14] R. Pellizzoni, N. Paryab, M. Yoon, S. Bak, S. Mohan, and R. B. Bobba. A generalized model for preventing information leakage in hard real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2015.

[15] T. Popp, S. Mangard, and E. Oswald. Power analysis attacks and countermeasures. *IEEE Design & test of Computers*, 24(6), 2007.

[16] J. Son and J. Alves-Foss. Covert timing channel capacity of rate monotonic real-time scheduling algorithm in MLS systems. In *Communication, Network, and Information Security*, 2006.

[17] R. Spreitzer, V. Moonsamy, T. Korak, and S. Mangard. Systematic classification of side-channel attacks: A case study for mobile devices. *IEEE Communications Surveys and Tutorials*, 20(1), 2018.

[18] A. Teixeira, I. Shames, H. Sandberg, and K. Johansson. Revealing stealthy attacks in control systems. In *Allerton Conference on Communication, Control, and Computing*, pages 1806–1813, 2012.

[19] A. Teixeira, I. Shames, H. Sandberg, and K. Johansson. Distributed fault detection and isolation resilient to network model uncertainties. *IEEE Transactions on Cybernetics*, 44(11):2024–2037, Nov 2014.

[20] M. Vai, R. Khazan, D. Utin, S. O'Melia, D. Whelihan, and B. Nahill. Secure embedded systems. Technical report, MIT Lincoln Laboratory Lexington United States, 2016.

[21] S. H. Weingart. Physical security devices for computer subsystems: A survey of attacks and defenses. In *Cryptographic Hardware and Embedded Systems*, CHES, 2000.

[22] M. Yoon, S. Mohan, C. Chen, and L. Sha. TaskShuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS, 2016.