Performance monitor counters: Interplay between safety and security in complex cyber-physical systems

Performance Monitor Counters: interplay between safety and security in complex Cyber-Physical Systems

Alberto Carelli, *Student Member, IEEE,* Alessandro Vallero, *Member, IEEE,*
and Stefano Di Carlo, *Senior Member, IEEE*

***Abstract*—Recent years have witnessed the growth of the adoption of Cyber-Physical Systems (CPSs) in many sectors such as automotive, aerospace, civil infrastructures and healthcare. Several CPS applications include critical scenarios, where a failure of the system can lead to catastrophic consequences. Therefore, anomalies due to failures or malicious attacks must be timely detected. This paper focuses on two relevant aspects of the design of a CPS: safety and security. It analyzes in a specific scenario how the Performance Monitor Counters (PMCs) available in several commercial microprocessors can be from the one hand a valuable tool to enhance the safety of a system and, on the other hand, a security backdoor. Starting from the example of a PMC based safety mechanism, the paper shows the implementation of a possible attack and eventually proposes a strategy to mitigate the effectiveness of the attack while preserving the safeness of the system.**

***Index Terms*—hardware security, safety, performance monitoring counters, cyber-physical systems.**

I. Introduction

CYBER-PHYSICAL Systems (CPSs) are the root of a fourth industrial revolution [1]. *"Cyber-physical systems are physical and engineered systems whose operations are monitored, coordinated, controlled and integrated by a computing and communication core"* [2]. A CPS integrates processing units, sensors and actuators, enabling the interaction of the computing infrastructure with the physical world. The Internet of Things (IoT) forms a foundation for this cyber-physical systems revolution [1]. All devices of a CPS and different CPSs are interconnected in order to create a network enabling billions of systems and devices to interact and share information. Thanks to their ability of transforming traditional processes by integrating technologies from various sectors, CPSs are bringing innovation to many industries including: automotive and aerospace, chemical processes, smart energy and water grids, healthcare, manufacturing and transportation [2]–[5]. Several of these application domains involve the control of critical infrastructures providing services that constitute the technological backbone of our society [6]. These infrastructures can damage themselves, people, or properties when they are improperly used [7], and this improper use can be either the result of a failure of one of their components or an intentional attempt to corrupt their behavior.

A. Carelli, A. Vallero and S. Di Carlo are with the Department of Control and Computer Engineering, Politecnico di Torino, Torino, 10129 Italy, e-mail: stefano.dicarlo@polito.it.

Safety and security are therefore two critical properties of every CPS, both sharing identical goals: protecting the CPS from hazards due to accidental failures (safety) or due to intentional attacks (security) [8], [9]. In this context, there is a recognized request to consider them under a unified view when designing and operating complex CPSs [7], [8], [10], [11]. This is particularly important every time a security mechanism may negatively impact the safety of the system or vice versa [12].

This paper analyzes in a specific scenario how the Performance Monitor Counters (PMCs) available in several commercial microprocessors may have severe implications on the interplay of safety and security of a CPS.

PMCs can be used for several purposes including performance modeling and optimization, debugging, benchmarking, and in-field monitoring (see Section II). This paper is particularly interested in this last category of uses. The integration of computing and physical elements in a CPS introduces a vast range of design and operational constraints. Among them, CPSs often require to operate under real-time constraints [7]. PMCs are an effective instrument to detect timing violations in multiprocessor systems. The timing of different tasks can be profiled by recording time related PMCs (e.g., the Clock Cycle Counter - CCC and the L1 Data Cache-Miss counter - DCM in the Intel architectures) over several executions in order to build a model of the behavior of the system. This can be done either by setting simple thresholds [13]–[15] or by exploiting machine learning models [16]. The model can then be used at run-time to detect anomalies.

However, time related PMCs have been exploited to perform different classes of attacks (see Section II). These PMCs can be used to implement the side-channel attack described by Bonneau and Mironov in [17], which is able to discover the secret key of the Advanced Encryption Standard (AES) cipher.

This paper proposes a mitigation strategy able to increase the complexity of this attack and discusses its interplay with the effect on a selected safety mechanism. Among the different safety techniques against timing violations, for its simplicity, this paper focuses on the methodology proposed by Esposito et al. in [13]. This technique builds a simple timing model for different tasks based on two thresholds.

The proposed attack mitigation technique is based on the application of a PMC poisoning schema. The poisoning alters

the value distribution of the PMCs in such a way to harden the work of the attacker while preserving those properties that allow detecting timing violations in the system.

This paper extends the preliminary results published by the authors in [18] by presenting a generalized technique able to protect both the CCC and the DCM counter.

Based on the concepts presented in this paper, different poisoning schemas can be derived in order to work with different monitoring techniques. However, it is hard to generalize the impact of the poisoning on each schema. This must be evaluated case by case and is out of the scope of this paper.

A set of experiments was carried out to evaluate the impact of the proposed attack mitigation technique on the safety and security of a sample node executing different tasks. Experiments consider both synthetic tasks hard to monitor and tasks taken from the MiBench benchmarks [19]. The node uses the AES cipher (victim of the attack) to encrypt information. MiBench benchmarks were selected since they implement a set of typical algorithms employed in several embedded systems and have been also used in several reliability evaluation studies [20]–[23]. Results show the capability of the proposed technique in increasing the complexity of the attack considering both the CCC and the DCM counters, while introducing a low impact on the timing violation detection capability of the system. The technique proposed in this paper outperforms results published in [18] both in terms of better security and reduced impact on the safety of the system.

The paper is organized as follows: Section II overviews related work on the use of PMCs both in the safety and security domain, while Section III describes the architecture of the considered CPS. Section IV introduces the proposed attack mitigation technique and Section V reports the validation results. Finally, Section V-C concludes this paper.

II. Background

PMCs are special registers available in most microprocessor architectures. They allow monitoring of several classes of events including branch predictions, cache hits/misses and process timing. Monitoring of events through PMCs has a variety of uses in application development, including performance modeling and optimization, debugging, and benchmarking [24]–[27]. The privileges required to access these counters depend on both the processor architecture and the operating system (OS). However, during application development, the designer has usually full control on the development environment and a full access to the available PMCs does not represent a security threat as considered in this paper.

When moving to in-field applications, access to PMCs requires to carefully consider the interplay between safety and security.

A set of applications evaluate predefined PMCs signatures in order to detect failures or attacks. PMCs can be exploited as a valuable tool for the development of software based self test (SBST) routines [28]. The capability of monitoring cache-misses through PMCs (DCM counter) was used in [29] to perform SBST of cache memories. In this type of approaches, an exact value of the counter is expected at the end of the test routine. PMCs signatures were successfully exploited to detect firmware modifications in a CPS controlling a power grid in [30]. For this class of PMC uses, poisoning techniques such as the ones presented in this paper cannot be applied, since they would prevent the deterministic behavior of the counters for selected processes.

Nevertheless, there is a large class of techniques that try to build models of the behavior of the system based on statistical off-line PMC profiling. These models can be used in-field to detect different types of anomalies.

Xia et al. employed the PMCs to monitor the control-flow integrity of software applications [31]. By analyzing the branch instructions behavior, they were able to detect deviations from the correct control flow.

Yilmaz [32] proposed a technique for faults localization in software applications. By off-line profiling the number of executed instructions considering correct and faulty executions, they were able to build a behavioral model of the software able to detect in-field applications differentiating from the model. A similar technique from the same authors based on the monitoring of the function execution time was also presented in [33].

The authors of [14] used the PMCs to estimate the Worst-Case Execution Time (WCET) for safety-critical applications.

In [15], WCET-aware Performance Monitoring Units were proposed for safety certification in the automotive domain.

In [13] PMCs were used to detect faults causing deadline violations in multi-core systems.

A set of 10 PMCs was analyzed in [16] to build a machine learning model based on Supported Vector Machines (SVM) able to detect different types of anomalies.

In this category of approaches, poisoning techniques such as the one presented in this paper can potentially be applied. Nevertheless, the actual impact of the poisoning strongly depends on the target technique and it is hard to generalize. For this reason this paper focuses on a selected technique presented in [13] to show the interplay between safety and security in a specific case.

When considering the use of PMCs to perform attacks, most related publications focus on the cache behavior during encryption with the AES algorithm. Bernstein was able to remotely recover the complete AES key exploiting timing information related to cache accesses [34]. Bounneau and Mironov performed a similar attack with a reduced number of samples to recover the AES key when applied to Intel architectures [17]. PMCs were employed as source of side-channel information also to attack encryption algorithms on AMD platforms in [35]. Side-channel attacks were possible also for asymmetric key cryptography, as reported in [36]. The attack, carried out on Intel platforms, targeted a 1024 bit key of RSA and exploited the monitoring of branch-miss events.

Proper defense measures can be taken if the attack is detected. The authors of [37] proposed a generic detection mechanism, using a pre-trained classifier, able to deal with a variety of micro architectural side-channel attacks, including also cache-based attacks. However, rather than reacting to an attack, it is important to work on proactive techniques able to prevent or increase the complexity of the attack.

Our previous work proposed a first attempt to protect the CCC counter against the Bounneau and Mironov attack taking into account the impact on the safety of the system [18]. However, as analyzed in this paper, the same technique is unable to properly protect the DCM counter. This paper moves forward by presenting a generalized technique able to protect both counters with a reduced impact on the safety of the system.

III. CPS Architecture

A. CPS and node architecture

Fig. 1 shows the general CPS architecture considered in this paper, which is a typical *Supervisory Control and Data acquisition* (SCADA) system. *Computing nodes* perform local computations and are responsible for controlling a network of sensors and actuators managing the operations of the physical infrastructure. Special *monitoring nodes* coordinate the work of a set of computing nodes by constantly exchanging data and information with them. Depending on the application, CPSs provide a wide set of specifications against which the systems must operate. Several manufacturing plants employ nodes based on low-end microprocessors either running bare-metal applications or old desktop operating systems [7]. However, the increasing complexity of the controlled infrastructures is quickly moving these systems toward more complex microprocessor architectures [38].

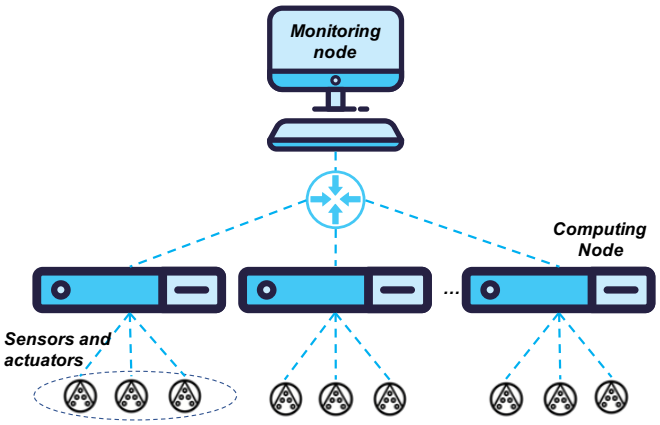


Fig. 1: Generic architecture of the considered CPS.

Fig. 2 shows the conceptual architecture of a generic computing node.

The node runs an operating system providing a set of services required to accomplish different application tasks. The number and type of tasks depends on the available sensors/actuators and on the function the node has to implement. Moreover, tasks must be executed under given timing constraints [7]. Overall, the goal of the executed tasks is to acquire data from sensors, elaborate raw data and encrypt/decrypt payloads to communicate with other nodes. Data exchanged with external nodes (e.g., monitoring nodes) are encrypted with a symmetric key by an appropriate service module integrated in the OS. In general, any task running at the application level can request the OS services.

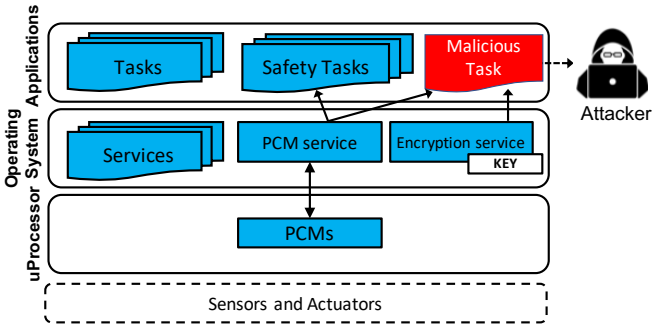


Fig. 2: Generic architecture of the considered computing node.

B. Safety task

As discussed in Section II, different safety mechanisms to control the correct operation of the node can be implemented as additional safety tasks at the application level (Fig. 2). The safety task considered in this paper (based on the technique proposed in [13]) exploits PMCs profiling and on-line monitoring to detect faults causing abnormal execution times of a task. Overall, the considered safety technique consists of two phases named: (i) *off-line phase* and (ii) *on-line phase*.

During the off-line phase, each task is profiled over several executions in order to analyze the distribution of the values assumed by the considered PMCs. Two PMCs strictly related to the execution time of an application are considered in this study: the Clock Cycle Counter (CCC) and the L1 Data Cache-Miss counter (DCM).

In principle, the value of a PMC for a given task with given inputs and execution environment, should provide deterministic and reproducible values. However, in complex multi-core systems as the ones considered in this paper, several complex often-unknown HW/SW interactions that are hard to predict (e.g., out-of-order execution models in which instructions are executed in a non-deterministic order, the memory hierarchy featuring different levels of cache memories that determine non-deterministic data access profiles, bus arbitration and in general all controllers, etc.) may introduce variability in the PMC values. Different inputs across different executions of a task are another source of variability of the observed PMCs. Therefore, the values measured for a PMC across different executions can be considered as a random variable X, characterized by an empirical Cumulative Density Function (CDF), $F_X(x)$ (see Fig. 3).

Based on the collected profiles, each task can be associated to three operating areas reflecting the state of the system: *safe area*, *critical area* and *warning area*. Two thresholds are defined to separate the aforementioned operating areas: W_{TH} and C_{TH} (see Fig. 3). These thresholds are chosen by selecting the confidence level for the warning area (C_W) and for the critical area (C_C). Confidence levels can be chosen arbitrarily during the design phase. Strict confidence levels increase the number of false positives and the performance overhead due to a larger number of recovery operations. Wide levels may not detect all failures of the system. In details, W_{TH} and C_{TH} can be computed by solving the following inequalities looking

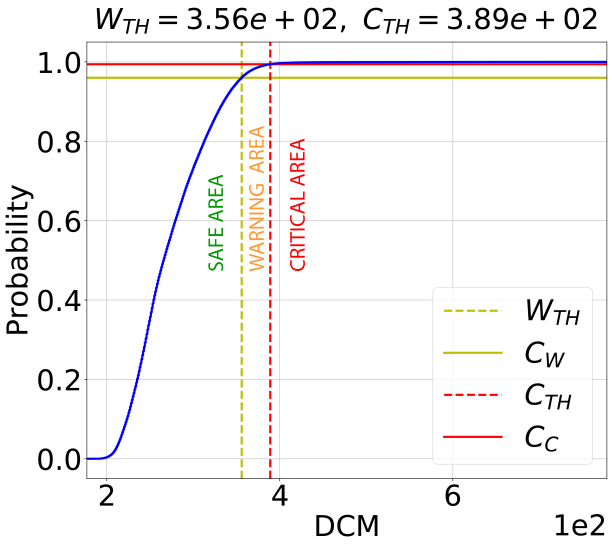


Fig. 3: CDF of DCM counter of the Susan-Smoothing MiBench benchmarks [19] computed profiling 100,000 executions of the task.

at the collected profiles:

$$P(X > W_{TH}) < C_W \Rightarrow F_X(W_{TH}) > 1 - C_W \quad (1)$$

$$P(X > C_{TH}) < C_C \Rightarrow F_X(W_{TH}) > 1 - C_C \quad (2)$$

During the on-line phase, the safety task monitors the PMCs of every task in order to determine in which area the execution can be mapped. The task state is then classified as follows:

- *safe direct*: if the value of the PMC is below the warning threshold.
- *critical direct*: if the value of the PMCs is above the critical threshold. In this case a recovery action must be issued;
- *warning*: if the value of the PMCs is between the warning and the critical threshold. In this case, if this conditions is detected α consecutive times the task is marked as *critical warning*, otherwise it is classified as *safe warning*.

The choice of α is related to the probability that the system is in a *safe* state after α consecutive *warning* classifications denoted as $P(FP_\alpha)$. $P(FP_\alpha)$ is obtained during the process profiling of the off-line phase. According to [13] α can be calculated as:

$$\alpha = \frac{ln(1 - P(FP_\alpha))}{ln(F(C_{TH}) - F(W_{TH}))} \quad (3)$$

C. Attack model

Securing the CPS architecture presented in Fig. 1 is an important task. This paper focuses on an attacker interested in recovering the AES encryption key of a node to carry out malicious actions. We suppose the attacker is not interested in denial-of-service attacks, because they disrupt the offered services and prevent the control of the CPS. A successful attack on a node may spread the infection to every node, thus compromising the whole system. In the case all nodes share the same secret key, the whole system would be immediately compromised when a single node is compromised. When the secret key is different for every node, the same malicious task could target another node and the attack could be repeated until all nodes composing the CPS are under control of the attacker.

Looking at the architecture of the node reported in Fig. 2, we assume that enough effort has been carried out to secure the hardware OS level of the node. This includes securing the secret key and the encryption/decryption service. Nevertheless, we assume that the attacker may exploit user level vulnerabilities to inject a malicious task (e.g., a virus or a malware) within a node. The attack could be undertaken on a specific node because the attacker could have gained physical access to it, or because that specific node offers unique vulnerabilities. The malicious task is a user application that can exploit the computational resources of the node as well as the services offered by the OS of the node. Therefore, the attacker can probe the PMCs, trigger the encryption process and send the information to a remote entity that can process them off-line in order to perform the attack.

Despite PMCs are a force point from the safety perspective, they represent a weak point from the security standpoint. They expose the system to *timing attacks*, a category of side-channel attacks [34]. PMCs can therefore be considered as a double edged weapon. This paper focuses on the side-channel attack presented by Bonneau and Mironov in [17], which targets the AES encryption algorithm.

The theory exploited to perform the attack is based on the concept of *cache-collisions* during the final round of the AES encryption cypher.

For performance reasons, the algebraic operations of a software-implemented AES cypher are combined in precomputed values stored in different lookup tables. Thus, the encryption can be considered as a sequence of table lookups. As all data of a program, these tables are loaded in the L1 data cache memory during the encryption process. If the data is already loaded in the cache memory, a lookup produces a cache-hit. On the other hand, when the data cannot be found in the cache memory, the lookup generates a cache-miss, which will take on average more time to be served since it requires to access data from a slower memory level. Depending on the organization of the cache, each cache line may store multiple table entries. A *cache-collision* occurs when a pair of different lookups targets the same cache line and for sure does not generate a cache-miss.

Being able to detect when a collision occurs is important for the attacker, since it gives a clue of which element of the table has been accessed.

Let us denote with i and j the index of two bytes of a generic 16 bytes ciphertext ($0 \leq i, j \leq 15$), with c_i and c_j their respective values, and let us consider the encryption time (i.e., CCC counter) as available information. As described in [17], the first goal of the attacker is to record in a data structure the timing data for random ciphertexts (samples) for several pairs (c_i , c_j). The data structure is organized as follows:

$$t[i, j, \Delta] = CCC_k \quad (4)$$

where $\Delta = c_i \oplus c_j$ and CCC_k is the encryption time of the k^{th} collected sample. If multiple data are collected for the same entry of the table, the average time is recorded. According to [17], if a cache-collision occurs the following equality becomes true:

$$k_i \oplus k_j = c_i \oplus c_j \tag{5}$$

As discussed before, when a collision occurs, the encryption time should be significantly lower that the cases in which there is no collision. To succeed in the attack, the attacker has to find one value $\Delta'_{i,j}$ for each i, j such that:

$$t[i, j, \Delta'_{i,j}] < \overline{CCC} \tag{6}$$

where $\overline{CCC}$ is the average encryption time over all collected samples. We denote this as *collision condition*. In this case $\Delta'_{i,j}$ becomes an accurate guess for the true value $k_i \oplus k_j$. This reduces the space of possible values of $k_i \oplus k_j$ that an attacker has to test to recover the key. Several further optimizations are proposed by [17] to reduce the number of samples required to perform the attack. Nevertheless, the general concept of the attack remains the same.

When using the CCC timer, the attacker exploits the timing as an indirect measure of the number of cache-misses. The correlation between the CCC timer and the number of cache misses is the key factor. In the Pentium 3 processor analyzed by [17] this correlation was very high and therefore a low number of samples (2^{16}) was enough to perform the attack. With a more complex Pentium IV Xeon processor, [17] found that, due to a lower correlation, the number of samples required to perform the attack increased to 2^{19}. We analyzed this correlation for the Core i7 processor considered in this paper. Due to the complexity of this processor, there is very low correlation between the timing and the number of cache-misses. This means that most of the collected samples are actually not useful for the attack and a significantly higher number of samples is required to recover the key (2^{27}). Nevertheless, the attack is still possible.

Based on this consideration, if a direct measure of the number of cache-misses is available to the attacker (through the DCM counter), a significantly lower number of samples should be enough to recover the key as confirmed by the experimental results provided in Section V.

The possibility to access the PMCs by the malicious task, that includes timing and cache access information, opens a path to properly implement this attack in the node architecture presented in Fig. 2.

IV. Attack Mitigation

The success of the attack introduced in Section III-C depends on the PMC samples collected by the malicious task. To achieve its goal, the attacker has to statistically analyze the distribution of the different samples.

The main idea proposed in this paper to counteract this attacker is to modify the PMC service implemented at the OS level of Fig. 2. Each PMC reading is poisoned in order to obfuscate the statistical properties of the PMC:

$$PMC' = cf(PMC) \tag{7}$$

where PMC' is the corrupted PMC reading, PMC is the correct PMC reading, and $cf(PMC)$ is the *corruption function* that is a function of the value of the counter.

However, this corruption may jeopardize the capability of the safety task described in Section III-B to detect timing violations. This may potentially create both false negatives (i.e., undetected time violations) or false positives (i.e., safe conditions detected as violations). Therefore the corruption level must be carefully considered both from the security and from the safety standpoint.

It is worth recalling that, in the proposed architecture, the PMC service is considered secure (see section III-C) and represents the only user access point to the PMCs.

The collision condition introduced in equation (6) tells us that samples with PMC values (i.e., CCC or DCM) significantly lower than the average value computed over all collected samples are the ones actually important to recover the key. To obfuscate the properties of these samples, one option is to alter their values in such a way to violate the collision condition, leaving the remaining values unaffected by the poisoning.

To identify the values to alter, we exploit the CDF obtained profiling the PMCs from the encryption service. Fig. 4 reports the CDF computed profiling the CCC counter over 100,000 samples.

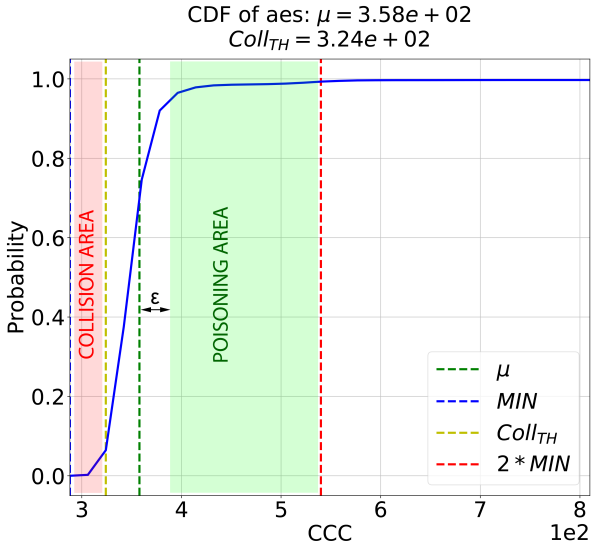


Fig. 4: CDF of the CCC counter for the encryption service. Vertical lines represent the position of the selected thresholds. $Coll_{TH}$ is set here in order to identify the lowest 5% samples.

The proposed solution is to set a threshold ($Coll_{TH}$) in such a way to identify the lowest x% samples of the CDF. Based on $Coll_{TH}$, it is possible to identify a *collision area* delimited by the lowest collected sample (MIN) and $Coll_{TH}$. Every sample falling in the collision area is moved to a random position of a different area called *poisoning area* (Fig. 4) by defining the corruption function of eq. (7) according to the following equation:

$$cf(PMC) = \begin{cases} PMC & (PMC < MIN) \vee (PMC > Coll_{TH}) \\ U(\mu + \varepsilon, ub) & MIN \leq PMC \leq Coll_{TH} \end{cases} \tag{8}$$

where PMC is the value of the counter, MIN is the lowest sample collected when profiling the encryption service and μ is the average value of the PMC over all samples collected during the profiling of the encryption service.

The upper bound of the poisoning area (ub) must not exceed two times the value of the minimum collected sample. This is motivated by the fact that, as described in [17], one of the techniques exploited by the attacker to optimize the attack is to discard samples higher than two times the minimum, considering them as outliers.

The lowest bound of the poisoning is instead defined by the ε parameter in order to create a guard band between the average (μ) and the beginning of the poisoning area. This is required since the effect of the corruption is to increase the average of the collected samples. The guard band can be empirically computed and must be large enough to guarantee that, even after the application of the poisoning, the corrupted samples are higher than the new average.

The decision of how many samples to corrupt (i.e., the selection of $Coll_{TH}$) and the size of the guard band (i.e., the selection of ε and ub) is a trade-off between security and amount of corruption. Higher values of $Coll_{TH}$ and ε increase the poisoning level and therefore the complexity of the attack (see Section V). However, increased poisoning levels may have a negative effect on those safety techniques that rely on the PMC to detect anomalies. Therefore, the selection of these parameters must be carefully analyzed considering the interplay between safety and security as will be discussed in Section V where the different values of poisoning will be selected in relation to the safety technique presented in Section III-B.

It is important to highlight that, a single corruption level is used for all tasks in the system. This translates into a very simple implementation at the OS level. Moreover, this simplicity opens up a path for possible hardware implementations of this technique that would relax some of the limitations currently imposed in the attack model. Nevertheless, this hardware implementation is out of the scope of this paper and will be considered in future extension of this research project.

As discussed in Section III-C, the proposed technique does not guarantee by proof the prevention of the considered attack. Differently, it increases the complexity of the attack that, in this case, translates in the requirement of collecting an increased number of samples in order to discover the secret key. This is an important result. In general a careful design of a CPS under the security domain would supply a limited operational lifetime of the encryption key, as well as key replacement policies. In this paper we do not consider how the different keys are generated or derived from previous keys , nor how they are distributed among the nodes of the CPS. Several techniques do exist in this domain [39], [40]. Instead we would like to discuss the impact of the proposed technique on the lifetime of the key. Experimental data (see Section V-B) show that the proposed approach significantly increases the number of samples required to perform the attack. Without considering the additional time required to analyze the samples (this can be carried out off-line with the support of high-performace computing facilities) the time required to collect additional samples has a positive impact on the possible lifetime of a key, thus reducing the overhead associated to the key generation and distribution.

V. Experimental Results

A. Experimental setup

To show the proposed approach at work, we evaluated the attack mitigation technique on a sample computing node.

To account for the fact that CPSs cannot constantly update their hardware architecture, we selected for our experiments a board equipped with a powerful but relatively old Intel Core i7-720QM CPU (released in Sept. 2009). This processor is based on the Intel first generation Nehalem microarchitecture. It embeds 4 cores, each equipped with a 64KB L1 cache (32 KB L1 data and 32 KB L1 instruction) and a 256KB L2 cache. Finally, the processor implements a 6MB shared L3 cache.

The poisoning technique was evaluated considering six different values of $Coll_{TH}$, each defined in order to identify the lowest x% samples (x $\in \{5\%, 7.5\%, 10\%, 12.5\%, 15\%\}$) obtained when profiling the encryption service (see Section IV). For this purpose, the encryption service was profiled collecting 100,000 samples. To define the poisoning area taking into account the considered safety technique, we selected $\varepsilon = W_{TH}^{AES} - \mu^{AES}$ (see equation(8) and Table I). This value is by construction large enough to guarantee a guard band as requested by our methodology. Moreover, to explore the impact of the size of the poisoning area on the effectiveness of the proposed technique, we evaluated different values for the upper bound of this area: $ub \in \{\frac{2 \cdot MIN}{16}, \frac{2 \cdot MIN}{8}, \frac{2 \cdot MIN}{4}, \frac{2 \cdot MIN}{2}, 2 \cdot MIN\}$,

To simulate the execution of different tasks we selected a set of 13 benchmarks from the MiBench benchmark suite [19] executed on a Linux operating system (kernel 3.11). These benchmarks represent typical algorithms implemented in several embedded systems sectors:

- *Automotive/Industrial*: susan-edges (edges), susan-corners (corners), susan-smoothing (smooth), quick sort (qsort), basic math tests (bscmath);
- *Consumer*: jpeg encoder (cjpeg), jpeg decoder (djpeg);
- *Office*: string search (strsrc);
- *Network and Security*: Dijkstra's algorithm (dijkstra), Secure Hash Algorithm (sha);
- *Telecommunications*: Fast Fourier Transform (fft), ADPCM encoder (rcaudio), ADPCM decoder (rdaudio).

Each task was profiled as described in Section III-B by collecting the value of the CCC and DCM counters over 100,000 executions. Input data for each task were taken from the "small" data set provided with the MiBench suite [19].

The collected profiles were used to compute W_{TH}, C_{TH} and the average PMC value (μ) of each task for the two considered counters as reported in Table I. The confidence levels to compute the two thresholds were set to $C_W = 4\%$ and $C_C = 0.6\%$ and according to equation (3) $\alpha = 3$.

For the CCC counter, Table I shows a clear separation of several orders of magnitude between W_{TH}, C_{TH} and μ of the encryption service (AES) and the ones of all other

TABLE I: Results of the profiling of the selected benchmarks. This also includes the profiling of the encryption service (AES).

	CCC counter			DCM counter		
Benchmark	W_{TH}	C_{TH}	μ	W_{TH}	C_{TH}	μ
AES	3.58e2	3.96e2	5.59e2	7.37e1	7.70e1	7.90e1
bscmath	5.35e6	5.50e6	6.21e6	1.69e1	2.30e1	2.60e1
cjpeg	1.20e7	2.64e7	2.95e7	3.31e4	3.33e4	3.36e4
corners	6.84e5	1.38e6	1.39e6	9.27e2	1.04e3	1.05e3
dijkstra	1.44e7	1.46e7	1.67e7	5.45e4	5.51e4	5.59e4
djpeg	3.38e6	8.01e6	8.11e6	6.73e3	6.96e3	7.23e3
edges	1.25e6	2.36e6	2.37e6	1.37e3	1.59e3	1.61e3
fft	3.54e5	4.04e5	4.32e5	3.78e3	4.10e3	4.31e3
qsort	1.70e7	3.27e7	3.28e7	4.18e5	4.19e5	4.21e5
rcaudio	2.15e9	4.12e9	4.27e9	1.84e9	3.54e9	3.66e9
rdaudio	2.15e9	4.12e9	4.27e9	1.80e9	3.46e9	3.58e9
sha	2.94e6	5.64e6	5.64e6	2.45e2	3.19e2	3.58e2
smooth	7.38e6	1.47e7	1.47e7	2.74e2	3.56e2	3.89e2
strsrc	1.26e5	3.24e5	3.24e5	1.49e2	1.75e2	2.05e2
synth01	2.69e2	3.96e2	5.33e2	4.84e1	7.70e1	7.80e1
synth02	2.88e2	3.96e2	5.36e2	6.38e1	7.70e1	7.80e1
synth03	3.35e2	3.96e2	5.34e2	5.58e1	7.70e1	7.80e1

benchmarks. Therefore, according to equation (8) and as will be analyzed in Section V-C the poisoning technique should have minimum impact on the monitoring capability of the safety task. Even if the gap is reduced for some benchmarks, the same separation can be observed when considering the CCC counter.

In order to better analyze how the safety task could be impacted by the presented poisoning technique for tasks with PMC profiles similar to the ones of the encryption service, a set of three synthetic profiles (synth01, synth02, and synth03 in Table I) was generated. These profiles contain random PMC values with a distribution that resembles the one of the encryption service.

To perform the attack, we adapted the C code presented in [17] available on the repository [41]. For each PMC, we collected 3 sets of samples on the node under attack. Each set contains $2^{30} = 1,073,741,824$ samples, with the related ciphertext of 16 byte. The samples were transferred to a remote workstation equipped with an Intel XEON E5-2680 2.70GHZ to be processed and attacked. For each set, the attack consisted in corrupting the samples and performing the attack. For each set we performed 3 repetitions of corruption and attack.

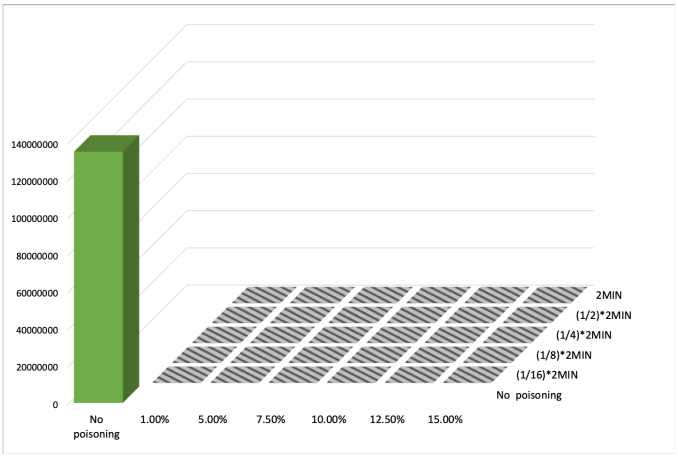
To analyze the behavior of the safety task, we profiled both the CCC and the DCM counters over 100,000 executions of each benchmark from Table I. After the profiling was completed we generated 1,000 traces for each benchmark (each of them composed of 100,000 samples) applying the same corruption levels used to perform the attack. All samples of each trace were classified according to the technique discussed in Section III-B.

To show the improvements of the attack mitigation technique proposed in this paper, we compared it with results obtained performing a similar analysis but applying our previous technique proposed in [18].

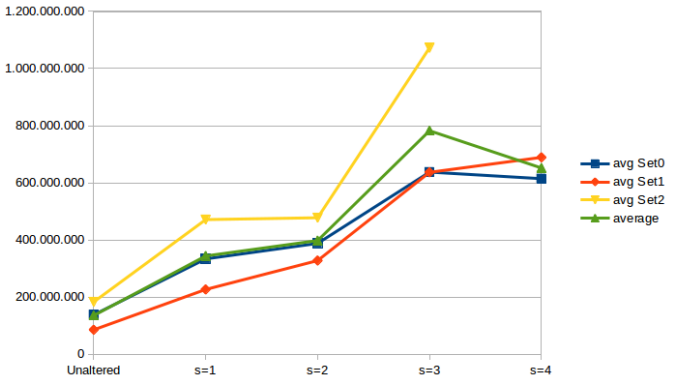
B. Attack mitigation results

The attack complexity can be measured for an attacker as the number of PMC samples needed to retrieve the encryption key.

Fig. 5 shows the number of samples required to perform the attack for the CCC counter for the two considered techniques.



(a) Results obtained using the poisoning technique presented in Section IV. Gray spots indicated the failure of the attack.



(b) Results obtained by applying the poisoning technique presented in [18] using different scaling factors .

Fig. 5: Number of samples to perform the attack for the CCC counter. Blue, orange and yellow data report the average values of the 3 repetitions of the experiment for each of the 3 considered sets of samples, while green values report the global average.

The number of samples required to retrieve the encryption key without any PMC poisoning is on average $\sim 135,000,000$ samples. This defines the baseline of the attack.

Fig. 5a shows that, regardless of the size of the collision (x axis) and poisoning area (y axis), the attack always fails after analyzing 2^{30} samples when applying the proposed attack mitigation technique. This is a significant improvement with respect to results obtained applying the technique proposed in [18] (Fig. 5b). In this last case, the attacker is in general able to recover the key with less samples while introducing a higher level of corruption of the counter.

While it is impossible to prove that for an increased number of samples the attack would not succeed, it is important to highlight that the amount of data the attacker has to collect passes from $\sim$2.5GB for the baseline up to $\sim$20GB when 2^{30} samples are collected. In a real scenario these data must be collected in a stealthy way, without draining all the resources

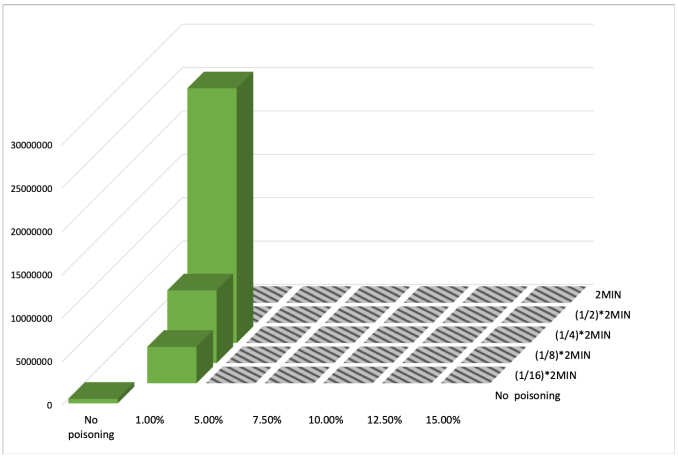
of the system. Therefore, this could require a significant amount of time that, coupled with the use of key replacement keys (see Section IV), may further help counteracting the attacker.

Fig 6 reports a similar analysis for the DCM counter. The attack against the unaltered set of samples requires on average $524,288$ samples to discover the key. This value is significantly lower than the one required to perform the attack using the CCC counter.

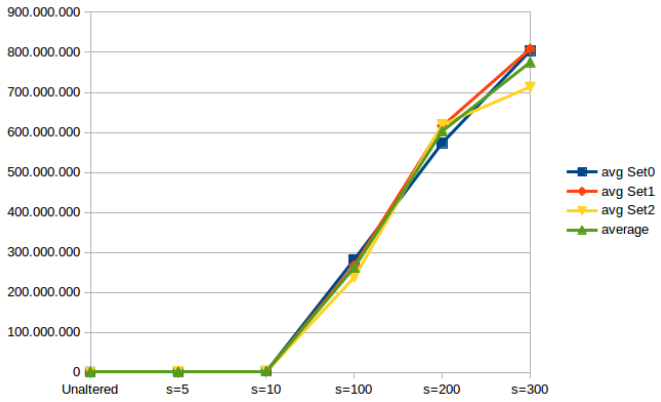
When looking at samples collected for the two considered PMCs (i.e., CCC and DCM), we noticed, as expected, that DCM samples are more stables (i.e., they have a lower variance). Differently, as discussed in Section III-C, the CCC counter is affected by several complex HW/SW interactions that are in general hard to predict. These interactions introduce variations in the timing behavior of the application across different executions, creating a higher variance in the observed values of the CCC timer and reducing its correlation with the number of cache misses (see Section III-C). This motivates the reduced number of samples required to perform the attack using DCM. The higher stability of this PMC makes it difficult to protect using the technique proposed in [18]. Even with very high scaling factors (Fig. 6b), the poisoning technique proposed in [18] is unable to introduce sufficient improvements from the security point of view. Moreover, this high poisoning would introduce an unacceptable corruption from the safety point of view. More in details, we observed that with this technique the altered samples maintain a similar allocation with respect to the average value, i.e., the majority of the original samples lower than the average remain lower than the average even after the alteration. They therefore retain the information exploited by the attacker to recover the key.

Differently, the technique proposed in this paper (Fig. 6a) is able to efficiently protect also the DCM counter. As for the CCC counter, the attack always fails with the only exception of the extreme cases in which both the collision area and the poisoning area are strongly reduced. Nevertheless, even in this cases we observe significant improvements in the number of samples required to perform the attack.

To summarize, the proposed attack mitigation technique, provides optimal results from the security perspective, considering both DCM and CCC counters.



(a) Results obtained using the poisoning technique presented in Section IV. Gray spots indicated the failure of the attack.



(b) Results obtained by applying the poisoning technique presented in [18] using different scaling factors

Fig. 6: Number of samples to perform the attack for the DCM counter. Blue, orange and yellow data report the average values of the 3 repetitions of the experiment for each of the 3 considered sets of samples, while green values reports the global average.

C. Safety results

Figs. 7 and 8 show, for each task, the number of executions that fall in the different safety states introduced in Section III-B for the CCC and DCM counter, respectively. Subfigures are associated to the different safety states: safe direct (Figs. 7a and 8a), critical direct (Figs. 7b and 8b), safe warning (Figs. 7c and 8c), and critical warning (Figs. 7d and 8d). Results are provided for different corruption levels of the CCC and DCM counters, averaging the results over 1,000 repetitions of the analysis.

Looking at the figures it is clear that the MiBench tasks are not impacted by the alteration of both the CCC and DCM counters for all considered percentage sizes of the collision area (x axis). For all cases we used the largest poisoning area (i.e., $ub = 2 \cdot MIN$). This is favored by the fact that W_{TH} and CT_{TH} of all these tasks are significantly different when compared to the considered poisoning windows and is a significant improvement with respect to the results presented in [18].

Some changes can be observed only when looking at the synthetic benchmarks.

Looking at the CCC counter, in the worst case (synth02) the number of safe direct classifications drops from 96,018 to 77,035 generating 18,985 potential fault positives. This is partially compensated by an increment of 3,884 safe warning cases. Nevertheless, the remaining 15,101 cases out of the 100,000 total executions represent false positives that must be handled. This impacts the performance of the system due to an increased number of recovery actions required during in field operations. A similar analysis can be carried out for the DCM counter. By construction, no false negatives can be introduced by the proposed technique, since the corruption is always introduced as an additive positive factor.

Even if this may look a negative result, it is important to recall that the profiles of the synthetic benchmarks have been specifically generated to resemble those of the encryption service. Therefore they represent an extreme worst case situations difficult to encounter in real applications. These results must be only considered as examples to better show the critical aspects that should be carefully taken into account when designing a CPS in which both safety and security must be guaranteed as described in this paper.

This paper studied the interplay of two challenging aspects of the design of a CPS: safety and security. It focused on the role that the PMCs have when implementing mechanisms able to enhance the safety of the system and, on the other hand, the risks they introduce when looking at the security of the system. Starting from the example of a PMC based safety mechanism, and from the implementation of a security attack, the paper proposed an attack mitigation strategy. Two different PMC were analyzed and an extensive experimental campaign shows the effectiveness of the proposed attack mitigation technique for both considered counters. This provides interesting suggestions on how a designer should decide which PMC can be securely exposed to the application software.

REFERENCES

[1] K. Carruthers, "Internet of things and beyond: Cyber-physical systems," *IEEE Internet of Things Newsletter*, vol. 10, 2014.

[2] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: The next computing revolution," in *Design Automation Conference*, June 2010, pp. 731–736.

[3] M. Bhrugubanda, "A Review on Applications of Cyber Physical Systems," *International Journal of Innovative Science, Engineering & Technology*, vol. 2, no. 6, pp. 728–730, June 2105.

[4] H. Chen, "Applications of cyber-physical system: a literature review," *Journal of Industrial Integration and Management*, vol. 2, no. 03, p. 1750012, 2017.

[5] L. Wang and X. V. Wang, *Cloud-Based Cyber-Physical Systems in Manufacturing*. Springer, 2018.

[6] J. Ding, Y. Atif, S. F. Andler, B. Lindström, and M. Jeusfeld, "CPS-based Threat Modeling for Critical Infrastructure Protection," *SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 2, pp. 129–132, Oct. 2017.

[7] M. Wolf and D. Serpanos, "Safety and Security in Cyber-Physical Systems and Internet-of-Things Systems," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 9–20, Jan 2018.

[8] G. Sabaliauskaite and A. P. Mathur, "Aligning cyber-physical system safety and security," in *Complex Systems Design & Management Asia*. Springer, 2015, pp. 41–53.

[9] T. Novak and A. Treytl, "Functional safety and system security in automation systems - a life cycle model," in *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, Sept 2008, pp. 311–318.

[10] L. Piètre-Cambacédès and C. Chaudet, "The SEMA referential framework: Avoiding ambiguities in the terms "security" and "safety"," *International Journal of Critical Infrastructure Protection*, vol. 3, no. 2, pp. 55–66, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1874548210000247

[11] L. Piètre-Cambacédès and M. Bouissou, "Cross-fertilization between safety and security engineering," *Reliability Engineering & System Safety*, vol. 110, pp. 110–126, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832012001913

[12] ——, "Modeling safety and security interdependencies with BDMP (Boolean logic Driven Markov Processes)," in *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*. IEEE, 2010, pp. 2852–2861.

[13] S. Esposito, M. Violante, M. Sozzi, M. Terrone, and M. Traversone, "A Novel Method for Online Detection of Faults Affecting Execution-Time in Multicore-Based Systems," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 4, pp. 94:1–94:19, May 2017.

[14] J. Nowotsch, M. Paulitsch, D. Bühler, H. Theiling, S. Wegener, and M. Schmidt, "Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement," in *2014 26th Euromicro Conference on Real-Time Systems*, July 2014, pp. 109–118.

[15] E. Mezzetti, L. Kosmidis, J. Abella, and F. J. Cazorla, "High-Integrity Performance Monitoring Units in Automotive Chips for Reliable Timing V and V," *IEEE Micro*, vol. 38, no. 1, pp. 56–65, January 2018.

[16] M. F. B. Abbas, S. P. Kadiyala, A. Prakash, T. Srikanthan, and Y. L. Aung, "Hardware performance counters based runtime anomaly detection using svm," in *2017 TRON Symposium (TRONSHOW)*, Dec 2017, pp. 1–9.

[17] J. Bonneau and I. Mironov, "Cache-collision timing attacks against AES," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2006, pp. 201–215.

[18] A. Carelli, A. Vallero, and S. Di Carlo, "Shielding Performance Monitor Counters: a double edge weapon for safety and security," in *24th IEEE International Symposium on On-Line Testing and Robust System Design*, 2018.

[19] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, Dec 2001, pp. 3–14.

[20] A. Vallero, A. Savino, G. Politano, S. D. Carlo, A. Chatzidimitriou, S. Tselonis, M. Kaliorakis, D. Gizopoulos, M. Riera, R. Canal, A. Gonzalez, M. Kooli, A. Bosio, and G. D. Natale, "Cross-layer system reliability assessment framework for hardware faults," in *2016 IEEE International Test Conference (ITC)*, Nov 2016, pp. 1–10.

[21] A. Chatzidimitriou, M. Kaliorakis, D. Gizopoulos, M. Iacaruso, M. Pipponzi, R. Mariani, and S. D. Carlo, "RT Level vs. Microarchitecture-Level Reliability Assessment: Case Study on ARM(R) Cortex(R)-A9 CPU," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, June 2017, pp. 117–120.

[22] A. Savino, A. Vallero, and S. D. Carlo, "ReDO: Cross-Layer Multi-Objective Design-Exploration Framework for Efficient Soft Error Resilient Systems," *IEEE Transactions on Computers*, pp. 1–1, 2018.

[23] M. Kaliorakis, D. Gizopoulos, R. Canal, and A. Gonzalez, "MeRLiN: Exploiting Dynamic Instruction Behavior for Fast and Accurate Microarchitecture Level Reliability Assessment," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: ACM, 2017, pp. 241–254. [Online]. Available: http://doi.acm.org/10.1145/3079856.3080225

[24] E. Novillo and P. Lu, "On-line debugging and performance monitoring with barriers," in *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001*, April 2001, pp. 8 pp.–.

[25] S. Moore, D. Terpstra, K. London, P. Mucci, P. Teller, L. Salayandia, A. Bayona, and M. Nieto, "Papi deployment, evaluation, and extensions," in *2003 User Group Conference. Proceedings*, June 2003, pp. 349–353.

[26] M. Domínguez-Morales, P. Iñigo, J. L. Font, D. Cascado, G. Jimenez, F. Díaz, J. L. Sevillano, and A. Linares-Barranco, "Frames-to-aer efficiency study based on cpus performance counters," in *Proceedings of the 2010 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS '10)*, July 2010, pp. 141–148.

[27] A. Kandalintsev, R. L. Cigno, D. Kliazovich, and P. Bouvry, "Profiling cloud applications with hardware performance counters," in *The International Conference on Information Networking 2014 (ICOIN2014)*, Feb 2014, pp. 52–57.

[28] G. Theodorou, N. Kranitis, A. Paschalis, and D. Gizopoulos, "Software-based self-test for small caches in microprocessors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1991–2004, Dec 2014.

[29] A. Paschalis and D. Gizopoulos, "Effective software-based self-test strategies for on-line periodic testing of embedded processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 1, pp. 88–99, Jan 2005.

[30] X. Wang, C. Konstantinou, M. Maniatakos, and R. Karri, "ConFirm: Detecting firmware modifications in embedded systems using Hardware Performance Counters," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2015, pp. 544–551.

[31] Y. Xia, Y. Liu, H. Chen, and B. Zang, "CFIMon: Detecting violation of control flow integrity using performance counters," in *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*. IEEE, 2012, pp. 1–12.

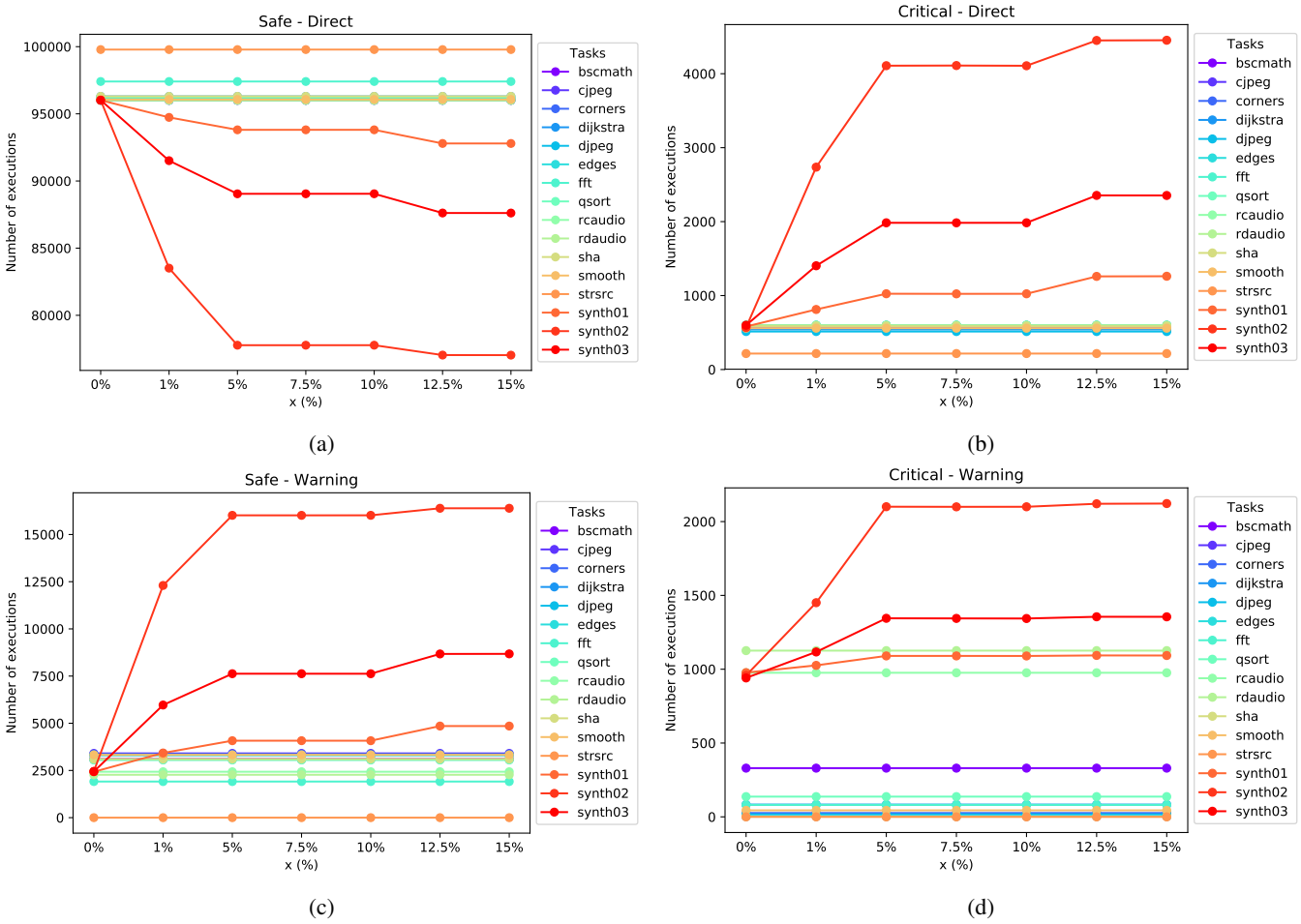


Fig. 7: Impact of the CCC corruption on the safety state classification of each benchmark

[32] C. Yilmaz, "Using hardware performance counters for fault localization," in *2010 Second International Conference on Advances in System Testing and Validation Lifecycle*, Aug 2010, pp. 87–92.

[33] C. Yilmaz, A. Paradkar, and C. Williams, "Time will tell," in *2008 ACM/IEEE 30th International Conference on Software Engineering*, May 2008, pp. 81–90.

[34] D. J. Bernstein, "Cache-timing attacks on AES," [Online] https://cr.yp.to/antiforgery/cachetiming-20050414.pdf, 2005.

[35] L. Uhsadel, A. Georges, and I. Verbauwhede, "Exploiting Hardware Performance Counters," in *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, Aug 2008, pp. 59–67.

[36] S. Bhattacharya and D. Mukhopadhyay, "Who watches the watchmen?: Utilizing Performance Monitors for Compromising keys of RSA on Intel Platforms," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 248–266.

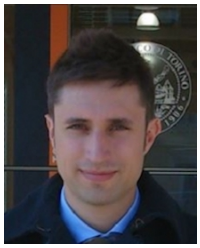
[37] M. Alam, S. Bhattacharya, D. Mukhopadhyay, and S. Bhattacharya, "Performance Counters to Rescue: A Machine Learning based safeguard against Micro-architectural Side-Channel-Attacks."

[38] H. Hassan, L. T. Yang, J. Xue, and E. Villar, "Special issue on: "Heterogeneous architectures for Cyber-physical systems (HACPS)"," *Microprocessors and Microsystems*, vol. 52, pp. 333–334, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933117301874

[39] E. Barker and A. Roginsky, "Recommendation for cryptographic key generation," *NIST Special Publication*, vol. 800, p. 133, 2012.

[40] J. Zhang and V. Varadharajan, "Wireless sensor network key management survey and taxonomy," *Journal of network and computer applications*, vol. 33, no. 2, pp. 63–75, 2010.

[41] J. Bonneau, "aes_cache," https://github.com/jcb82/aes_cache, 2014.



Alessandro Vallero (S'15) received a Ph.D. in computer engineering from Politecnico di Torino in Italy and a M.Sc. degree in electronic engineering from the University of Illinois at Chicago, US, and Politecnico di Torino, Italy. Currently he is a postdoc at the Department of Control and Computer Engineering of Politecnico di Torino in Italy. His research interests focus on system level reliability and reliable reconfigurable systems.

Alberto Carelli (S'18) received a M.Sc. in computer engineering from Politecnico di Torino in Italy. Currently he is a Ph.D. student at the Department of Control and Computer Engineering of Politecnico di Torino in Italy. His research interests focus on hardware security and reliability for cyber-physical systems and critical infrastructures.

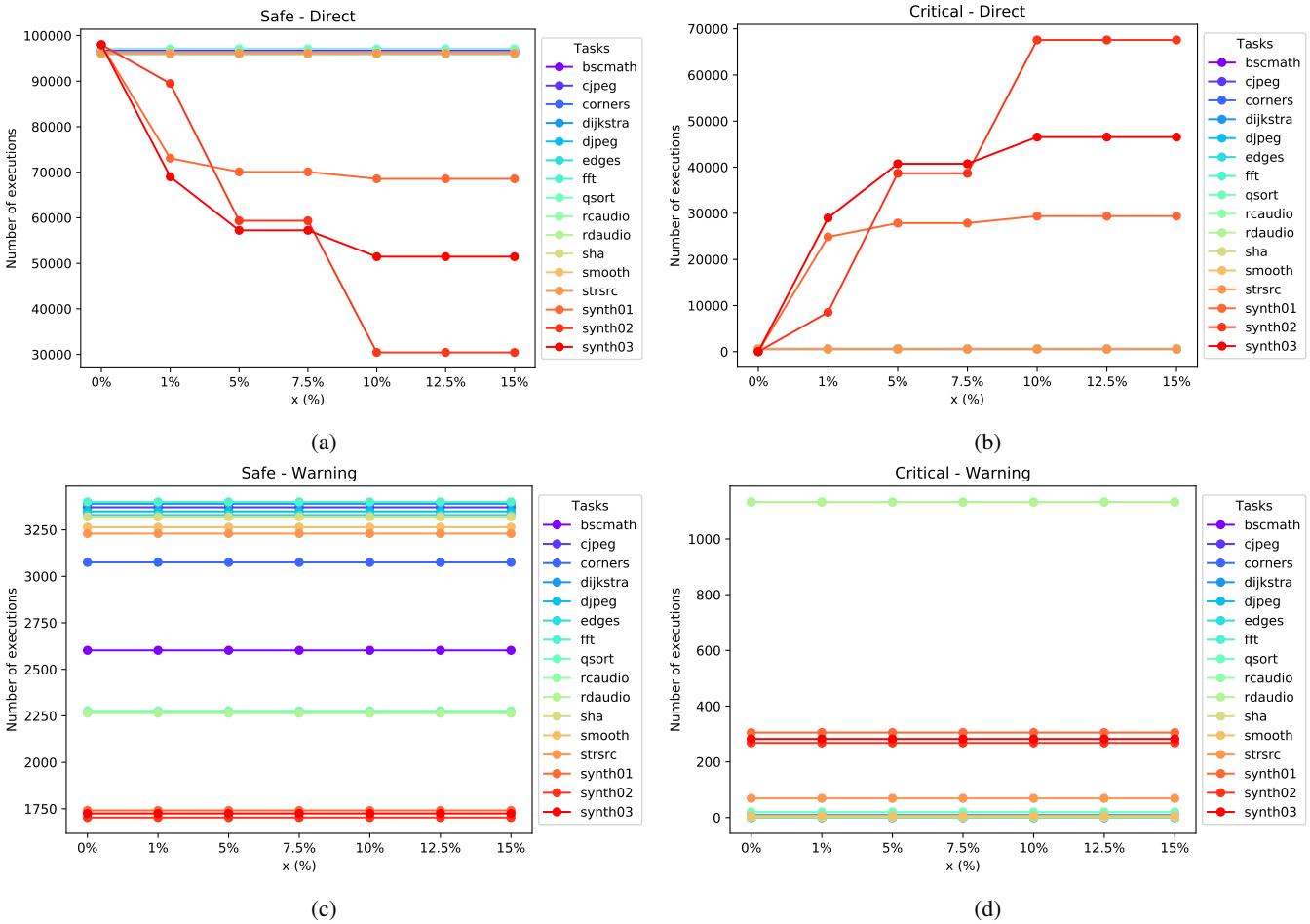


Fig. 8: Impact of the DCM corruption on the safety state classification of each benchmark

Stefano Di Carlo (SM'00-M'03-SM'11) received a M.Sc. degree in computer engineering and a Ph.D. degree in information technologies from Politecnico di Torino, Italy, where he is a tenured Associate professor. His research interests include DFT, BIST, and dependability. He has coordinated the EU-FP7 CLERECO on Cross- Layer Early Reliability Estimation for the Computing cOntinuum. Di Carlo has published more than 170 papers in peer reviewed IEEE and ACM journals and conferences. He regularly serves on the Organizing and Program Committees of major IEEE and ACM conferences. He is a golden core member of the IEEE Computer Society and a senior member of the IEEE.