SPARTA: A Scheduling Policy for Thwarting Differential Power Analysis Attacks

Ke Jiang*, Petru Eles*, Zebo Peng*, Sudipta Chattopadhyay*, Lejla Batina†
ke.jiang@liu.se, petru.eles@liu.se, zebo.peng@liu.se, sudipta.chattopadhyay@liu.se, lejla@cs.ru.nl
*Department of Computer and Information Science, Linköping University, Sweden
†Institute for Computing and Information Sciences, Radboud University Nijmegen, The Netherlands

***Abstract*—Embedded systems (ESs) have been widely used in various application domains. It is very important to design ESs that guarantee functional correctness of the system under strict timing constraints. Such systems are known as the real-time embedded systems (RTESs). More recently, RTESs started to be utilized in safety and reliability critical areas, which made the overlooked security issues, especially confidentiality of the communication, a serious problem. Differential power analysis attacks (DPAs) pose serious threats to confidentiality protection mechanisms, i.e., implementations of cryptographic algorithms, on embedded platforms. In this work, we present a scheduling policy, SPARTA, that thwarts DPAs. Theoretical guarantees and preliminary experimental results are presented to demonstrate the efficiency of the SPARTA scheduler.**

I. Introduction

Nowadays, it is common to find embedded systems controlling critical functionalities in various aspects of our society, from consumer electronics to military weapons. Many of these systems, called real-time embedded systems (RTESs), are demanded to provide strict guarantees on the timeliness of delivered results. In such systems, not only the correctness of the final result is important, but the timing of the computation is also vital for the system. That is, the system is expected to deliver a correct result within a given deadline. How to achieve timeliness has been widely studied under different contexts in literature, for example, [1], [2].

The emerging trend of using embedded systems in safety and reliability critical applications urges the needs of providing security properties. As security was often treated as an afterthought by system designers, more and more security drawbacks of various embedded systems, e.g., [3], as well as embedded implementations of cryptographic algorithms, e.g., [4], have been revealed in recent years. In order to achieve the best protection, in the presence of timing and resource constraints imposed in embedded environment, the security issues should be studied and taken into consideration during the early system design and optimization phase, e.g., in [5].

Among the concepts of security, confidentiality is of central importance. The fundamental step to achieve confidentiality is to apply cryptography. Considering the actual constrained environment of RTESs, the most applicable cryptographic algorithm for confidentiality protection is the Advanced Encryption Standard (AES) [6], because of its robust protection strength and high throughput rate on microprocessors [7]. We will, in this paper, concentrate on AES, which is arguably the most widely used cryptosystem. However, the concepts and techniques presented in this paper are general enough to be also applied on other iterated block ciphers.

The side-channel attacks (SCAs) targeting the implementations of cryptographic algorithms (including AES) pose severe threats to RTES security. The authors of [8] presented an automated hardware design methodology that inserts random jitters for counteracting SCAs. In [9], a software approach for generating random delays is proposed, and can be used against SCAs. However, neither of these works can be applied in RTESs, since the timing properties may be affected. Previously, we have studied the influence of existing schedulers on difficulties of mounting differential power analysis attacks (DPAs), a specific kind of SCA, on RTESs in [10]. However, no extra protection scheme was presented. To the best of our knowledge, this is the first scheduling policy that both guarantees the real-time properties of the system and thwarts potential DPAs on AES implementations.

II. The System

The system we consider is a mono-processor RTES. It is connected with various peripherals, e.g., sensors, actuators, and communication modules, via which it interacts with the environment or other peers. The microprocessor μP is implemented with tamper-proof technology [11] making the operations and memory hierarchy not manipulatable. The set of preemptive and periodic computation tasks running on the microprocessor is denoted as $\mathcal{T}$. The execution time and period (also the deadline) of task $\tau_i \in \mathcal{T}$ is denoted as e_i and T_i, respectively. Task τ_i may generate or/and receive a set of messages $\mathcal{M}_i$ to interact with other nodes ($\mathcal{M}_i = \emptyset$ if no message is associated). The length (in number of AES block) of message $m_{ij} \in \mathcal{M}_i$ is l_{ij}. The tasks are scheduled based on the proposed policy which is elaborated in Section V.

To make sure the communication from/to the system is confidential, we carry out AES on associated messages. The messages in $\mathcal{M}_i$ are all processed with the same AES secret key K_i. The AES encryption/decryption process τ_{ij}^{AES} on message m_{ij} is part of the processing in task τ_i. In other words, task τ_i (and execution time e_i) includes the normal computations of the task and the corresponding AES operations (and respective time overhead e_{ij}^{AES}) on all associated messages $m_{ij} \in \mathcal{M}_i$.

III. DPA Attacks and Countermeasures

The attacker aims to find the secret key(s) of the system using DPA attacks, and can accurately measure the power consumption of the microprocessor. She knows the task parameters, e.g., their execution times and periods, and can replace the messages to AES operations with arbitrary data, e.g., by changing the data from the sensors. As the microprocessor is tamper-resistant, she cannot change the task schedule, or directly read out the secret key(s) from memory.

A. DPA Attacks

AES is known to be robust against the theoretical crypt-analysis attacks. However, the practical attacks on AES implementations have presented serious threats to the security of AES. In [4], Paul Kocher et al. presented the so-called differential power analysis attack (DPA), one type of SCA, that has become the most efficient attack scheme targeting AES implementations on embedded platforms, e.g., [12]. The power consumption of a device at a certain time depends on the performed operations and processed data. For the case of AES encryptions on different inputs with the same secret key, the operations and involved secret keys are the same, at the same relative time t with respect to corresponding start time of AES. The differences in power consumption, in the ideally noisy-free case, only come from the different input data.

The DPA attacks try to reveal an AES secret key K_i in the granularity of subkey[1], and the procedure is, in brief, as follows. The attacker first identifies a fraction in an AES round that is a function of a given text and a 8-bits subkey sk_{ij}. This fraction is referred to as a leakage point LP_{ij}. Of all the AES encryption (or decryption) operations with the same AES secret key K_i, the same subkey sk_{ij} is used to operate on different input texts using the same function at LP_{ij}. Therefore, there exists a certain relation among all the measured power consumptions at LP_{ij}. After identifying LP_{ij}, the attacker feeds the AES process with chosen plaintexts, and measures the power of the processor. Then, based on period T_i of task τ_i, she divides the whole obtained power trace (of G time units) into $S = G/T_i$ number of samples, and organizes the samples into a 2D matrix $P = [i - j](i = 1, ..., S; j = 1, ..., V)$ with size $S * V$, in which $V = T_i \times \mathcal{F}$. $\mathcal{F}$ is the measurement frequency of her attacking equipment. Thus, V is the number of obtained power values within T_i. Each $P_{x,y}$ denotes an actual measured power value of sample x at relative time point y.

The next step is to produce the hypotheses regarding the processor power at leakage point LP_{ij}. Since there are only $2^8 = 256$ different possibilities of subkey sk_{ij}, the attacker can enumerate all the possible sk_{ij} values on all the plaintexts she used, and derive another 2D matrix $H = [i - j](i = 1, ..., S; j = 1, ..., 256)$, each of which is a hypothetical power value of corresponding plaintext-subkey pair, and is calculated depending on her knowledge about the underlying hardware. The last step is to find correlations between the actual power of the processor and the attacker's hypothetical power on each column P_i of P and H_j of H, e.g., by calculating the Pearson correlation coefficient (PCC) ρ_{ij}. Column P_i and H_j has high correlation if ρ_{ij} is high. And the highest value $\rho_{max} = \rho_{xy}$ reveals that sk_y was the real subkey used at relative time t_x. Then, the attacker tries to recover the whole secret key K_i by going through all the subkeys, or until it is trivial to mount a brute-force attack on the rest of the key bits.

[1]The definition and how to obtain the AES subkeys can be found in [6].

B. Time Dimension Shuffling Based Countermeasures

In order to make AES robust against DPA attacks, we need to introduce countermeasures on the AES implementation. As can be noticed from the previous sections, the DPA attacks have certain limitations, i.e., the samples need to be noise-free and well aligned. Having noise-free samples means that the power of the processor is accurately predictable from the preformed operations and processed data. Thus, the attacker can make relatively accurate hypotheses, and the highest correlation is obvious enough. This leads to the type of countermeasures aiming at reducing the signal-noise-ratio (SNR), e.g., [13]. In this work, we focus on exploiting the second limitation of DPAs, which is that the values in column P_i indicating the power value of all the samples at time i should be due to the same operation.

Fig. 1 depicts two aligned power traces (indicated by supply voltage) of AES on two different messages with the same key K. The leakage point of the first subkey sk_1 occurs at 1.2ms (highlighted by the red rectangle). Assuming that two samples are sufficient to observe a high correlation with the hypothetical powers, then the attacker can already retrieve sk_1 if Fig. 1 is obtained. However, if the two leakages happened at different time, then she needs more samples to observe a dominating correlation for obtaining sk_1. Therefore, what we propose in this paper is to reduce the alignment probability of leakage points in the columns of matrix P. Such an approach of implementing countermeasures is often referred to as *hiding in time dimension* [14]. Note that both countermeasure techniques are compatible with each other, and it is recommended to implement multiple countermeasures to achieve the best protection to the AES implementations.

As just mentioned, DPA attacks work better if the operations at the same relative time point of all samples are the same. That is, in all the samples, the leakage points all occur at the same relative time with respect to the starting time of the sample. However, just assigning random delays into the execution, e.g., [9], does not work in the context of RTES, since it may break the deadline constraints. In the meantime, dynamic task schedulers, such as the earliest deadline first policy (EDF), can serve as support for implementing the idea of *hiding in time dimension*. Now the question is how to quantify the performance of a scheduler as a countermeasure.

Assuming that the highest correlation between two columns of P and H is ρ_{max}, then our goal is to reduce ρ_{max}. Due to the preemptions introduced by the dynamic schedulers, the occurrences of leakage point LP_{ij} with respect to sk_{ij} may happen at different times for different releases of task τ_i, thus, reducing the value of ρ_{max}. If $\forall \rho : \rho \approx 0$, then there is no dominating ρ_{max}, that is, there is no clear correlation between any columns of P and H. This is the optimal case in terms of protection against DPA, meaning that the attacker needs infinite amount of samples to observe a high ρ_{max}. Let us denote the moment of time when LP_{ij} happens with the highest probability among all the samples as $\hat{t}$. The corresponding column of $P_{\hat{t}}$ will have the highest correlation with the column $H_{sk_{ij}}$ of H which is the actual subkey used

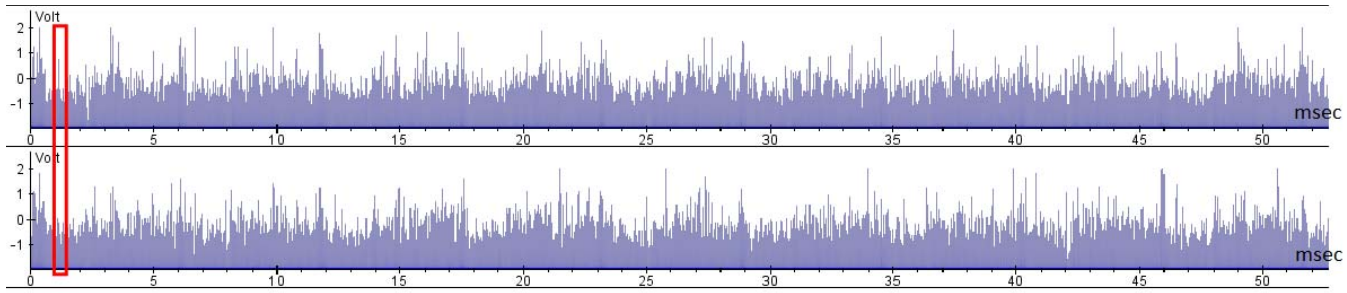

Fig. 1. Power traces of AES on two plaintexts with the same secret key

in the leakage point. If the probability that the leakage point occurs at $\hat{t}$ is $\hat{p}$, then, as shown in [14], we can calculate the lower bound of the number of samples to observe a noticeable ρ_{max} as follows,

$$\mathcal{N} = 3 + \frac{13.148}{ln^2(\frac{1+\hat{p}}{1-\hat{p}})}. \quad (1)$$

Now, we define the robustness of the secret key K_i against DPA attacks as the total amount of time needed to gather those $\mathcal{N}$ samples:

$$\mathcal{R}_i = \mathcal{N} * T_i, \quad (2)$$

where T_i is the period of τ_i having the secret key K_i.

IV. An Illustrative Example

As mentioned in the previous section, the highest probability $\hat{p}$ that the leakage point occurs at the same relative time point $\hat{t}$ determines the robustness of the key under analysis. One way to get $\hat{p}$ is through simulation of system execution, that is, to simulate and record the system execution and then to apply statistical methods on the recorded schedule. Thereafter, the robustness of the key can be calculated using Eq. 2. Let us consider a simple example with three tasks τ_1, τ_2, and τ_3. The execution times are 3, 8, and 9, respectively, and the periods are 10, 20, and 30, respectively. Task τ_1 and τ_3 do not have message communication, while, task τ_2 generates one 128-bits long message m_2, and encrypts it with AES before sending it out over the communication module. Due to the lack of space, we make the following simplification in the rest of this paper: Each task is only associated with at most one message encryption occurring in the end of its execution, and the leakage point happens at the last δ time units of the task execution time. However, all the presented techniques and analyses can be trivially extended to more practical cases in which a task can be associated with multiple messages, and the leakage point can occur arbitrarily anywhere within task executions. In this example, we assume that the 8-th time unit of τ_2 is the leakage point.

If the system is scheduled by EDF, we will get the schedule as shown in Fig. 2 (a) for one hyperperiod $\mathcal{HP}$ of the task set $\mathcal{T}$. The hyperperiod of $\mathcal{T}$ is the least common multiplier of all task periods, i.e., $\mathcal{HP} = 60$ in this case, and is the minimal time interval that the task execution pattern repeats. In other words, the system schedule is identically repeating after one hyperperiod. The gray rectangles represent the normal task executions including the non-leakage parts of AES, and the red rectangles depict the leakage points. Assuming that the attacker measures the μP power at each time unit, she will obtain 60 discrete power values, each of which corresponds to a specific operation, e.g., task execution or AES, retrievable from the simulated schedule in Fig. 2 (a). She then divides the whole obtained power trace into 3 individual samples based on $T_2 = 20$ to align the samples (as depicted in Fig. 2 (b)). To highlight the leakage points, we do not show the non-leakage executions (the gray rectangles), since they are independent from the leakages. After the simulated hyperperiod, the same schedule will repeat. Therefore, we can find that the leakage occurs at relative time $\hat{t} = 13$ with the highest probability $\hat{p} = \frac{2}{3} = 0.67$. By now, we can calculate the robustness of the key K_2 using Eq. 2, i.e., $\mathcal{R}_2 = 180$.

However, this does not seem to be a satisfying solution. An obvious alternative of scheduling the tasks is, for instance, to exchange the execution order of the first leakage point at $t = 13$ with the first fraction of τ_3 from $t = 14$ to $t = 19$ as shown in Fig. 3 (a). If we align the samples as explained just now, we would get the result as in Fig. 2 (b). We can observe that, now, the three leakage points occur at three different times, i.e., $t = 20$, $t = 16$, and $t = 13$. However, this schedule repeats after one hyperperiod, making $\hat{p} = \frac{1}{3} = 0.33$. Then the robustness of K_2 is $\mathcal{R}_2 = 620$.

An even better solution will be an enhanced random scheduler, which can both guarantee the deadlines and reduce the value of $\hat{p}$. Fig. 4 (a) illustrates the simulation of the system with such a scheduler under the first hyperperiod $[0, \mathcal{HP})$. As can be noticed, the leakage points occur at three different times, respectively. Furthermore, due to the randomness introduced by the scheduler, the simulation of the next hyperperiod (shown in Fig. 4 (b)) gives a different schedule than Fig. 4 (a). Thereby, if we align the two hyperperiods, we can get the samples as in Fig. 4 (c). In fact, in order to get the actual $\hat{p}$, we need to simulate the system execution to an extended timespan, and, consequently, we would get $\hat{p} = 0.11$ which leads to $\mathcal{R}_2 = 5460$.

V. Proposed Scheduler

In this section, we present our EDF-based SPARTA scheduler and related theoretical guarantees. We use the following notations to describe the scheduler:

- $\mathbf{L_s}$: An ordered list of active tasks within time-interval $[T_s, T_{s+1})$.
- $\mathbf{L'_s}$: A list of tasks picked by SPARTA to be executed within time-interval $[T_s, T_{s+1})$. Note that $L'_s \subseteq L_s$.

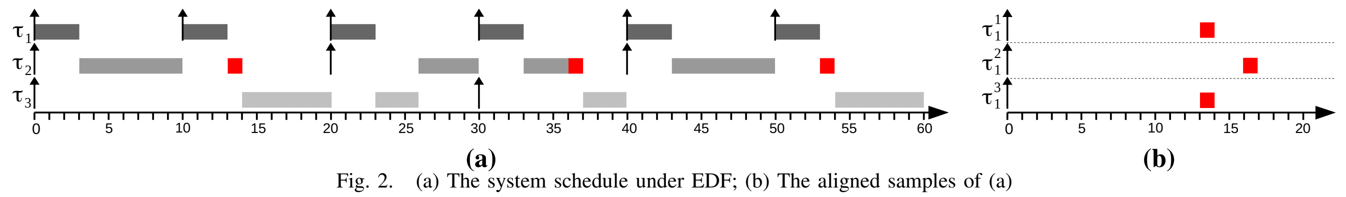
Fig. 2. (a) The system schedule under EDF; (b) The aligned samples of (a)

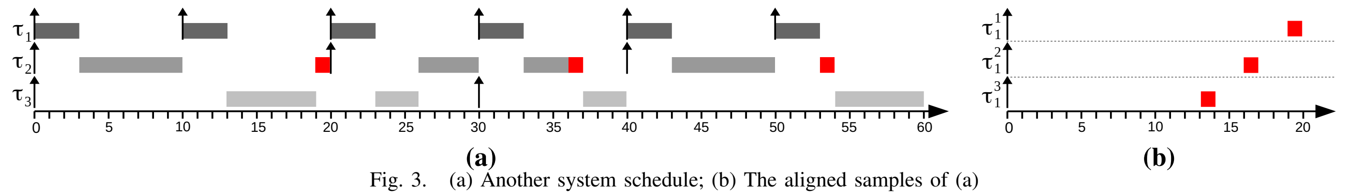
Fig. 3. (a) Another system schedule; (b) The aligned samples of (a)

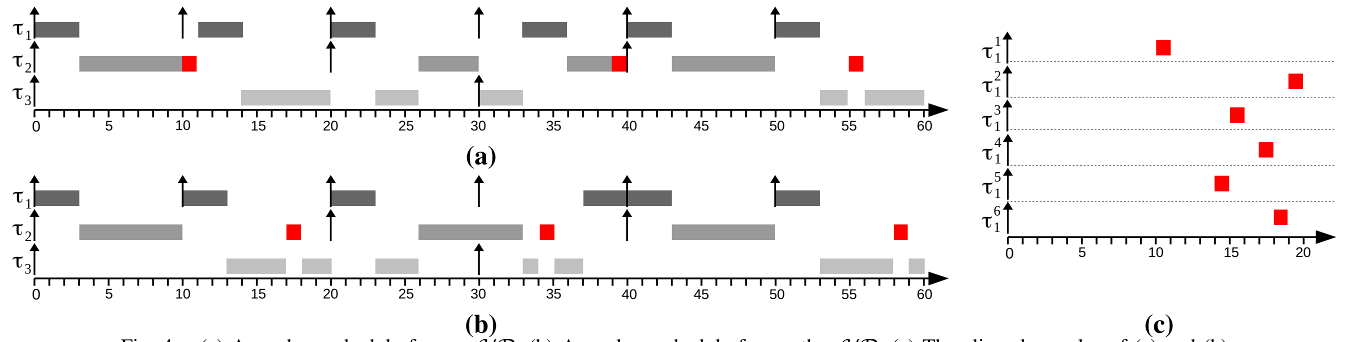
Fig. 4. (a) A random schedule for one $\mathcal{HP}$; (b) A random schedule for another $\mathcal{HP}$; (c) The aligned samples of (a) and (b)

- $\mathbf{rt_i}$: The execution time of a task τ_i to be carried out within a given time-interval.
- $\mathbf{et_i}$: The execution time of a task τ_i, ignoring the execution time of its leakage point, to be carried out within a given time-interval, and $et_i = rt_i - \delta$
- $\mathbf{t'_i}$: The time instant where the leakage of task τ_i starts.
- $\mathcal{X}_\mathbf{i}$: The time-interval of the leakage for task τ_i, i.e., $\mathcal{X}_i = [t'_i, t'_i + \delta)$.
- $\mathbf{Z_s}$: An ordered list of leakage points within $[T_s, T_{s+1})$.

In the following, we describe the workflow of SPARTA:

1) The scheduler is activated at each *job arrival*. The time of the s-th activation of SPARTA is denoted as T_s. At time T_s, the scheduler considers the set of tasks to be executed within time-interval $[T_s, T_{s+1})$, and then decides a concrete schedule of these tasks to help thwart DPA attacks.
2) Unlike the EDF policy, which, at each *task finish*, executes the task that is closest to its deadline, SPARTA operates on a set of tasks that will be executed within $[T_s, T_{s+1})$. It first sorts the active tasks at time T_s based on the proximity to their respective deadlines. The resulting sorted list is captured by L_s and the first entry of L_s contains the task closest to its deadline.
3) The scheduler walks through L_s, starting from its first entry. The scheduler continues picking a task from L_s, until processor bandwidth within $[T_s, T_{s+1})$ is fully utilized, or L_s becomes *empty*. The list of tasks picked by the scheduler is kept into the list L'_s and rt_i captures the execution time of τ_i to be carried out within $[T_s, T_{s+1})$. Therefore, once the scheduler finishes walking through L_s, the following relationship must hold:
$$\left(\sum_{\tau_i \in L'_s} rt_i = T_{s+1} - T_s\right) \vee (L_s = \phi)\,. \qquad (3)$$
4) In this phase, SPARTA decides a concrete schedule of tasks by first assigning leakage points at different time instants. et_i captures the execution time of task τ_i to be carried out within $[T_s, T_{s+1})$, ignoring the execution time of its leakage point. Therefore, $et_i = rt_i - \delta$ (recall that δ is the execution time of a leakage point), for tasks with leakage points, and $et_i = rt_i$, otherwise. Our scheduler randomly picks a task τ_{x_1}, which has leakage point, from L'_s and randomly allocates the leakage point at time t'_{x_1}. In order to preserve the timing constraints, t'_{x_1} must be within the time interval $[T_s + et_{x_1}, T_{s+1})$. Once t'_{x_1} is assigned, the scheduler randomly selects another task τ_{x_2} having leakage point and randomly assigns its leakage point at time t'_{x_2}, where t'_{x_2} satisfies the following condition:
$$t'_{x_2} \in \begin{cases} [T_s + rt_{x_1} + et_{x_2}, T_{s+1}), \\ \quad \text{if } t'_{x_1} \le T_s + et_{x_1} + rt_{x_2} \\ [T_s + et_{x_2}, T_{s+1}) \backslash \mathcal{X}_{z_1}, \\ \quad \text{otherwise.} \end{cases} \qquad (4)$$
In general, SPARTA keeps an ordered list Z_s for the allocated leakage points. Z_s is ordered based on the time of occurrence of a leakage point and the first entry of Z_s holds the leakage point occurring the *earliest* in time. Let us assume n leakage points have been assigned (*i.e.* $|Z_s| = n$) and these leakage points are ordered in Z_s as follows: $t'_{z_1} < t'_{z_2} < \ldots < t'_{z_{n-1}} < t'_{z_n}$. To assign the leakage point of a task $\tau_{x_{n+1}}$ (from L'_s), we choose

$t'_{x_{n+1}}$, satisfying the following criteria:

$$t'_{x_{n+1}} \in \begin{cases} [T_s + \sum_{i=1}^{n} rt_{z_i} + et_{x_{n+1}}, T_{s+1}), \\ \quad \text{if } t'_{z_n} \leq T_s + \sum_{i=1}^{n-1} rt_{z_i} + et_{z_n} + rt_{x_{n+1}} \\ [T_s + \sum_{i=1}^{n-1} rt_{z_i} + et_{x_{n+1}}, T_{s+1}) \backslash \mathcal{X}_{z_n}, \\ \quad \text{if } t'_{z_n} > T_s + \sum_{i=1}^{n-1} rt_{z_i} + et_{z_n} + rt_{x_{n+1}} \\ \quad \wedge t'_{z_{n-1}} \leq \sum_{i=1}^{n-2} rt_{z_i} + et_{z_{n-1}} + rt_{x_{n+1}} \\ \ldots \\ [T_s + et_{x_{n+1}}, T_{s+1}) \backslash \bigcup_{i \in [1,n]} \mathcal{X}_{z_i}, \text{ otherwise.} \end{cases} \quad (5)$$

This step ends when all the leakage points are allocated.

5) Once all the leakage points are assigned, we try to glue the task executions as tight as possible to reduce preemptions. Let us assume that the list Z_s is ordered as follows: $t'_{z_1} < t'_{z_2} < \ldots < t'_{z_{l-1}} < t'_{z_l}$, where $|Z_s| = l$. Our scheduler picks τ_{z_1} and assigns the remaining execution time et_{z_1} directly before t'_{z_1}. Similarly, the remaining execution time of τ_{z_2} (*i.e.* et_{z_2}) is alloted before t'_{z_2}. Once the remaining execution time of all the tasks in Z_s has been allocated, the tasks in $(L'_s \setminus Z_s)$, that is, tasks without leakage points, are considered. In particular, each such task is scheduled by greedily occupying the first available time slots.

As can be noticed, we consider each task to be associated with at most one message, and the leakage point always occurs at the end of the task execution. However, it is worthwhile to mention that the scheduler can be extended to support the situations where a task is associated with multiple messages, and the leakage point occurs at an arbitrary position in the execution. Both aspects can be solved by dividing the task into subtasks based on occurrences of leakage points. Subsequently, the assignments of leakage points can be carried out on these subtasks similar to Eq. 5.

A. Schedulability Guarantee

SPARTA satisfies the following crucial property.

Property 5.1: Let us assume that Z_s holds l entries after all leakage points have been assigned and the list Z_s is ordered as follows: $t'_{z_1} < t'_{z_2} < \ldots < t'_{z_{l-1}} < t'_{z_l}$, where $|Z_s| = l$. Our scheduler guarantees that for all $i \in [1, l]$, the remaining execution time et_{z_i} of task τ_{z_i} is assigned before t'_{z_i}.

Proof: To prove for the general case, we consider an arbitrary leakage point t'_{z_n}. Let us distinguish between the following scenarios to show that the assignment of t'_{z_n} is *safe*.

- **Case I:** All the tasks $\tau_{z_i, \forall i \in \{1,2,\ldots,n-1\}}$ have been processed from list L'_s before task τ_{z_n} (*cf.* step 4 of SPARTA).
  - **Subcase A:** $t'_{z_n} > t'_{z_{n-1}}$, *because* t'_{z_n} *cannot be placed before any* t'_{z_i}, *where* $i \in [1, n-1]$. From Eq. 5, we can observe that this scenario must happen for the first case, i.e., when the lower bound of t'_{z_n} is at least $T_s + \sum_{i=1}^{n-1} rt_{z_i} + et_{z_n}$. This directly implies the fact that the non-leakage part of τ_{z_n} can finish before t'_{z_n}. This is because of the assignment via Eq. 5, as $t'_{z_n} \geq T_s + \sum_{i=1}^{n-1} rt_{z_i} + et_{z_n}$.
  - **Subcase B:** $t'_{z_n} > t'_{z_{n-1}}$, *but* t'_{z_n} *could have been placed before some* t'_{z_i}, *where* $i \in [1, n-1]$, *according to Eq. 5.* This means t'_{z_n} might, at least, be assigned before $t'_{z_{n-1}}$, according to Eq. 5. Based on the constraints starting from the second condition in Eq. 5, we can observe that $t'_{z_{n-1}} > T_s + \sum_{i=1}^{n-2} rt_{z_i} + et_{z_{n-1}} + rt_{z_n} = T_s + \sum_{i=1}^{n} rt_{z_i} - \delta$. From our hypothesis, we have $t'_{z_n} > t'_{z_{n-1}}$. Therefore, $t'_{z_n} > t'_{z_{n-1}} > T_s + \sum_{i=1}^{n} rt_{z_i} - \delta$. This also implies directly that the assignment of t'_{z_n} is safe.
- **Case II:** There is at least one task τ_{z_i} ($i \in [1, n-1]$) that was processed from list L'_s *after* task τ_{z_n} (*cf.* step 4 of SPARTA). Consider the last task, say τ_{z_y} ($y \in [1, n-1]$), with leakage point assigned within $[T_s, t'_{z_n})$. The reason why t'_{z_y} can be assigned before t'_{z_n} is that there is sufficient processor bandwidth left for τ_{z_y} before t'_{z_n}. From the constraints of Eq. 5, we know this is possible only if $t'_{z_n} > T_s + \sum_{i \in \{1,2,\ldots,n-1\} \backslash \{y\}} rt_{z_i} + et_{z_n} + rt_{z_y}$. Therefore, $t'_{z_n} > T_s + \sum_{i}^{n} rt_{z_i} - \delta$. This proves that the assignment of t'_{z_n} is *safe*. ■

Schedulability test: From Property 5.1, we can state that SPARTA guarantees the assignments of leakage points to be *safe*. This means, within a time-interval $[T_s, T_{s+1})$, SPARTA guarantees to schedule all tasks from the list L'_s, which is the same set of tasks (along with their respective execution time to be carried out within the time-interval $[T_s, T_{s+1})$) picked by a classic EDF scheduler. Since the choice of $[T_s, T_{s+1})$ is arbitrary, we can conclude that the schedulability test for SPARTA is exactly the same as of EDF.

B. Upper-bound of Context Switches

Property 5.2: The total number of system context switches[2] introduced by SPARTA is at most 2 times of that implied by EDF. The proof is skipped due to space limit.

C. Complexity of SPARTA

Property 5.3: The complexity of each SPARTA activation is $O(n^2)$. The proof is skipped due to space limit.

VI. Experimental Results

In order to evaluate the achieved protection strength against DPA attacks enhanced by SPARTA, we have performed experimental evaluations under two representative criteria, i.e., different processor utilization levels and different problem sizes (different amount of tasks). Each task τ_i is associated with at most one message m_i with length $l_i = 1$. The task periods are chosen uniformly at random from $\{200, 400, ..., 2000\}$ to avoid simulation of long hyperperiods. On each problem size, the task execution times were randomly generated under the current utilization level.

We have implemented SPARTA in C, and conducted experiments under a simulation environment on a Linux machine

[2]We refer to the start of a task execution or continuation of a task after being interrupted for some time as a context switch.

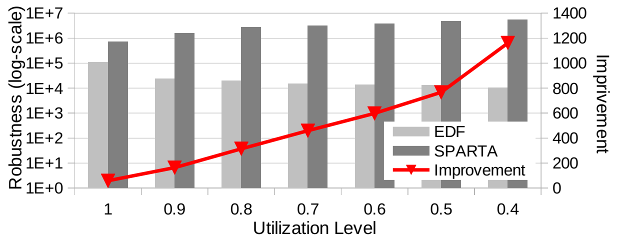

Fig. 5. Different Processor Utilizations

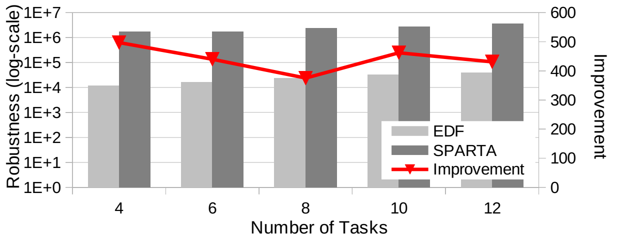

Fig. 6. Different Problem Sizes

with a quad-core Intel Xeon processor and 8GB of memory. For illustration purposes, we compared the protection strength delivered by SPARTA with protection provided by EDF. We define the robustness of a set of experiments as the average achieved robustness over all secret keys, i.e.,

$$\overline{\mathcal{R}} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{|\mathcal{M}_i|}\sum_{m_j\in\mathcal{M}_i}\mathcal{R}_j, \tag{6}$$

where n is the number of experiments, and $\mathcal{M}_i$ is the set of messages in the i-th experiment. $\mathcal{R}_j$ is calculated according to Eq. 2. The improvement achieved by SPARTA over EDF is

$$\overline{\mathcal{I}} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{|\mathcal{M}_i|}\sum_{m_j\in\mathcal{M}_i}\frac{\mathcal{R}_j^{SPARTA}-\mathcal{R}_j^{EDF}}{\mathcal{R}_j^{EDF}}. \tag{7}$$

The two evaluation scenarios are presented as follows.

A. *Different Processor Utilizations*

In this set of experiments, we would like to study the influence of different processor utilizations on achieved protection strength. We conducted experiments with different processor utilization levels on the same problem size. That is, all experiments are with 6 tasks. The studied processor utilization levels are $U \in \{1, ..., 0.5, 0.4\}$, and on each level, 200 random test applications were generated with different task periods and execution times. The results are presented in Fig. 5. The light gray and dark gray bars show the average robustness $\overline{\mathcal{R}}$ of EDF and SPARTA, respectively, with corresponding setups, and are aligned to the primary y-axis. The red line indicates the improvement $\overline{\mathcal{I}}$ of each set of experiments, and is aligned to the secondary y-axis on the right. We can observe that the achieved improvements are gradually increasing as processor utilization drops (indicated by the red line). We can also notice that SPARTA dominates EDF on all utilization levels, and it works better, when the processor is less loaded, due to the larger amount of free time slacks that it can operate on. Therefore, SPARTA has more flexibility in allocating leakage points when U is low.

B. *Different Problem Sizes*

We now evaluate the performance of SPARTA on different problem sizes, i.e., systems with different amount of tasks. There are in total five sizes studied, namely, $|\mathcal{T}| \in \{4, 6, ..., 12\}$. On each size, we randomly generated 200 test applications (having different task periods and execution times) with utilization $U = 0.7$. The obtained results are shown in Fig. 6, and the same representations are shared with the previous subsection. From Fig. 6, we can find that SPARTA delivers nearly a constant improvement level (indicated by the red line) compared with EDF, and the absolute delivered protection is higher on larger problem sizes. It is also worth mentioning that, for individual cases, SPARTA delivered better protections for the AES secret keys in all the instances of experiments, and the average improvements is $\overline{\mathcal{I}} = 441$.

VII. Conclusions

In this paper, we presented the first real-time scheduling policy that thwarts DPA attacks. SPARTA can both guarantee the deadline constraints, and, at the same time, provides extensive enhancement to security of underlying AES implementations. As demonstrated in the formal proofs, SPARTA shares the same guarantee of system schedulability as the optimal EDF policy. We have conducted experiments to evaluate the protection strength of SPARTA comparing with EDF under different evaluation criteria.

Acknowledgements

We would like to thank Łukasz Chmielewski for sharing valuable experience of DPAs.

References

[1] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer, 2011, vol. 24.
[2] W. Jiang, K. Jiang, and Y. Ma, "Energy Aware Real-Time Scheduling Policy with Guaranteed Security Protection," in *Asia and South Pacific Design Automation Conference (ASPDAC)*, 2014.
[3] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *Security and Privacy (SP)*. 2010, pp. 447–462.
[4] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *CRYPTO*, 1999, pp. 388–397.
[5] K. Jiang, P. Eles, and Z. Peng, "Optimization of Secure Embedded Systems with Dynamic Task Sets," in *DATE*, 2013.
[6] J. Daemen and V. Rijmen, *The design of Rijndael: AES–the advanced encryption standard*. Springer, 2002.
[7] J. Nechvatal, E. Barker, L. Bassham, W. Burr, and M. Dworkin, "Report on the Development of the Advanced Encryption Standard (AES)," DTIC Document, Tech. Rep., 2000.
[8] A. G. Bayrak, N. Velickovic, F. Regazzoni, D. Novo, P. Brisk, and P. Ienne, "An eda-friendly protection scheme against side-channel attacks," in *DATE*, 2013, pp. 410–415.
[9] J.-S. Coron and I. Kizhvatov, "An efficient method for random delay generation in embedded software," in *CHES*. Springer, 2009.
[10] 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)*, July 20145.
[11] G. E. Suh, D. Clarke, B. Gassend, M. Van Dijk, and S. Devadas, "Aegis: architecture for tamper-evident and tamper-resistant processing," in *International conference on Supercomputing*. 2003, pp. 160–171.
[12] S. B. Ors, F. Gurkaynak, E. Oswald, and B. Preneel, "Power-Analysis Attack on an ASIC AES implementation," in *International Conference on Information Technology: Coding and Computing (ITCC)*,2004.
[13] K. Tiri, M. Akmal, and I. Verbauwhede, "A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards," in *European Solid-State Circuits Conference (ESSCIRC)*, 2002, pp. 403–406.
[14] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer, 2007.