# Leibniz Transactions on Embedded Systems

**Volume 7 | Issue 1 | August 2021**
**Special Issue on Embedded System Security**

**ISSN 2199-2002**

*Published online and open access by*
the European Design and Automation Association (EDAA) / EMbedded Systems Special Interest Group (EMSIG) and Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany.
Online available at
`https://www.dagstuhl.de/dagpub/2199-2002`.

*Publication date*
August 2021

*Bibliographic information published by the Deutsche Nationalbibliothek*
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://dnb.d-nb.de.

*Digital Object Identifier*
10.4230/LITES-v007-i001

*Aims and Scope*
LITES aims at the publication of high-quality scholarly articles, ensuring efficient submission, reviewing, and publishing procedures. All articles are published open access, i.e., accessible online without any costs. The rights are retained by the author(s).

LITES publishes original articles on all aspects of embedded computer systems, in particular: the design, the implementation, the verification, and the testing of embedded hardware and software systems; the theoretical foundations; single-core, multi-processor, and networked architectures and their energy consumption and predictability properties; reliability and fault tolerance; security properties; and on applications in the avionics, the automotive, the telecommunication, the medical, and the production domains.

*Contact*
Schloss Dagstuhl – Leibniz-Zentrum für Informatik
LITES, Editorial Office
Oktavie-Allee, 66687 Wadern, Germany
lites@dagstuhl.de
`http://www.dagstuhl.de/lites`

# Contents

# Special Issue on Embedded System Security: Foreword

**Alan Burns**
University of York, UK

**Steve Goddard**
University of Iowa, Iowa City, US

**2012 ACM Subject Classification** Security and privacy → Embedded systems security
**Keywords and Phrases** Foreword, Embedded System Security
**Digital Object Identifier** 10.4230/LITES.7.1.0
**Published** 2021-08-12
**Editor** Alan Burns and Steve Goddard
**Special Issue** Special Issue on Embedded System Security

Embedded systems are now an integral part of our lives. We have smart phones, smart meters, smart appliances, smart cars, smart grids, and smart houses–most relying on embedded systems with outdated security mechanisms, if they have any at all. A renewed emphasis on embedded systems security research is critical to our economies and our daily lives. This special issue on Embedded System Security attempts to contribute to this work by drawing attention to a number of key topics including Intrusion Detection and Tolerance, Confidence and Threat Modelling, Enhancing Dependability in Embedded Systems, and reducing Vulnerabilities in System Architectures for Embedded Systems.

Two papers are included in this initial instalment of the Special Issue. In the first paper "Randomization as Mitigation of Directed Timing Inference Based Attacks on Time-Triggered Real-Time Systems with Task Replication" by Kristin Krüger, Nils Vreman, Richard Pates, Martina Maggio, Marcus Völp and Gerhard Fohler, the vulnerabilities of time-triggered systems are investigated. They note that the assumption that faults are independent, which is often made for accidental faults, is not valid for malicious attacks. They go on to introduce two runtime mitigation strategies to withstand directed timing inference. Both involve the introduction of a level of randomization within the usual deterministic behaviour of time-triggered systems.

In the second paper "We know what you're doing! Application detection using thermal data", Philipp Miedl, Rehan Ahmed and Lothar Thiele consider how sensitive runtime information can be extracted from a system by just using temperature sensor readings from a mobile device. They employ a Convolutional-Neural-Network to identify the sequence of executed applications over time. They test their hypothesis via collected data from two state-of-the-art smartphones and real user usage patterns. The accuracy of their finding demonstrated that this is a clear vulnerability in mobile devices, including the potential to compromise sensitive user data.

Alan Burns and Steve Goddard
August, 2021

# Randomization as Mitigation of Directed Timing Inference Based Attacks on Time-Triggered Real-Time Systems with Task Replication

**Kristin Krüger**
Department of Electrical and Computer Engineering, Technische Universität Kaiserslautern, Kaiserlautern, Germany

**Nils Vreman**
Department of Automatic Control, Lund University, Lund, Sweden

**Richard Pates**
Department of Automatic Control, Lund University, Lund, Sweden

**Martina Maggio**
Department of Automatic Control, Lund University, Lund, Sweden
Department of Computer Science, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

**Marcus Völp**
SnT – Université du Luxembourg, Esch-sur-Alzette, Luxembourg

**Gerhard Fohler**
Department of Electrical and Computer Engineering, Technische Universität Kaiserslautern, Kaiserslautern, Germany

## Abstract

Time-triggered real-time systems achieve deterministic behavior using schedules that are constructed offline, based on scheduling constraints. Their deterministic behavior makes time-triggered systems suitable for usage in safety-critical environments, like avionics. However, this determinism also allows attackers to fine-tune attacks that can be carried out after studying the behavior of the system through side channels, targeting safety-critical victim tasks. Replication – i.e., the execution of task variants across different cores – is inherently able to tolerate both accidental and malicious faults (i.e. attacks) as long as these faults are independent of one another. Yet, targeted attacks on the timing behavior of tasks which utilize information gained about the system behavior violate the fault independence assumption fault tolerance is based on. This violation may give attackers the opportunity to compromise all replicas simultaneously, in particular if they can mount the attack from already compromised components. In this paper, we analyze vulnerabilities of time-triggered systems, focusing on safety-certified multicore real-time systems. We introduce two runtime mitigation strategies to withstand directed timing inference based attacks: (i) schedule randomization at slot level, and (ii) randomization within a set of offline constructed schedules. We evaluate these mitigation strategies with synthetic experiments and a real case study to show their effectiveness and practicality.

**2012 ACM Subject Classification** Computer systems organization → Real-time systems; Security and privacy → Operating systems security; Software and its engineering → Scheduling; Computer systems organization → Redundancy

**Keywords and Phrases** real-time systems, time-triggered systems, security

**Digital Object Identifier** 10.4230/LITES.7.1.1

**Supplementary Material** *Software*: `https://gitlab.control.lth.se/NilsVreman/rand-sched`

**Funding** This work is partially supported by: (i) the EC through H2020 grant 871259, ADMORPH, (ii) funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 432878494, and (iii) FNR by FNR-Core Jr. Project HyLIT – CS18/IS/12686210. Nils Vreman, Richard Pates, and Martina Maggio are members of the ELLIIT stategic research area at Lund University.

**Acknowledgements** We want to thank the reviewers for their helpful comments which greatly improved this paper. We are also gratefully indebted to Florian Heilman, Gautam Gala, Luiz Maia and Alexandre Venito from TU Kaiserslautern for their comments on an earlier version of this paper. Their insights and expertise assisted research, however, any errors found are our own.

**Received** 2020-04-14 **Accepted** 2020-12-14 **Published** 2021-08-12

**Editor** Alan Burns and Steve Goddard

**Special Issue** Special Issue on Embedded System Security

## 1 Introduction

In the past, real-time systems security had been given little thought, primarily because systems were closed, ran on specialized hardware, and had limited access from the outside. This has changed in the past decade due to the growing need for connectivity, computing power, the shift from single to multi- and many-core architectures, and software-component reuse (e.g., from federated avionics architectures to Integrated Modular Avionics (IMA) [56]). These trends result in a general increase of complexity, in particular at real-time application level, with the entailed increase of the likelihood of vulnerabilities going undetected, even in well-tested software. Consequently, real-time system designers must now anticipate the risk of hackers trying to compromise components and being partially successful in this task. Security has become a primary concern during system design and deployment, not only to prevent unauthorized information disclosure, but also to prevent malicious exploitation of vulnerabilities [54]. This is especially true for systems in safety-critical environments, thus for the majority of real-time systems.

Real-time systems provide deterministic and dependable behavior, both in the value-domain and in the timeliness of responses. To achieve IEC 61508 or ISO 26262 compliance [19,21], industry employs time-triggered real-time systems which, in addition to guaranteeing that deadlines will be met, provide *determinism* in the sense that it is known which task executes at any given point in time. Unfortunately, the very essence of this determinism opens threat vectors that attackers can exploit to harm the system and the environment in which it operates. Aside from inheriting all classic security concerns related to providing correct, trustworthy results, real-time systems must also produce these results in time. Attackers may exploit this property by delaying the execution of individual tasks in targeted time-domain attacks. Moreover, the traditional means for tolerating accidental faults, for example active replication [10, 14, 20] or their counterparts for intentionally malicious faults [2, 28, 40], which require only a majority of components to work correctly, are not immune against time-domain attacks. Such attacks constitute a common mode fault against which replication does not protect, even if replicas are diversified (e.g., through n-version programming).

In general, attacks on the timing of tasks are more effective the more information adversaries obtain about the system and its schedule. Adversaries may fine-tune attacks to precise points in time and thus remain stealthy to evade detection and act when the attack is most effective. For such attacks, time-triggered systems appear the perfect target due to their deterministic schedules. On the other hand, this determinism seems to be the perfect protection as tasks can only execute during predefined time windows which do not overlap with other tasks execution. However, this perception hinges on the assumption of perfect isolation, which is brittle as long as tasks are analyzed only according to their specified behavior (e.g., when determining cache-related preemption delays) and as long as isolation remains imperfect. We therefore have to agree with Yoon et al. [58] in their conclusion that time-triggered scheduling is inherently vulnerable to timing inference based attacks.

In this article, we consolidate the findings of three independent works [25, 27, 53] of the authors into a comprehensive analysis of the vulnerabilities and countermeasures of time-triggered systems against timing interference based attacks. We investigate how adversaries can exploit vulnerabilities in an accomplice task to prepare and execute targeted attacks to singleton and replicated critical subsystems. Based on this analysis, we propose two basic strategies to mitigate attacks:

1. online randomization of the time-triggered schedule while preserving all timing constraints
2. random online selection of offline-prepared time-triggered schedules at hyperperiod boundaries.

These results, in particular the second, are formalized by constructing schedule sets of minimal size that achieve the highest possible upper-approximated entropy. Through this formalization, we were able to conclude the effectiveness of online randomization in preventing adversaries from mounting targeted attacks. Moreover, we were able to conclude that a set of 100 offline computed schedules suffices for our second mitigation strategy. We have evaluated our approach using synthetic task sets and the ROSACE real-world case study.

In addition to already published results, this article extends our mathematical foundation from implicit to constrained deadline task sets (although the bound for the latter is not tight). The extension is purely theoretical, but shows that constrained deadlines reduce the achievable upper-approximated entropy (because they limit the amount of randomization that can be introduced in the system). The article also presents the results of an upper-approximated entropy limitation analysis for the ROSACE case study. This last result demonstrates that entropies close to the theoretical optimum can be achieved with realistic task sets that do not necessarily have the ideal parameter distribution for this optimum.

Section 2 introduces our system model. Section 3 describes our attacker model. We present two mitigation strategies in Section 4. In Section 5, we discuss inherent limitations of schedule diversity and randomization. We discuss attack vectors and mitigations strategies in a multicore system with task replication in Section 6. In Section 7, we evaluate our strategies using two sets of experiments: synthetic task sets and a real-world case study. We discuss our findings and explore limitations that are due to the implementation in embedded systems. Section 8 shows related work. Finally, we conclude our work in Section 9.

## 2 System model

We analyze a time-triggered real-time system implemented on a multicore platform, following a partitioned schedule. For each core, a local schedule is constructed offline and then adjusted online as discussed in Section 4. We assume the schedules have been validated. Precautions such as authenticated boot are in place to ensure that the validated schedule is correctly deployed to the real-time system. A real-time operating system is present, which we assume to be correct. That is, we do not consider bare metal real-time implementations. Safety-critical tasks are replicated and replicas are distributed across cores to benefit from improved resilience should one core fail. The implementation of these replicas follows an implementation diversification approach to increase system dependability. We assume the system is configured to be able to tolerate up to $f$ faults of arbitrary kind simultaneously and focus in this work on what is required to tolerate time-domain attacks in addition. That is, our approach is equally applicable to systems prepared for tolerating accidental faults or maliciously induced (value-domain) faults and for tolerating crash as well as Byzantine faults. The replication degree, i.e. the number $n$ of replicas required to achieve the mentioned fault tolerance, depends on this fault model, on the achieved system synchrony and on the kind of agreement, e.g. single value versus interval mid point. Again, our approach is prepared for any combination of the above. We therefore assume what the above models have in common, namely that no more than $f$ replicas fail simultaneously and that replicas fail independently in the value-domain. We shall see that, without further precautions, a general fault independence requirement can no longer be maintained in the presence of time-domain attacks.

The scheduler on each core has access to a global time base typical for time-triggered systems. We consider global time adequately protected and divide it into slots of the same size, which are the granule for job preemption. Our focus lies on CPU-level scheduling, hence we do not consider time-triggered networks or communication channels. Moreover, we assume a purely time-triggered system without asynchronous event activation.

For each core $c$, we are given a task set $\mathcal{T}^c = \{\tau_1^c, \tau_2^c, \ldots, \tau_m^c\}$ of $m^c$ periodic tasks. Each task $\tau_i^c$ is defined as the tuple $\tau_i^c = \{e_i^c, t_i^c, d_i^c\}$, where $e_i^c$ is the worst-case execution time of the task[1] , $t_i^c$ is the task activation period, and $d_i^c$ is the task (relative) deadline. Throughout the rest of the paper, we assume that $d_i^c \leq t_i^c$, adopting the constrained deadline task model.

We define the time interval between task release and task deadline as this task's execution window. We denote with $U^c$ the task set utilization for core $c$, i.e., $U^c = \sum_{i=1}^{m^c} e_i^c/t_i^c$. We assume that the task set is schedulable; ergo, there exists a resource distribution such that each task meets its corresponding deadline. We denote with $\ell^c$ the hyperperiod of the task set for core $c$, i.e., the least common multiple $\mathrm{lcm}(\cdot)$ of the task periods, $\ell^c = \mathrm{lcm}(t_1^c, \ldots, t_m^c)$.

We now look at a specific core $c$.[2] A schedule $s^c$ for the task set in core $c$ is a sequence of $\ell^c$ elements, that contains numbers in the set $\{0, 1, \ldots, m^c\}$, a number $j \in \{1, \ldots, m^c\}$ denotes the execution of task $\tau_j^c$. Assigning a given task to an element means selecting which task is executed on core $c$ for the corresponding time slot. Choosing $j = 0$ represents the execution of the idle task, meaning keeping the processor in the idle state. We denote the idle task with $\tau_0^c = \{e_0^c, t_0^c, d_0^c\}$ and we determine its characteristics based on the characteristics of the task set. In particular, $t_0^c = d_0^c = \ell^c$, and $e_0^c = \ell^c\,(1 - U^c)$.

Formally,

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

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

## 3 Threat Model and Vulnerability Analysis

In this section, we first describe our threat model, highlighting in particular the assumptions we make on the attacker and how he or she is constrained by time-triggered systems. After that, we analyze the vulnerabilities present in time-triggered systems.

### 3.1 Threat Model

We assume attackers are able to successfully infiltrate the system through undetected vulnerabilities. Less stringent evaluation requirements make non real-time tasks as well as not safety-critical tasks primary targets. In particular, we assume that critical tasks are sufficiently shielded against direct attacks which requires attackers to find a pathway through less critical tasks. Firewalls and gateways in autonomous vehicles and planes support this assumption. Even though we assume intrusion detection [8, 32, 35, 60], hardening mechanisms and other defenses against the common attack vectors (e.g. DoS attacks) are in place, we acknowledge that these techniques are imperfect and compromises may go undetected.

Of particular concern to us are stealthy attackers [6, 7] that continue normal operation of the compromised tasks while gathering timing information about other, critical tasks. The knowledge about the timing of critical tasks allows to determine the point in time when a directed attack

[1] To guarantee the correct execution of the task set, we ensure that a time equal to the worst-case execution time is assigned to each task in each of the execution periods (i.e., to each job of each task). This means that the time budget is allocated to the task even when the job completes its execution early.

[2] In the following, when it is clear from the context that we talk about one specific core (and in particular in Section 5), we will drop the superscript $c$ and use $\mathcal{T}$, $\tau_i$, $e_i$, $t_i$, $d_i$, $U$, $\ell$, $s$ and $s_j$.

is most effective, e.g., immediately before a safety-critical victim task is run. Possible targets of such attacks in time-triggered systems are the low-level control loops. Destabilizing these components (e.g., by increasing the dead time or by introducing jitter in the control cycle) may provoke critical failure modes and thus result in a continuous denial of service [55], or worse, unsafe control decisions.

The timing information required for coordinating such a stealthy attack can be inferred via side channels constructed using shared resources like cache [3] or memory, or through covert timing channels, such as the scheduling-covert-channel described by Boucher et al. [5].

While there exist mitigation strategies for closing side channels (for example in the real-time context, the works of Völp et al. [52] or Mohan et al. [36] on fixed-priority schedulers), these methods are incomplete. Additionally, systematically closing all side channels typically entails significant performance overheads, e.g. when flushing caches prior to scheduling a lower classified task [17]. Meltdown [30] and Spectre [22] are recent examples demonstrating the difficulty of identifying and closing such channels in sufficiently complex architectures. Exploiting non-architectural channels (e.g., cache allocation) as communication medium, Meltdown and Spectre extract confidential information from speculative processor state, breaking security on most Intel and many high-end ARM and AMD processors. While real-time systems traditionally avoid such complex hardware, their future integration in real-time system-on-chip, e.g., for meeting the extended demand of autonomous driving functionalities, cannot be excluded.

We assume the real-time system features isolation mechanisms for enforcing the schedule of tasks and for limiting direct access to the memory of other tasks. Real-time operating systems (RTOS) that feature memory isolation support this assumption unless attackers are able to penetrate the operating system. For the purpose of this paper, we assume the deployed RTOS excludes the possibility of OS penetration.

One immediate consequence of this isolation assumption is that when the attacker has infiltrated the system, he or she is inherently constrained by properties of the system and its architecture for subsequent attacks on more critical tasks. In time-triggered systems, table-driven scheduling prevents influencing other tasks, e.g. by manipulating the execution time of a compromised task. That is, in contrast to event-triggered scheduling, each task is confined to its execution window and thus the actual task execution time has no influence on subsequent tasks. Time-triggered systems therefore provide temporal isolation of CPU time irrespective of the actual behavior of tasks and without having to revert to timing leak transformations as described for example by Völp et al. [52]. Additionally, messages are only accepted during a certain time window, i.e., if they are timely.

Operating system enforced schedules combined with the assumed impenetrability of the OS ensure that the attacker can neither directly influence the scheduler nor can he or she read the offline constructed scheduling tables. Instead, the attacker has to infer the current schedule from observations he or she makes about the system behavior. As we show in Section 3.2, schedules typically carry too little information to remain secure over extended periods of time even if this information is leaked only over low bandwidth channels. Furthermore, we assume that the global clock remains under exclusive control of the operating system and that it cannot be affected by the attacker.

Even though time-triggered systems eliminate CPU time as shared resource over which information can be leaked and through which other tasks may be influenced, other resources remain through which attackers may gain information and through which they can impact the

[3] Depending on the system configuration, both data and instruction caches may exhibit similar channels or interference possibilities (e.g., instruction cache evictions to maintain inclusiveness in the last-level cache). We shall therefore not further distinguish the type of cache.

timing behavior of other tasks. One prominent example of such a resource is the processor cache, which healthy tasks leave behind in a predictable state but which compromised tasks can put into a state that may not have been anticipated when computing the worst-case execution time of subsequent tasks.

The use of time-triggered systems imposes further limitation on attackers. For example, side channels and covert channels can only be constructed over explicitly or implicitly shared resources, most of which time-triggered systems already multiplex with the table driven schedule in a manner that is agnostic to the behavior of executing tasks. Access controls and partitioning techniques like cache coloring [29] or bank coloring [59] further constrain the attacker. However, each such countermeasure negatively impacts system performance and they are generally not complete.

For example Bechtel et al. [3] demonstrate a Denial-of-Service attack in caches that are not prepared for partitioning and that therefore retain shared resources (e.g., for cache-miss handling). Once exhausted by the attacker's accomplice task, these resources are no longer available to the victim, stalling its execution until all outstanding write-backs are handled. The authors only consider the accomplice and victim task to be present, both running on different cores and – except for sharing the cache – in isolation. However, usually there are multiple tasks running in a real-time system and the attacker may not hit the victim in all cases without previously aligning its execution to the victim task. If the attacker has inferred the schedule instead, he or she is able to discern the best point in time to attack (e.g. right before the victim task runs). The attacker can remain stealthy until the time of attack has come when the victim is most vulnerable.

In addition to the above, as we show in greater detail in Section 3.2, mitigating attacks may require avoiding tempting optimizations such as bounding the delay a task can impose through the cache by evaluating its execution pattern. Designers may be tempted to implement optimizations for the sake of increasing performance while neglecting security.

In summary, we assume an attacker who has infiltrated non-critical tasks of the system and wants to infer timing information (i.e. the system schedule) in order to mount a directed attack against a critical victim task.

## 3.2 Vulnerability Analysis

One of the main vulnerabilities of a time-triggered system lies in its *deterministic behavior*. The schedule is the same offline constructed schedule for every hyperperiod. For each point in time, the task executing is known. An attacker who listens to the schedule over a side channel is able to reconstruct the schedule in reasonable time even when the channel has low bandwidth. The schedule comprises only a few bytes of information, thus even with a very low channel bandwidth of, for example, 1 byte per second the schedule is found out in a matter of a few minutes. As we show in Section 7.2.3, an offline schedule of a real-world system can consist of just 52 bytes. Through the aforementioned channel, the attacker would know the schedule after one minute. Therefore, we reason that timing information can be inferred and focus on mitigating directed attacks under this assumption.

Another vulnerability of real-time systems in general is that *worst case execution time (WCET) derivation* does not take malicious behavior into account. WCET estimated through simulation of the expected behavior of the system does not account for malicious behavior. If a task is infiltrated at runtime and, as shown in [3], starts accessing the cache to create maximum interference for the next task execution, the tasks simulated worst case does not account for this malicious behavior if this behavior is not encountered during uncompromised execution. Prior research on abstract interpretation WCET derivation claims the assumption of cold caches is too pessimistic for a real system and shows methods to achieve tighter and less pessimistic WCET bounds [18], [11]. Such optimizations based on assumptions on task behavior increase the severity of this vulnerability. We have to choose WCET estimates in a way such that they also account for malicious behavior and we have to check the impact of performance optimizations on security.

Another countermeasure, typically applied for accidental faults, but found to be effective also against malicious faults aims to limit attacks only to up to $f$ out of $n$ replicas of critical services. This is achieved by tolerating the attack through the remaining $n - f$ replicas operating in consensus (e.g., triple-modular redundant systems [20] tolerate one fault with $n = 3$ replicas). The condition for such replicated systems to work is a synchronchonous invocation with the same (sensor) value such that healthy instances produce the same result in a timely manner, which forms the majority decision to apply. Assuming synchrony (i.e., reliable time), Byzantine fault tolerant state machine replication (BFT-SMR) [2, 28, 40] relaxes the first assumption of synchronous identical invocation at the cost of having to execute an initial agreement phase on a singular value or a vector median point (excluding up to $f$ outliers on both ends) [33].

Given that replicas should operate on the same data to produce (at least approximately) the same results, it is tempting to schedule them as a gang in the same slots on all cores they span. However, as we shall see in more detail in Section 6, this optimization is brittle and may lead to attacks. We shall see in particular that some attacks, like the above cache DOS attack by Bechtel et al. [3], bear the potential to cause common mode timing faults in all replicas simultaneously.

In the next section, we show mitigation strategies for directed attacks which prevent an attacker from exploiting the vulnerability which results from not taking malicious behavior into account.

## 4 Mitigation Strategies

An attacker's goal is to predict as precisely as possible when a victim task gets scheduled immediately after a compromised task to then mount a directed attack. Our primary mitigation strategy is therefore to impede predictions about the point in time when the victim is executed. While we do not prevent timing inference, i.e. we assume the attacker may gain information about the schedule, we are able to counter predictions by changing the points in time when tasks are executed at runtime. For this purpose, we present two strategies to mitigate directed attacks in this section. The first strategy takes an offline constructed time-triggered schedule as input and randomizes the schedule online at job-level while maintaining deadline constraints. The second strategy consists of a set of offline precomputed schedules one of which is randomly chosen at the end of each hyperperiod during runtime. Both strategies are presented in [27] and are implemented on each core of the system.

### 4.1 Slot-level Online Randomization

This mitigation strategy impedes the ability of an attacker to make predictions by randomizing job execution in a time-triggered system at runtime. Schedules for time-triggered systems are typically constructed offline [9], where real-time constraints are resolved and represented in a scheduling table. If not handled properly, online randomization may violate deadline constraints. Therefore, our approach analyzes the scheduling table offline and maps timing constraints of jobs onto execution windows. Execution windows are time intervals defined by the earliest start time of a job and its deadline. Each task has to finish execution within its execution window. Proper handling and, possibly, modification of execution windows solves precedence constraints. Additionally, if one of the goals of the system is to achieve low jitter, we can reduce the size of execution windows accordingly.

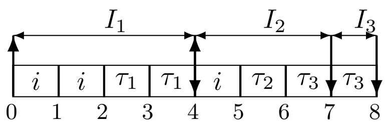
During runtime, we randomize job execution within their respective execution windows. While we confine jobs to their execution windows, they still share the same processor so we also have to guarantee that their execution does not lead to a deadline miss of other jobs. Slot shifting is a scheduling algorithm which introduces the concept of spare capacities to ensure timely execution [12]. We adopt this concept to guarantee task execution within their respective execution windows even though the scheduling decision is randomized.

### 4.1.1 Background

Slot shifting uses a discrete time model [24], where the time interval which separates two successive events (i.e. the granularity of the system) is called a slot [42]. We analyze the time-triggered schedule and its task set offline to determine available leeway and unused resources in the schedule for subsequent online adjustment. In order to track the available leeway of jobs in each execution window, a capacity interval is created for each distinct deadline in the system. Jobs with the same deadline belong to the same capacity interval. The start of a capacity interval $I_j$, $start(I_j)$, is defined as the maximum of the earliest start time $est(I_j)$ of jobs $\tau_i$ in this interval and of the end of the previous, i.e. preceding, capacity interval:

$$start(I_j) = \max(end(I_{j-1}), est(I_j)), \text{ where } est(I_j) = \min(est(\tau_i)) \ \forall \tau_i \in I_j \tag{2}$$

The end of the capacity interval is determined by the common deadline of all $\tau_i \in I_j$. If needed, empty capacity intervals without assigned jobs are created to fill gaps between capacity intervals with assigned jobs. Figure 1 shows an example job set derived from an offline schedule with earliest start times $est_i$, worst case execution times $C_i$ and deadlines $d_i$. We derive the presented schedule in Section 4.1.3. In the schedule presented in Figure 1, $i$ denotes the idle task. The schedule does not represent a hyperperiod as this is not necessary for illustratory purposes. The algorithm operates the same whether considering the hyperperiod or not.



| $\tau_i$ | $est_i$ | $d_i$ | $C_i$ |
|---|---|---|---|
| $\tau_1$ | 0 | 4 | 2 |
| $\tau_2$ | 0 | 7 | 1 |
| $\tau_3$ | 4 | 8 | 2 |

**Figure 1** Job set and capacity intervals derived from offline schedule

Three distinct deadlines exist for that job set, thus at least three capacity intervals have to be created. The first interval $I_1$ starts at 0 and ends at the deadline of its assigned set of jobs $\{\tau_1\}$, which is 4. The job assigned to next interval, $\tau_2$, shares the earliest start time of $\tau_1$, but according to Equation 2, a capacity interval is not allowed to start before the end of the previous interval. Note that capacity intervals do not overlap, while execution windows may. Thus, $I_2$ starts at 4 and ends at the deadline of its assigned set of jobs $\{\tau_2\}$, which is 7. We create interval $I_3$ accordingly. We show the resulting capacity intervals together with an exemplary schedule in Figure 1.

The spare capacity $sc(I_j)$ of a capacity interval $I_j$ is equal to the amount of free slots in $I_j$. $sc(I_j)$ is defined as the interval length minus the sum of worst case execution times $C_i$ of all its jobs $\tau_i$ minus slots borrowed from the succeeding interval (denoted as negative spare capacity), see Equation 3 below.

$$sc(I_j) = |I_j| - \sum_{\tau_i \in I_j} C_i + \min\left(sc(I_{j+1}), 0\right) \tag{3}$$

Spare capacities are calculated starting from the latest capacity interval in the hyperperiod to the earliest. Borrowing occurs in those cases where the current capacity interval provides insufficient slots to accommodate all its jobs, which results in a negative spare capacity ($I_3$ in Figure 2). Capacity intervals with a negative spare capacity borrow the needed amount of slots from the preceding interval. Negative spare capacities do not necessarily imply infeasibility in the scheduling sense. Spare capacities are a means to track "free" slots in a capacity interval. We show the resulting offline calculated spare capacities (for time $t = 0$) in Figure 2 of Section 4.1.3, where we present the calculation of spare capacities using Equation 3.

If we have calculated all spare capacities, the first capacity interval has a non-negative spare capacity provided the task set is schedulable. A task set is schedulable when its utilization is equal to or less than one since we consider each core of a partitioned multicore system separately. Positive spare capacities represent the amount of unused resources and leeway [12] of an interval which can be given to other tasks with overlapping execution windows to adjust the schedule. Such adjustments may require updating spare capacities. At runtime, we update the spare capacities after each slot to reflect the impact of scheduling decisions on the availability of "free" slots.

We consider three different cases for spare capacity updates:

1. No job executes in a given slot. In this case we have to decrease the spare capacity of the current capacity interval by one.
2. A job executes which belongs to the current capacity interval. In this case the spare capacity of the current interval does not change because the WCET of this job is already considered.
3. A job executes which belongs to a later capacity interval. In this case the current interval's spare capacity needs to be decreased by one, but executing the job ahead of time frees resources in its assigned interval. We can therefore increase the spare capacity of the job's interval by one. If this capacity increase happened on a negative spare capacity (i.e., the job's interval is borrowing from another capacity interval), we also increase the spare capacity from the interval from which it borrows, as it needs to lend one slot less. Cascaded borrows are resolved recursively in a similar fashion.

The original slot shifting algorithm in [12] and [42] further integrates aperiodic tasks into a time-triggered schedule. In this paper, we only adopt the concept of capacity intervals and spare capacities to guarantee timely execution of periodic jobs within their execution windows without violating constraints of other jobs. Thus, our offline algorithm needs to create only one table with execution windows and a second one with capacity intervals and their respective spare capacities. For our online randomizing scheduler, we update the spare capacities at runtime to keep track of scheduling decisions.

### 4.1.2 Slot-Level Randomization of Jobs

Our first attack mitigation strategy is to randomize job execution at runtime. Therefore, at the beginning of each slot, we invoke the online scheduler to select the next job from all tasks in the ready queue at random. We consider the idle task to be part of the ready queue in order to allow for more permutations of the schedule. Even though we select tasks randomly, we have to guarantee that no scheduled job violates the deadline constraints of other jobs. Thus, before taking a scheduling decision, we check if the spare capacity of the current capacity interval is greater than zero. If this condition is fulfilled, any job is allowed to run, as sufficient time remains in the current and later intervals such that no job misses its deadline. In other words, as long as the schedule has leeway, each ready job has the same probability of getting selected for a slot. Otherwise, if the spare capacity of the current interval drops to zero, there is no more leeway to schedule arbitrary jobs. However, because we have already considered jobs of the current capacity interval in the spare capacity computation and because all such jobs share the same deadline, we can still randomize their execution. That is, in the case of zero leeway, the online scheduler randomly selects among the jobs of the current capacity interval. After the job has run during its assigned slot, we update spare capacities as shown in Section 4.1.1.

Combining time-triggered scheduling with our slot-level randomization impedes online predictions about the schedule. Since the scheduler randomly selects the next job at runtime, predictions about which job runs next are not possible as long as execution windows allow for leeway. Furthermore, time-triggered scheduling inherently confines application-level leakage to shared resources which are held across slots [51]. An investigation of leakage countermeasures for such resources is out of the scope of this paper. While our randomization algorithm does not allow for slot-level determinism typical for time-triggered systems, it still allows for execution window determinism [13].

### 4.1.3 Example

Let us illustrate the proposed scheduling algorithm for our example jobset depicted in Figure 1. First, we have to calculate the initial spare capacities of the capacity intervals. Starting at the last capacity interval, $I_3$, its spare capacity is the difference between the interval length of 1 and the worst case execution time of its assigned jobs, $\tau_3$, which results in a spare capacity of $-1$. $I_2$ has an interval length of 3, from which we substract the worst case execution time of $\tau_2$ (i.e., $C_2 = 1$) and the slots borrowed by the preceding interval $I_3$ (by adding $sc(I_3) = -1$), which results in a spare capacity of 1. We calculate the spare capacity of $I_1$ accordingly. Figure 2 shows the resulting spare capacities in the column for time $t = 0$.

| time $t$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| $sc(I_1)$ | 2 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| $sc(I_2)$ | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 0 | 0 |
| $sc(I_3)$ | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 0 | 0 |

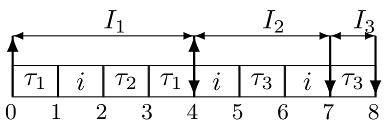


**Figure 2** Left: Spare Capacities of $I_1$, $I_2$ and $I_3$ over time, Right: Randomized Schedule

At time $t = 0$, the scheduler sees that the spare capacity of the current interval $I_1$ is positive and picks $\tau_1$ randomly for the first slot at $t = 0$ from the list of ready jobs $\tau_1$, $\tau_2$, plus the idle job $i$. As $\tau_1$ executes within its own interval, the current spare capacity does not change and remains positive. The idle job $i$ is selected to execute during the next slot starting at $t = 1$, necessitating a decrease of the spare capacity by one. $\tau_2$ is randomly selected for time $t = 2$. $\tau_2$ does not execute within its own capacity interval, therefore we reduce $sc(I_1)$ by one and increase $sc(I_2)$ by one, since $\tau_2$ belongs to interval $I_2$ and $I_2$ does not borrow from $I_1$. $sc(I_1) = 0$ at $t = 3$ constrains the online scheduler to select from the set of jobs $\{\tau_1\}$ that belong to $I_1$. $\tau_3$ becomes active at time $t = 4$ and is selected to execute at time $t = 5$ after picking the idle thread to run at $t = 4$. This is valid, as $sc(I_2)$ is positive, and thus we reduce $sc(I_2)$ by one and increase the capacity interval of $\tau_3$, $I_3$, by one. However, at this time, $I_3$ is still borrowing one slot from $I_2$. $\tau_3$ executed prior to its own capacity interval, thus $I_3$ needs to borrow one slot less from $I_2$ and therefore we increase $sc(I_2)$ by one, resulting in no change of $sc(I_2)$. In summary, $sc(I_2)$ stays at 1 and $sc(I_3)$ is increased by one. We show further exemplary scheduling decisions and the resulting spare capacity updates in Figure 2.

## 4.2 Offline Schedule-Diversification

The second mitigation strategy we investigate in this work precomputes multiple schedules offline and switches between them randomly at hyperperiod boundaries during runtime. Resolving scheduling constraints offline ensures lower runtime overheads, but increases the chance of attackers to guess the schedule and launch directed attacks. For example, repeating the same offline computed schedule several times allows an attacker to infer the schedule, as illustrated in [36], and to coordinate directed attacks from compromised tasks scheduled later in the same hyperperiod or in subsequent hyperperiods. To partially mitigate this threat vector, we randomly switch schedules at the end of each hyperperiod. As a consequence, even when the attacker is able to recognize different schedules and has enough memory available to store them, the more schedules have been generated, the harder it is for the attacker to recognize which schedule has been chosen for the current hyperperiod and the less time remains to perform a directed attack. In particular, if the attacker is not able to identify the current schedule in time for his attack, the attacker misses the opportunity to perform a directed attack. Additionally, carefully created execution windows solve deadline and precedence constraints.

We show in Section 7.2.5 that computing and storing all possible, feasible schedules in memory is impractical. However, in non-embedded systems (e.g., SCADA), we foresee the continuing generation of schedules in a non real-time subsystem (e.g., in a sufficiently protected external control station) and an update of the set of schedules downloaded to the real-time device. This way, once a new set of schedules has been produced (possibly by recombining precomputed and stored schedules), the real-time device can switch to the new set at the end of the hyperperiod. Double buffering, signing and encryption of schedules ensures that the current set of schedules remains valid while the system validates the confidentiality and integrity of the new schedules (e.g., in a background task). Irrespective of update possibilities, the selected subset of schedules out of the set of all feasible schedules for a given task set should impede directed attacks as much as possible. We present two criteria to select subsets that complicate directed attacks in addition to guaranteeing deadlines and respecting task precedence constraints.

### 4.2.1 Random Selection

For the sake of low implementation complexity, the subset can be selected randomly. That is, schedules are created randomly and checked to meet all scheduling constraints. The schedules fulfilling this requirement form the set of schedules for the system. Schedule creation is stopped after a certain number of feasible schedules has been constructed. We recommend this method for large subsets, when enough memory is provided to store a large number of different schedules. If the subset is large enough, the random selection process provides a set of schedules with a schedule entropy close to the set of all feasible schedules. Other criteria impose more constraints on the selection process and therefore increase its complexity.

### 4.2.2 Schedule Entropy

Another criterium for schedule selection is schedule entropy as presented in [58]. This metric makes use of an approximation of the Shannon entropy and can be used to quantify the diversity between schedules. Using an entropy-like function, we can quantify what is the diversity of a set of schedules and then we can compute the set of schedules that maximizes the diversity. This allows us to randomly pick a schedule in this set and give the attacker the least possible amount of information (because from the outside, the task set execution would seem as diverse as possible).

Clearly, deadline and period constraints limit the amount of diversity that can be achieved. In Section 5 we investigate the fundamental limitations to the achievable diversity that are imposed by the task set characteristics.

## 5 Fundamental limitations to diversity and randomization

In this section, we analyze the schedule of each core separately. For each of them, our final objective is to determine a set $\mathcal{K}$ of $k$ schedules that is as *diverse* as possible, keeping $k$ as small as possible. What we are trying to assess is how effective is the randomization, given that the schedules should respect the time constraints. We are then trying to determine how diverse can a generic set of schedules be, given the constraint that it preserves the deadlines of all the tasks in the task set.

The diversity of the set of selected schedules $\mathcal{K}$ can be measured in different ways. The authors of [58] propose the use of entropy-like functions to measure the diversity of the schedule set, the *slot entropy* and the *upper-approximated entropy*. In the following, we are going to borrow these definitions to evaluate the schedule set diversity. We first define the slot count $C_{j,i,\mathcal{K}}$, as the number of occurrences of a task $i$ in a given scheduling slot $j$ in the set $\mathcal{K}$, and then use that to formally define the slot entropy $H_j(\mathcal{K})$ and the upper-approximated entropy $\tilde{H}(\mathcal{K})$. We denote with $\phi(x)$ the function

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

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

$$C_{j,i,\mathcal{K}} = \sum_{s^{(q)} \in \mathcal{K}} \left[ s_j^{(q)} = i \right] = \sum_{q=1}^{k} \left[ s_j^{(q)} = i \right]. \tag{4}$$

Using the slot count, we can now formally write the slot entropy and the upper-approximated entropy according to the definitions given in [58].

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

$$H_j(\mathcal{K}) = \sum_{i=0}^{m} \phi\left( \frac{C_{j,i,\mathcal{K}}}{k} \right) = \sum_{i=0}^{m} -\frac{C_{j,i,\mathcal{K}}}{k} \cdot \log_2 \frac{C_{j,i,\mathcal{K}}}{k}. \tag{5}$$

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

$$\tilde{H}(\mathcal{K}) = \sum_{j=1}^{\ell} H_j(\mathcal{K}) = \sum_{j=1}^{\ell} \sum_{i=0}^{m} -\frac{C_{j,i,\mathcal{K}}}{k} \cdot \log_2 \frac{C_{j,i,\mathcal{K}}}{k}. \tag{6}$$

Now we can formally state the objective of minimizing the information that can be extracted from the system by observing its schedule. Given a task set $\mathcal{T}$ and the set of all valid schedules $\mathcal{S}$, what is the smallest subset of $\mathcal{S}$ that maximizes the upper-approximated entropy?

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

▶ **Problem** 4. Given a task set $\mathcal{T}$ and a valid set of schedules $\mathcal{S}$, solve

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

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

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

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

Problem 4 could be approached by an exhaustive search. If one evaluated the upper-approximated entropy for every element in the power set of $\mathcal{S}$, the optimal solution to Problem 4 could be obtained by choosing the smallest subset that achieves $\tilde{H}^*$. However since the cardinality of $\mathcal{S}$ is typically large, this will be infeasible in practice. Such an approach is also naïve. Afterall it seems improbable that the solution to Problem 4 would have cardinality one, so we could rule those subsets out of our exhaustive search.

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

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

▶ **Theorem 5.** *Given any schedule set $\mathcal{K} \subseteq \mathcal{S}$ for a constrained deadline task set,*

$$\tilde{H}(\mathcal{K}) \leq \ell \cdot \sum_{i=0}^{m} \frac{d_i}{t_i} \cdot \phi\left({}^{e_i}/_{d_i}\right) =: \tilde{H}^{ub} \tag{7}$$

**Proof.** The proof hinges on three key observations. The first is that since the function $\phi(x) = -x \cdot \log_2(x)$ is continuous and concave for all $x > 0$, given any set of positive weights $a_j \geq 0$,

$$\sum_{j=1}^{\ell} a_j \phi\left(\frac{C_{j,i,\mathcal{K}}}{k}\right) \leq \left(\sum_{j=1}^{\ell} a_j\right) \phi\left(\frac{\sum_{j=1}^{\ell} a_j C_{j,i,\mathcal{K}}/k}{\sum_{j=1}^{\ell} a_j}\right). \tag{8}$$

The second is that the $i$-th task can only be active during the first $d_i$ slots of the task period $t_i$. Therefore since $\phi(0) = 0$,

$$\phi\left(\frac{C_{j,i,\mathcal{K}}}{k}\right) = I_{j,i}\phi\left(\frac{C_{j,i,\mathcal{K}}}{k}\right), \tag{9}$$

where $I_{j,i} = 1$ if $\mod (j-1, t_i) < d_i$, and $I_{j,i} = 0$ otherwise (i.e. $I_{j,i}$ indicates whether the $i$-th task can be active in slot $j$). By setting $a_j \equiv I_{j,i}$, together (8)–(9) imply that

$$\sum_{j=1}^{\ell} \phi\left(\frac{C_{j,i,\mathcal{K}}}{k}\right) \leq \left(\sum_{j=1}^{\ell} I_{j,i}\right) \phi\left(\frac{\sum_{j=1}^{\ell} C_{j,i,\mathcal{K}}/k}{\sum_{j=1}^{\ell} I_{j,i}}\right). \tag{10}$$

The final observation is that

$$\sum_{j=1}^{\ell} \frac{C_{j,i,\mathcal{K}}}{k} = \ell\frac{e_i}{t_i}, \quad \sum_{j=1}^{\ell} I_{j,i} = \ell\frac{d_i}{t_i}. \tag{11}$$

These relations follow from the fact that $\mathcal{K}$ contains $k$ valid schedules (i.e. task $i$ is given $e_i$ slots per period $t_i$, and can only be active in the first $d_i$ slots). Substituting (11) into (10) shows that

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

The result follows by summing over all the tasks (c.f. the definition of the upper-approximated entropy). ◀

▶ Remark. Notice that in the case of an implicit deadline task set ($d_i = t_i, \forall i$), the condition of the theorem simplifies to

$$\tilde{H}(\mathcal{K}) \leq \ell \cdot \sum_{i=0}^{m} \phi\left({}^{e_i}/_{t_i}\right). \tag{12}$$

We now present two corollaries that link the upper-approximated entropy to aggregate characteristics of the task set, i.e., the number of tasks and the utilization.

▶ **Corollary 6.** *Given any schedule set $\mathcal{K} \subseteq \mathcal{S}$,*

$$\tilde{H}(\mathcal{K}) \leq -\ell \cdot \log_2\left({}^{1}/_{(1+m)}\right). \tag{13}$$

**Proof.** Knowing that ${}^{e_i}/_{t_i} > 0$, $\phi({}^{e_i}/_{t_i}) = -{}^{e_i}/_{t_i} \cdot \log_2({}^{e_i}/_{t_i})$ is continuous and concave. This implies that ${}^{1}/_{(m+1)} \sum_{i=0}^{m} \phi({}^{e_i}/_{t_i}) \leq \phi\left({}^{1}/_{(m+1)} \sum_{i=0}^{m} {}^{e_i}/_{t_i}\right)$. This gives us

$$\sum_{i=0}^{m} \phi({}^{e_i}/_{t_i}) \leq (m+1)\,\phi\left({}^{1}/_{(m+1)}\right) = -\frac{m+1}{m+1} \cdot \log_2\left({}^{1}/_{(m+1)}\right).$$

◀

The expression in Equation (13) is not always reachable, depending on the characteristics of the task set. This leads us to consider the task set characteristics. In particular, we can compute a bound that takes into account the utilization $U$ of the task set, leading to the following corollary.

▶ **Corollary 7.** *Given any schedule set $\mathcal{K} \subseteq \mathcal{S}$,*

$$\tilde{H}(\mathcal{K}) \leq \ell \cdot \{-(1-U) \cdot \log_2(1-U) - U \cdot \log_2({}^{U}/_{m})\}. \tag{14}$$

This corollary can be proven by splitting the contribution of the idle task and the regular tasks in the task set. The contribution of the idle task to the upper-approximated entropy is equal to $\ell \cdot \{-(1-U) \cdot \log_2(1-U)\}$. The maximum value for the upper-approximated entropy is reached when the utilizations of the tasks allow them to be evenly distributed. The contribution of each of them is then $-\ell \cdot {}^{U}/_{m} \cdot \log_2({}^{U}/_{m})$. Deriving the expression in Equation (14) allows us also to study when we can be closer to the bound in Equation (13) — and when can we expect to reach the maximum upper-approximated entropy for a set of $m$ tasks, depending on the task characteristics. With respect to the utilization $U = \sum_{i=1}^{m} {}^{e_i}/_{t_i}$, the upper approximated entropy can reach its maximum when the utilization of the system is equal to $U = {}^{m}/_{(1+m)}$.

We will now show that a given schedule set $\mathcal{K} \subseteq \mathcal{S}$ can only achieve an upper-approximated entropy of $\tilde{H}^{ub}$ (from Equation (7)) if all the tasks have implicit deadlines and the cardinality of $\mathcal{K}$ is at least

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

This is important, since it shows that if we want to achieve the upper bound on the upper-approximated entropy from Theorem 5, we must enforce $d_i = t_i, \forall i$ and use a schedule set of at least the size given above.

▶ **Theorem 8.** *Given any schedule set $\mathcal{K} \subseteq \mathcal{S}$ for a constrained deadline task set, if there exists a task $p$ such that $d_p < t_p$ then the inequality in Equation* (7) *is strict. Furthermore, if all the tasks have implicit deadlines ($d_i = t_i, \forall i$) and*

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

*then the inequality in Equation* (7) *is strict.*

**Proof.** The upper-approximated entropy is the sum of the contribution of each slot. Therefore, to achieve the upper bound $\tilde{H}^{ub}$, we should maximize all the summands – i.e., the contribution to the upper-approximated entropy of each slot should be maximized.

It follows from Jensen's inequality and Equation (11) that the inequality in Equation (8) is strict unless

$$C_{j,i,\mathcal{K}}/k = \alpha \tag{15}$$

for all $j$ such that $I_{j,i} = 1$ (reusing the notation of the proof of Theorem 5). The constant $\alpha$ can be found using Equations (10)–(11). In particular, Equation (10) implies that

$$\sum_{j=1}^{\ell} C_{j,i,\mathcal{K}}/k = \alpha \sum_{j=1}^{\ell} I_{j,i} = \alpha\ell\frac{d_i}{t_i} \underset{\text{Equation (11)}}{\Longrightarrow} \alpha = \frac{e_i}{d_i}$$

Since every task is active in the first slot, this implies that the inequality in Equation (7) is strict unless

$$C_{1,i,\mathcal{K}}/k = \frac{e_i}{d_i}. \tag{16}$$

To prove the first part of the theorem, suppose that $d_i < t_i$ for some $i$. Now assume that Equation (16) holds. This implies that

$$\sum_{i=0}^{m} C_{1,i,\mathcal{K}}/k = \sum_{i=0}^{m} \frac{e_i}{d_i} > 1.$$

However $\sum_{i=0}^{m} C_{1,i,\mathcal{K}} = k$, since every slot in the schedule set is assigned to one task. This is a contradiction.

To prove the second part, now suppose that $d_i = t_i$ for all $i$. The contribution of each task to the slot entropy should then be equal to the task utilization ${}^{e_i}/_{t_i}$ – notice that this includes the idle task. To find a lower bound for $k$, we can then look at a single slot $j$. In fact, additional slots will only potentially increase the number of schedules needed to achieve higher upper-approximated entropy. We can then formulate the problem of finding $|\mathcal{K}^*| = k^*$ as an optimization problem.

$$\min_{\substack{k \in \mathbb{Z}^+ \\ C_{j,i,\mathcal{K}} \in \mathbb{Z}^+}} \quad k \quad \text{s.t.} \quad C_{j,i,\mathcal{K}} = \frac{e_i}{t_i} \cdot k, \quad \forall i \in \{0, \ldots, m\}. \tag{17}$$

This means that we have a positive integer number of schedules in the set $\mathcal{K}$ that allows $C_{j,i,\mathcal{K}}$ (the number of times task $i$ appears in slot $j$ in set $\mathcal{K}$) to be positive a integer number for each task $i$. We perform a variable substitution and define $y = \ell/k$. Minimizing $k$ now becomes equivalent to maximizing $y$. Relaxing temporarily the requirement that $k$ be an integer, the problem in Equation (17) is reformulated as

$$\max_{C_{j,i,\mathcal{K}} \in \mathbb{Z}^+} \quad y \quad \text{s.t.} \quad C_{j,i,\mathcal{K}} = \frac{e_i}{t_i} \cdot \frac{\ell}{y}, \quad \forall i \in \{0, \ldots, m\}. \tag{18}$$

The solution to the problem in Equation (18) is that $y$ should be equal to the greatest common divisor of the utilizations multiplied by the hyperperiod, $y = \gcd({}^{e_i}/_{t_i} \cdot \ell)$, which yields to

$$k^* = \frac{\ell}{\gcd\left({}^{e_i}/_{t_i} \cdot \ell\right)}. \tag{19}$$

For this to be the solution of the optimization problem in Equation (17), $k^*$ must be an integer number. Because the utilizations of the task set (including the idle task) sum to one, $\sum_{i=0}^{m} {}^{e_i}/_{t_i} = 1$, the constraint for the idle task in the optimization problem of Equation (18) can also be written as

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

The solution of problem (18) ensures that $\frac{\ell}{y} \cdot \sum_{i=1}^{m} {}^{e_i}/_{t_i} \in \mathbb{Z}^+$ due to the $m$ constraints for the (non idle tasks in the) task set. The value of $\ell/y$ (equal to $k^*$) must then be a positive integer number. This implies that the solution of the problem in Equation (18) is also a solution of the problem in Equation (17) and $k^* \in \mathbb{Z}^+$. ◀

▶ **Corollary 9.** *A simple modification of the proof of Theorem 8 shows that for any $\mathcal{K} \subseteq \mathcal{S}$, if*

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

*then $\tilde{H}(\mathcal{K}) < \tilde{H}^{ub}$. This means that the cardinality of $\mathcal{K}$ must be a multiple of $\frac{\ell}{\gcd({}^{e_i}/_{t_i} \cdot \ell)}$ in order to achieve the bound in Theorem 8.*

## 6 Extension to replicated systems on multicores

So far, we have assumed that accomplices of attackers are limited to the tasks of non-critical subsystems. Through the use of replication, we can consider stronger adversaries, capable of compromising also up to $f$ replicas of an $n$-fold replicated critical subsystem. This leads to the following additional attack vectors:

AV1: A compromised task or replica attacks the task executing next on the same core.
AV2: A compromised task or replica attacks replicas executed in parallel on another core.
AV3: A combination of both attack vectors AV1 and AV2, i.e., attacks are mounted from one core shortly before a replica is run on another core.

The challenge with AV2 and AV3 lies in ensuring that at most $f$ replicas can be affected in any cycle required to rejuvenate all $n$ replicas, returning them to a healthy state [45, 46]. In particular in real-time systems, where results must be timely, systematic delays of more than $f$ replicas (minus those already compromised) may turn out fatal, as this allows compromised but timely replicas and consequently the attacker to gain control. Extending offline and online schedule-diversification allows us to protect against the above threat vectors.

### 6.1 Offline Schedule-Diversification

Offline schedulers posess the advantage to solve complex constraints before runtime. In addition to the deadline constraints already discussed in Section 4.1.1 (and possibly further constrains to accomodate for precedence or jitter), we shall impose as constraint that at the end of each slot, no two cores switch to the same job. This way, schedules across cores are diverse and we mitigate AV2, by avoiding replica execution at the same time.

However, the offline constructed schedule still provides the determinism an attacker can exploit for AV1 and AV3. As the constructed schedules are diverse, each cores RTOS stores all schedules. The RTOS instances on these cores switch at the end of the hyperperiod to another schedule in an offline defined, deterministic way, such that each core executes a different schedule. This still provides deterministic execution to the attacker, however the window of repetition, that is, the size of the hyperperiod is larger.

Alternatively, the scheduler may choose the next schedule at the end of the hyperperiod at random. If the set of diverse schedules is diverse enough, the possibility of retrieving information using the schedule is low. In addition, if the upper-approximated entropy bound given by Equation (7) is reached for the set of schedules that is used for schedule diversification, it is possible to state that there is no additional diversity that can be added by varying the schedule.

Cores may execute the same schedule at the same time and thus replicas may be executed in parallel, but the attacker is not able to rely on this possibility in a deterministic fashion.

## 6.2 Online Slot-Level Randomization

Online slot-level randomization, as discussed in Section 4.1.2, prevents predictions about the points in time when replicas are executed in parallel or on other cores and thus mitigates all attack vectors, AV1 to AV3. The attacker is unable to predict system behavior and cannot rely on deterministic assumptions to coordinate his or her attack. This strategy prevents the stealthy attack we consider for all three attack vectors.

## 6.3 Combining Offline and Online Strategies

In this approach, we combine both strategies presented in Section 6.1 and Section 6.2. First we create non-overlapping execution windows of replicas offline, i.e., replica execution windows are restricted not to overlap each other to fulfill this constraint. As consequence, the replicas have different earliest start times and deadlines on each core. This constraint eliminates AV2, where replicas are scheduled at the same time. Then, during runtime, we use slot-level online randomization to prevent predictions and thus eliminate attack vectors AV1 and AV3 enabled by a deterministic schedule. Compared to the pure online approach presented in Section 6.2, the combined strategy never schedules replicas in parallel. However, depending on the task set, the restriction of replica execution windows may leave few leeway in the schedule for the online randomization algorithm.

## 7 Experiments

We evaluate our mitigation strategies and insights on schedule entropy with experiments. First, we present a search algorithm that generates a set of schedules for each synthetic task set under the constraint that it maximizes the task sets upper-approximated entropy. We investigate the limits on randomization and entropy through the achievable entropy and its variance. Second, we employ the ROSACE case study [38] to evaluate our attack mitigation strategies. The ROSACE case study provides task set parameters for a safety-critical real-time system of an aircraft called the longitudinal flight controller. We provide an analysis of runtime overhead and memory cost.

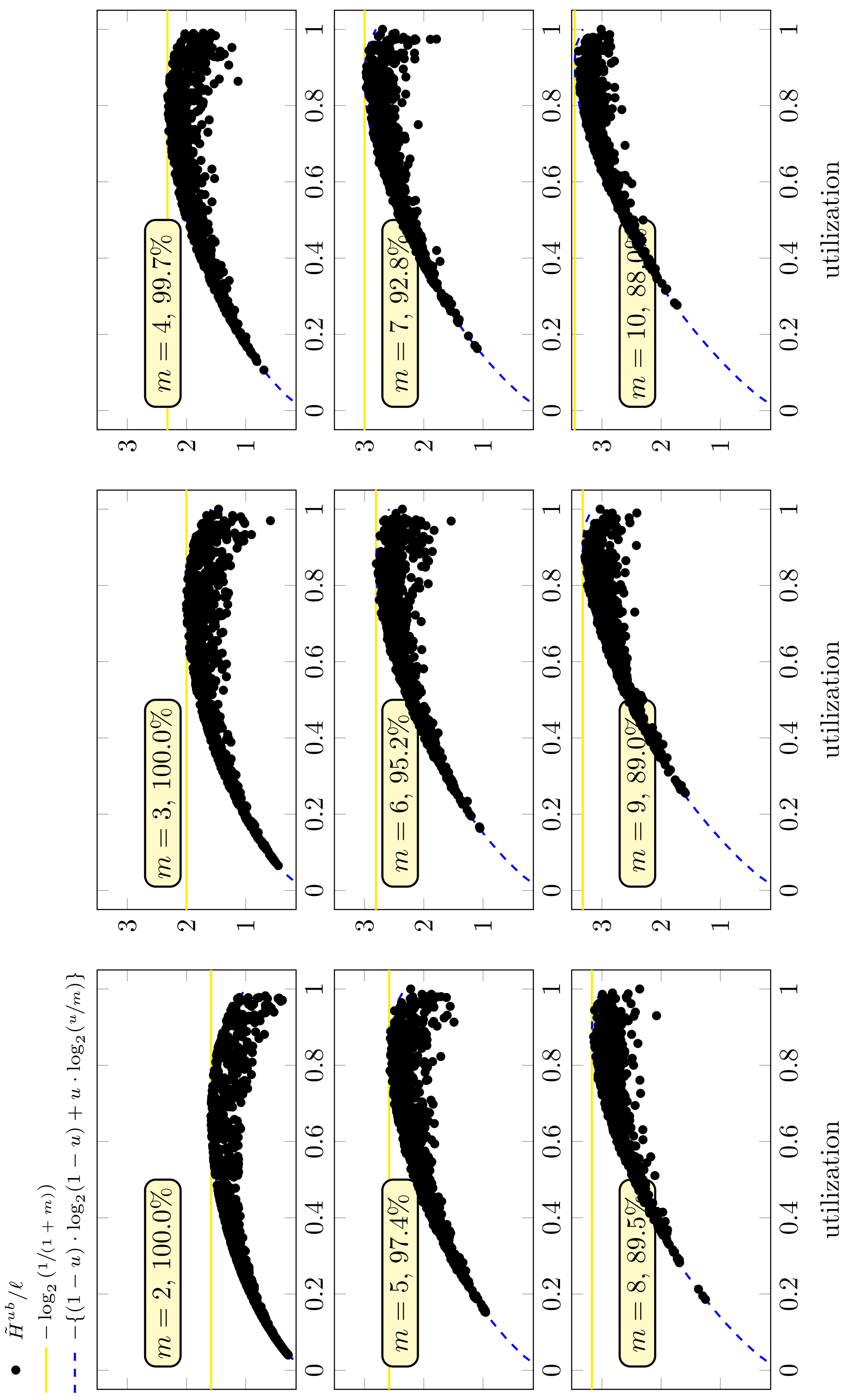


**Figure 3** Average slot entropy of random sets composed of $m$ tasks with $m \in \{2, \ldots, 10\}$ and maximum hyperperiod $\ell = 100$.

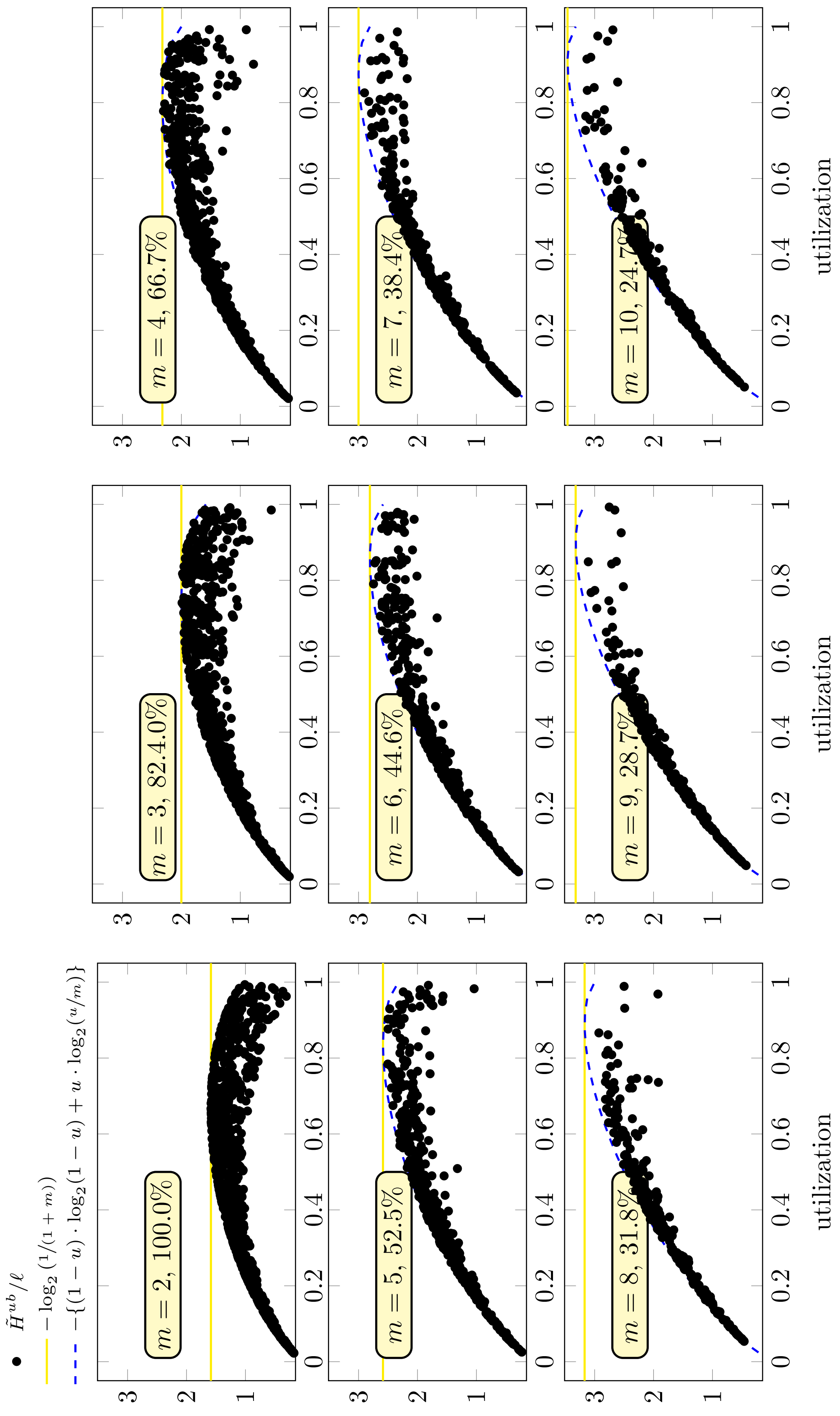


**Figure 4** Average slot entropy of random sets composed of $m$ tasks with $m \in \{2, \ldots, 10\}$ and maximum hyperperiod $\ell = 500$.

## 7.1 Synthetic Task Sets

In order to investigate the limits on schedule randomization and schedule entropy presented in Section 5, we implemented a search algorithm based on constraint optimization that generates a set of schedules maximizing the upper-approximated entropy for a given task set.[4]

We looked at each core separately and tested the algorithm with synthetic task sets, composed of $n$ tasks with $n \in \{2, \ldots, 10\}$ and maximum hyperperiod $\ell \in \{100, 200, 300, 400, 500\}$. For each of the possible combinations, we generated 1000 task sets using an extension of the UUniFast algorithm [4], that allows us to control the maximum hyperperiod. We randomized the periods $t_i$ of each task, and imposed an implicit deadline constraint $t_i = d_i, \forall i \in \{1, \ldots, n\}$. We computed the upper-approximated entropy $\tilde{H}^{ub}$, for the task set according to Theorem 5. We then computed the average contribution of each slot $\tilde{H}^{ub}/\ell$, to be able to compare task sets with different hyperperiods. Finally, we ran our algorithm to generate the schedule set $\mathcal{K}$, with the lowest cardinality $k^*$ given by Theorem 8.

In principle, the algorithm could run for a long time to find the maximum. There is also the possibility that the maximum is not reachable with any schedule set that satisfies all the constraints (in our experimental campaign, we did not encouter such a set, but our theoretical results do not exclude this possibility). Therefore, we limited the duration and allowed a budget of 60 minutes to compute the schedule set.

We define the algorithm *accuracy* as the percentage of task sets for which the algorithm found an optimal schedule set. In Figures 3 and 4, we show the results for the cases with $\ell = 100$ and $\ell = 500$, respectively. In particular, we show the average contribution of each slot to the upper-approximated entropy $\tilde{H}^{ub}/\ell$. In each plot, we write the number of tasks $m$ and the algorithm accuracy.

The dots in the Figures 3 and 4 show the tight bounds and the achievable entropy, while the dashed line represents the (non-tight) bound discussed in Corollary 7, Equation (14). As can be seen, for some of the task sets, the constraints imposed by the execution times and the periods allow the upper-approximated entropy to reach this bound, but in other cases the bound presented in Theorem 5 is tighter. One can also notice that the variance of the achieved upper-approximated entropy increases when the utilization increases. This is because a higher utilization introduces tighter constraints on the achievable entropy.

Another important result is the fact that the upper-approximated entropy reaches the maximum bound discussed in Corollary 7 for $m/(m+1)$. The implication of this is that from an entropy perspective the optimal system utilization is $m/(m+1)$. This is very important for determining critical system parameters like the task set utilization for security-aware embedded and real-time systems.

Finally, the yellow solid lines represent the (non-tight) bound introduced by Corollary 6. In most cases, this bound is unreachable, due to the constraints introduced by the tasks characteristics.

## 7.2 Real-world case study (ROSACE)

We evaluated our two directed attack mitigation strategies presented in Section 4 using the ROSACE case study [38]. ROSACE is a practical, real-world example of a real-time system in a safety-critical avionics environment. Pagetti et al. [38] carried out a case study of a longitudinal flight controller of an aircraft. The longitudinal flight controller helps the pilot to accurately track altitude, vertical speed and airspeed of the aircraft. Pagetti et al. describe two control loops: the

[4] The code for the algorithm is available at `https://gitlab.control.lth.se/NilsVreman/rand-sched`, together with the full set of results.

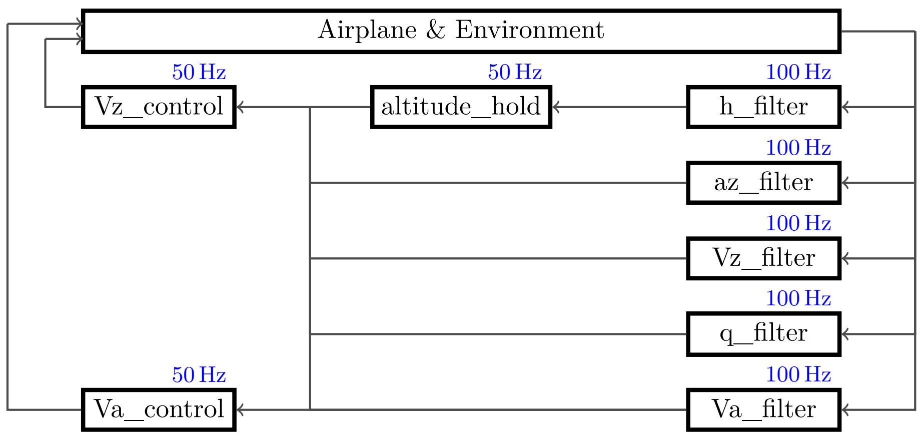


**Figure 5** Longitudinal flight controller design

**Table 1** Flight controller task set [38]

| Taskname | Frequency | WCET |
|---|---|---|
| Vz_control | 50Hz | 100$\mu$s |
| Va_control | 50Hz | 100$\mu$s |
| altitude_hold | 50Hz | 100$\mu$s |
| h_filter | 100Hz | 100$\mu$s |
| az_filter | 100Hz | 100$\mu$s |
| Vz_filter | 100Hz | 100$\mu$s |
| q_filter | 100Hz | 100$\mu$s |
| Va_filter | 100Hz | 100$\mu$s |

**Table 2** Execution windows in terms of slots

| Name | Start | End | WCET |
|---|---|---|---|
| h_filter | 0 | 50 | 1 |
| az_filter | 0 | 50 | 1 |
| Vz_filter | 0 | 50 | 1 |
| q_filter | 0 | 50 | 1 |
| Va_filter | 0 | 50 | 1 |
| h_filter | 50 | 100 | 1 |
| az_filter | 50 | 100 | 1 |
| Vz_filter | 50 | 100 | 1 |
| q_filter | 50 | 100 | 1 |
| Va_filter | 50 | 100 | 1 |
| altitude_hold | 0 | 100 | 1 |
| Vz_control | 0 | 100 | 1 |
| Va_control | 0 | 100 | 1 |

$Va$*_control* loop handles airspeed control by maintaining the desired airspeed $Va$; the second control loop — altitude control — combines *altitude_hold* and $Vz$*_control*. First, *altitude_hold* translates altitude commands to vertical speed commands. Then, $Vz$*_control* tracks the vertical speed $Vz$ of the aircraft. Both control loops are fed with filtered data: $h$, $az$ and $q$ for altitude, vertical acceleration and pitch rate, respectively. Vertical airspeed $Vz$ and true airspeed $Va$ are also inputs to the control loops. We show the design of the controller in Figure 5.

According to Pagetti et al. [38], the closed-loop system with continuous-time controllers can tolerate delays of up to roughly 1 second before destabilizing. To preserve stability as well as to increase performance, Pagetti et al. chose lower sampling periods of 50 Hz for the digitalization tasks of the three controller blocks and 100 Hz for the filter tasks which feed the data to the controller. Pagetti et al. derived worst case execution times of all tasks using a measurement-based approach by measuring the repeated execution of a task in isolation. The granularity the authors chose for the measuring clock was 100$\mu$s, thus the worst case execution times for the tasks shown are the same as they presumably finished execution in that granule. Table 1 shows the task set with implicit deadlines for the longitudinal flight controller. In this work, we do not consider environment simulation tasks as they are not part of the controller but only of the test environment.

We construct the execution windows of all tasks from the task set in Table 1. Schorr [42] suggests 200,000 clock cycles as slot shifting slot length. The processor cores in ROSACE run at 1.2GHz, which results in 167 $\mu$s for 200,000 clock cycles. We choose 200 $\mu$s as slot length to evenly divide the task periods into slots. Task execution is non-preemptive, as the worst case execution times are smaller than the slot length. Table 2 shows the resulting execution windows.

### 7.2.1 Runtime Overhead for Slot-Level Randomization

Our slot-level randomization algorithm is based on Schorr's [42] slot shifting algorithm. Schorr measured the runtime overhead of the unmodified slot shifting algorithm on a cycle-accurate ARM quadcore simulator — MPARM — with ARM7 cores running at 200 Mhz, 8kB 4-way set associative L1 cache, 8kB direct mapped L1 instruction cache, 1MB core-private memory and 1MB shared memory. Schorr provided minimum and maximum runtimes of all parts of the slot shifting algorithm for single core execution. Using the timing measurements of [42], shown in Table 3, we approximate the runtime overhead of slot-level randomization, when executed on the same processor.

**Table 3** Minimum and maximum runtime overhead per slot for single core execution in ns [42]

| **Function** | **Min** | **Max** |
|---|---|---|
| update spare capacity ($up_{sc}$) | 2,655 | 10,145 |
| update ready list ($up_{ready}$) | 3,500 | 9,115 |
| next job selection ($sel$) | 1,850 | 2,350 |
| ISR overhead ($ISR$) | 2,560 | 3,120 |

Slot-level randomization invokes the same functions to update spare capacities and the ready list. The cost of the function to update spare capacities increases linearly with the number of intervals due to cascaded borrowing in the worst case. However, according to the slot shifting algorithm as explained in Section 4.1.1, only 2 intervals are created for the presented task set. Hence, the costs of both functions remain the same. The interrupt service routine (ISR) overhead is architectural and hence should not change for an implementation of slot-level randomization in the same operating system. Randomization is not part of slot shifting and as such not covered by the above measurements. As calculating random numbers for each slot is independent of parameters like the number of tasks or intervals, we assume a constant per slot overhead. Moreover, assuming an O(1) `get_length` implementation of the ready list, pruning random values to a list index remains a constant operation.

We calculate the maximum runtime overhead as:

$$t_{ov,rand,max} = rand_{max} + up_{sc,max} + up_{ready,max} + sel_{max} + ISR_{max} \qquad (20)$$

Accordingly, the minimum runtime overhead results in:

$$t_{ov,rand,min} = rand_{min} + up_{sc,min} + up_{ready,min} + sel_{min} + ISR_{min} \qquad (21)$$

Using the measurements from Table 3 for equation 20 and assuming $rand_{max} = 5,000ns$, the maximum runtime overhead results in $t_{ov,rand,max} = 29,730ns$, which is around 3 percent of the assigned slot size of 1ms in [42]. Keeping in mind that ROSACE uses 6 times faster cores than [42] and that execution time does not scale exactly linear with processor speed, we can approximate the runtime overhead for ROSACE. Therefore, we divide these values by 5 for a core with 1.2 Ghz and approximate the maximum runtime overhead for ROSACE to be $t_{ov,rand,max} = 6,000ns$.

Under the assumption that $rand_{min} = 2,000ns$, the minimum runtime overhead results in $t_{ov,rand,min} = 12,565ns$, which is around 1.3 percent of the slot size in [42]. Dividing these values by 5 as explained earlier, we approximate the minimum runtime overhead for ROSACE to be $t_{ov,rand,min} = 2,500ns$.

### 7.2.2 Runtime Overhead for Offline Precomputed Schedules

The runtime overhead for offline precomputed schedules is lower than that of scheduling algorithms which have to take more complex decisions online, which we also prove in this section. Again we can make use of the overhead measurements done in [42], which we show in Table 4.

**Table 4** Minimum and maximum runtime overhead for single core execution in ns [42]

| **Function** | **Min** | **Max** |
|---|---|---|
| next job selection (*sel*) | 1,850 | 2,350 |
| ISR overhead (*ISR*) | 2,560 | 3,120 |

At runtime, the scheduler performs a table lookup to select the next job after each slot. In constrast to the slot-level randomization scheduling algorithm, the overhead only consists of the next job selection and the interrupt service routine. At the end of the hyperperiod, we select the next offline precomputed schedule randomly. We calculate best and worst case runtime overhead for selecting a precomputed schedule in MPARM as shown below.

$$t_{ov,prec,max} = rand_{max} + sel_{max} + ISR_{max} = 10470ns \tag{22}$$

$$t_{ov,prec,min} = rand_{min} + sel_{min} + ISR_{min} = 6410ns \tag{23}$$

Using the same estimation on the execution time of the randomization function for the ROSACE case study as in Section 7.2.1, best and worst case approximated overhead results in 1300 ns and 2100 ns, respectively. Thus, around 1 percent of the chosen slot size is used for scheduling for both ROSACE and on the ARM simulator MPARM.

### 7.2.3 Memory Cost for Offline Precomputed Schedules

Each precomputed schedule needs to be stored in memory. For ROSACE, we can build an offline schedule in the same way as shown in Table 2. Each task has its own task ID, an entry for the start and end of the execution of its instance, and a fourth entry for its worst case execution time. The difference between start and end time must be equal to its worst case execution time and the execution windows for different jobs must not overlap. Table 5 shows an example for a precomputed time-triggered schedule.

**Table 5** Exemplary precomputed time-triggered schedule for ROSACE

| ID | 0 | 1 | 2 | 3 | 4 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Start | 1 | 8 | 22 | 33 | 35 | 51 | 58 | 66 | 67 | 71 | 80 | 88 | 94 |
| End | 2 | 9 | 23 | 34 | 36 | 52 | 59 | 67 | 68 | 72 | 81 | 89 | 95 |
| WCET | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Assuming each entry has the size of 1 byte, a single schedule with this information needs $13 * 4 = 52$ bytes of memory. Theorem 8 and Corollary 9 state that we need only $k^*$ schedules to achieve the maximum diversity. In the case of ROSACE, this number is 100. Thus, we are able to calculate the total memory cost to store all these schedules, which amounts to $100 * 52 = 5200$ bytes. This example also shows that the memory cost scales linearly with the number of tasks involved. Each instance of a periodic task, i.e. each job adds 4 entries to this table.

### 7.2.4 Upper-approximated Entropy Analysis

Here we derive the actual numbers for the ROSACE case study, determining the limitations of the achievable upper-approximated entropy. We first look at the specifics of the given task set, and all the individual task utilizations, using Theorem 5. Then we look at a generic task set with the same utilization as the ROSACE utilization and apply the results of Corollary 7. Here, we see how ROSACE compares with other task sets with the same utilization (unsurprisingly, ROSACE does not have the "perfect" workload distribution to achieve the highest entropy). Notice that our solution is optimal in the space of solutions that satisfy the ROSACE schedulability.

Using a slot length of $200\,\mu$s, we can compute the hyperperiod $\ell = 100$ slots and the minimum number of schedules $k^*$ needed to achieve the maximum value of the upper-approximated entropy according to Equation (18), which amounts to 100 schedules. We use the tool presented in Section 7.1 to generate a set of 100 schedules that achieves the maximum upper-approximated entropy. The generated set reaches a value of $\tilde{H}^* = 93.8495$, which is the optimal value that can be reached for the characteristics of the given task set. This means that each of the schedule slots contributes with a diversity of 0.9385. This number is, again, optimal with respect to the utilizations and periods of the task set of this specific case study.

We now turn to the question how good the given scenario performs in terms of diversity compared to what is possible for task sets of the same utilization. For that, we can look at Corollary 7. The maximum upper-approximated entropy contribution per slot is 0.9474, given by Equation (14). This means that the distribution of the tasks that we find in our scenario is very close to the optimum that can be achieved at the given utilization level.

### 7.2.5 Discussion

Real-time systems are often implemented as embedded systems. As such, they are not only subject to size, weight and power considerations, but also have only limited memory available. Low memory cost and low computation overhead become even more important for these constrained systems. We will analyze our mitigation methods with respect to these constraints.

Slot-level randomization proves to be practical, as the approximated overhead in Section 7.2.1 shows. In the worst case, slot-level randomization uses less than 3 percent of the slot size for scheduling. Precomputing offline schedules can further reduce this overhead to roughly 1 percent of the slot size, but physical memory capacity limits the number of offline precomputed schedules that can be stored in a system. In general, it is possible to offload scheduling tables to secondary storage by accepting an increase of scheduling overhead while loading the selected scheduling table from this memory. However, we know from Theorem 8 and Corollary 9 that we need only $k^*$ schedules to achieve the maximum diversity and we are able to calculate this number. In the case of ROSACE, we need 100 schedules to achieve maximum diversity, which amounts to 5200 bytes of required memory storage (see Section 7.2.3 for details).

As we mentioned in Section 3.2, an attacker might identify a small number of schedules after several minutes or a few hours even for side channels with low bandwith. The attacker might even be able to derive a minimal set of schedules that achieves maximum diversity, as scheduling parameters might be known, derived from system observation or reverse engineered. However, this set is not unique, i.e. different sets of schedules are able to achieve maximum diversity. Even under the assumption that the attacker is able to store a huge number of schedules, the higher the number of precomputed schedules, the longer it takes for the attacker to be sure which schedule is used. Updating the stored scheduling tables partially mitigates the threat that the attacker might eventually identify the schedule in time. The threat is fully mitigated with slot-level randomization, which we recommend in general, due to the comparable overhead, and for systems with strict memory constraints.

In order to show how many possible schedules slot-level randomization covers, we calculate the total number of possible feasible schedules for the task set presented in Table 2. For each execution window, the binomial coefficient $\binom{n}{k}$ calculates the number of possibilities to execute the task in different slots, where $n$ is the window size and $k$ the worst case execution time, both quantified in slots. The binomial coefficients of neighbouring and overlapping execution windows are multiplied with each other. If execution windows overlap, we subtract the worst case execution time of tasks belonging to execution windows whose binomial coefficients are already accounted for in the equation ("preceding" binomial coefficients) from the window size. Thus, we calculate the number of possible feasible schedules for the presented task set as shown below. On the left side of the equation, the binomial coefficients of the five tasks with periods of 50 slots are calculated two times, because the hyperperiod results in 100 slots. Their combined worst case execution time of 10 slots is then substracted from the execution window sizes of the tasks with a period of 100 slots.

$$\left[\binom{50}{1}\binom{49}{1}\binom{48}{1}\binom{47}{1}\binom{46}{1}\right]^2 \times \binom{90}{1}\binom{89}{1}\binom{88}{1} = 4.56 \times 10^{22} \tag{24}$$

$4.56 \times 10^{22}$ schedules with 52 bytes require $2^{81}$ bytes of storage, so we can safely conclude that it is infeasible to track or store all possible schedules in terms of memory space and computation time needed. Positive spare capacities, i.e. leeway in the schedule, are key for a high number of distinct feasible schedules.

## 8 Related Work

Security is a major concern for real-time and control systems [15, 16, 48–50, 57]. Modern embedded systems are vulnerable to many different security threats [23, 39], one of them being side-channel attacks [47]. Side-channel attacks are based on attackers gathering knowledge about a system, and exploiting this knowledge to influence its behavior [1, 34, 41, 44, 47]. For example, recently, a team of researchers showed that it is possible to retrieve the engine speed from the frequency of execution of its control task [31]. In general an attacker knowing the schedule of an embedded control system can infer that the controller is sending a control signal to a plant periodically in predictable time slots. They can then use this knowledge to jam the network only when the control signal is being transmitted. Reducing the need for the attacker to be active also reduces the possibility of detecting the ongoing attack.

Several security solutions exist which prevent information leakage in real-time systems. For example, Völp et al. show in [52] how to prevent timing leaks in fixed-priority schedulers by exploiting the idle task to mask early stops or blocks of a high priority task such that a low priority task always has the same view of the high priority task. Naturally, time-triggered systems do not require this modification since no two tasks execute in the same time window on the same processor. In [36], Mohan et al. focus on the problem of information leakage over shared resources. They define security levels for tasks and prevent undesirable information flow between tasks of different security levels by flushing the resource. Further, they discuss the integration of security constraints into the design of fixed-priority schedulers. In contrast to [52] and [36], we consider time-triggered systems which have no concept of task priority. Additionally, we do not focus on preventing timing channels or information leakage. In fact, we assume timing information, in particular task set parameters, may be inferred.

One of the logical countermeasures against this type of attacks is to impede the information gathering phase. In particular, an attacker who observes the execution of the real-time system should not be able to get timing information beneficial for an attack. However, classical scheduling algorithms are designed exactly for predictability and repeatability. Schedules (for periodic task sets) usually repeat after a predetermined amount of time. This is precisely what gives an attacker the ability to observe the system and infer knowledge. An observer collecting information for long enough can then infer the execution pattern and rely on the real-time systems predictability for a directed attack. Schedule randomization was proposed [58] to defend real-time systems against side-channel attacks. During the execution of the system, as long as deadline constraints are not violated, the next task is picked randomly from the ready queue. The schedule is either generated online [27, 58], or selected from a set of pre-generated schedules [26, 27]. For embedded systems, the overhead of online generation can be avoided if it is possible to compute and store a schedule set with acceptable diversity [26].

Nasri et al. [37] analyze the conditions for successful time-domain attacks, concluding the difficulty of mounting such attacks in event-triggered systems. While it is true that some of these attacks can be difficult to mount for a generic system, the predictable schedule of time-triggered systems makes them more vulnerable. The attacks are simpler to carry on and can be more disruptive.

Two examples for state-of-the-art research deal with security for time-triggered communication. In [43], Skopik et al. introduce a security architecture for time-triggered communication which adds device authentication, secure clock synchronization and application level security. Wasicek et al. [54] investigate the security of time-triggered transmission channels and show how an authentication protocol secures these channels without violating timeliness properties. In our work, we do not consider intended communication channels for infering timing information, but instead focus on covert or side channels and the implication of attackers learning timing information to coordinate their attacks.

## 9 Conclusion

In this paper we analyzed vulnerabilities of time-triggered systems with respect to timing-inference based directed attacks, presented two mitigation strategies, and analyzed the randomness of schedules. The deterministic behaviour of time-triggered systems allows attackers to infer timing information over side channels and precisely target victim tasks. Worst case execution time assumptions, on which schedules are based, do not take malicious behaviour into account. As the schedule of a time-triggered system comprises only a few bytes, it can be inferred by an attacker over side-channels. In order to prevent attackers from predictions about the point in time when a certain task is executed, we presented two mitigation strategies for directed attacks. First, we introduced slot-level randomization, which impedes predictions about the schedule by selecting the next job at random. We employ concepts of slot shifting to allow randomization of a time-triggered schedule without violating timing constraints. Secondly, we proposed online selection of offline precomputed schedules for mitigation of directed attacks. At runtime, a schedule from a precomputed set of schedules is randomly selected at the end of each hyperperiod. We showed how to compute the minimum number of schedules needed to achieve maximum schedule diversity and devised an algorithm to find these schedules. First, we tested our algorithm with synthetic task sets and presented results regarding achievable entropy respecting varying hyperperiods and utilization levels. Then, we evaluated our mitigation strategies with respect to overhead and memory cost with a practical, real-world case study of a safety-critical flight controller. Slot-level randomization has a runtime overhead of around 3 percent of the slot size in the worst case, which makes it suitable for practical use. Scheduling precomputed schedules reduces the worst case runtime overhead to around 1 percent of the slot size, but is more costly in terms of memory. A single schedule for the case study has a size of 52 bytes, but the total number of feasible schedules lies in the magnitude of $10^{22}$. Out of this large amount of schedules, only 100 are needed to achieve the optimal upper-approximated entropy. Thus, both mitigation strategies proved to be practical. Attackers could still try to launch undirected attacks, but they will be easier to detect this way.

## References

1 Dakshi Agrawal, Bruce Archambeault, Josyula Rao, and Pankaj Rohatgi. The EM side-channel(s). In *4th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES, 2002. doi:10.1007/3-540-36400-5_4.

2 Amotz Bar-Noy, Danny Dolev, Cynthia Dwork, and H. Raymond Strong. Shifting Gears: Changing Algorithms on the Fly to Expedite Byzantine Agreement. *Inf. Comput.*, 97(2):205–233, 1992. doi:10.1016/0890-5401(92)90035-E.

3 Michael G. Bechtel and Heechul Yun. Denial-of-service attacks on shared cache in multicore: Analysis and prevention. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019. doi:10.1109/RTAS.2019.00037.

4 Enrico Bini and Giorgio Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2), 2005. doi:10.1007/s11241-005-0507-9.

5 Peter K. Boucher, Raymond K. Clark, Ira B. Greenberg, E. Douglas Jensen, and Douglas M. Wells. *Toward a Multilevel-Secure, Best-Effort Real-Time Scheduler*, pages 49–68. Springer Vienna, Vienna, 1995. doi:10.1007/978-3-7091-9396-9_8.

6 Luis Brandao and Alysson Bessani. On the Reliability and Availability of Systems Tolerant to Stealth Intrusion. In *5th Latin-American Symposium on Dependable Computing (LADC'11)*, Brazil, April 2011. doi:10.1109/LADC.2011.27.

7 Luis Brandao and Alysson Bessani. On the Reliability and Availability of Replicated and Rejuvenating Systems under Stealth Attacks and Intrusions. *Journal of the Brazilian Computer Society*, 18:61–80, March 2012. doi:10.1007/s13173-012-0062-x.

8 Xi Chen, Juejing Feng, Martin Hiller, and Vera Lauer. Application of software watchdog as a dependability software service for automotive safety relevant systems. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2007. doi:10.1109/DSN.2007.14.

9 Silviu S. Craciunas and Ramon Serna Oliver. SMT-based Task- and Network-level Static Schedule Generation for Time-Triggered Networked Systems. In *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*, RTNS '14, pages 45:45–45:54, New York, NY, USA, 2014. ACM. doi:10.1145/2659787.2659812.

10 Joanne Bechta Dugan and Randy Van Buren. Reliability evaluation of fly-by-wire computer systems. *Journal of Systems and Software*, 25(1):109–120, 1994. doi:10.1016/0164-1212(94)90061-2.

11 Christian Ferdinand and Reinhard Wilhelm. Efficient and precise cache behavior prediction for real-time systems. *Real-Time Systems*, 17(2):131–181, November 1999. doi:10.1023/A:1008186323068.

12 G. Fohler. Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. In *Proceedings 16th IEEE Real-Time Systems Symposium*, pages 152–161, December 1995. doi:10.1109/REAL.1995.495205.

13 Gerhard Fohler. *Advances in Real-Time Systems, Chapter Predictably Flexible Real-time Scheduling*. SPRINGER, 2012.

14 Alain Girault, Hamoudi Kalla, and Yves Sorel. An active replication scheme that tolerates failures in distributed embedded real-time systems. In *Design Methods and Applications for Distributed Embedded Systems*, pages 83–92. Springer, 2004. doi:10.1007/1-4020-8149-9_9.

15 Monowar Hasan, Sibin Mohan, Rodolfo Pellizzoni, and Rakesh B. Bobba. A design-space exploration for allocating security tasks in multicore real-time systems. In *Design, Automation & Test in Europe*, DATE, 2018. doi:10.23919/DATE.2018.8342007.

16 J.M. Hendrickx, K.H. Johansson, R.M. Jungers, H. Sandberg, and K.C. Sou. Efficient computations of a security index for false data attacks in power networks. *IEEE TAC*, 59(12):3194–3208, 2014. doi:10.1109/TAC.2014.2351625.

17 W. M. Hu. Lattice scheduling and covert channels. In *Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 52–61, May 1992. doi:10.1109/RISP.1992.213271.

18 B. K. Huynh, L. Ju, and A. Roychoudhury. Scope-aware data cache analysis for wcet estimation. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 203–212, April 2011. doi:10.1109/RTAS.2011.27.

19 Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems, 2010.

20 Rolf Isermann, Ralf Schwarz, and Stefan Stolzl. Fault-tolerant drive-by-wire systems. *IEEE Control Systems*, 22(5):64–81, 2002. doi:10.1109/MCS.2002.1035218.

21 Road vehicles – Functional safety, 2011.

22 P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre attacks: Exploiting speculative execution. In *2019*

*IEEE Symposium on Security and Privacy (SP)*, pages 1–19, 2019. doi:10.1109/SP.2019.00002.

23 Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. meltdownattack.com, 2018. URL: https://spectreattack.com/spectre.pdf.

24 H. Kopetz. Sparse time versus dense time in distributed real-time systems. In *[1992] Proceedings of the 12th International Conference on Distributed Computing Systems*, pages 460–467, June 1992. doi:10.1109/ICDCS.1992.235008.

25 Kristin Krüger, Gerhard Fohler, Marcus Völp, and Paulo Esteves-Verissimo. Improving security for time-triggered real-time systems with task replication. In *24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, August 2018. doi:10.1109/RTCSA.2018.00036.

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

27 Kristin Kruger, Marucs Völp, and Gerhard Fohler. Vulnerability analysis and mitigation of directed timing inference based attacks on time-triggered systems. In *Euromicro Conference on Real-Time Systems*, ECRTS, 2018. doi:10.4230/LIPIcs.ECRTS.2018.22.

28 Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. doi:10.1145/357172.357176.

29 J. Liedtke, H. Hartig, and M. Hohmuth. OS-controlled cache predictability for real-time systems. In *Proceedings Third IEEE Real-Time Technology and Applications Symposium*, pages 213–224, June 1997. doi:10.1109/RTTAS.1997.601360.

30 Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *ArXiv e-prints*, 2018. arXiv:1801.01207.

31 Songran Liu, Nan Guan, Dong Ji, Weichen Liu, Xue Liu, and Wang Yi. Leaking your engine speed by spectrum analysis of real-time scheduling sequences. *Journal of Systems Architecture*, 2019. doi:10.1016/j.sysarc.2019.01.004.

32 Sixing Lu, Minjun Seo, and Roman Lysecky. Timing-based anomaly detection in embedded systems. In *20th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2015. doi:10.1109/ASPDAC.2015.7059110.

33 Keith Marzullo. Tolerating failures of continuous-valued sensors. *ACM Transactions on Computer Systems*, 8(4):284–304, November 1990. doi:10.1145/128733.128735.

34 Edgar Mateos and Catherine Gebotys. A new correlation frequency analysis of the side channel. In *Workshop on Embedded Systems Security*, 2010. doi:10.1145/1873548.1873552.

35 Robert Mitchell and Ing-Ray Chen. A survey of intrusion detection techniques for cyber-physical systems. *ACM Computing Surveys (CSUR)*, 46(4), 2014. doi:10.1145/2542049.

36 Sibin Mohan, Man-Ki Yoon, Rodolfo Pellizzoni, and Rakesh B Bobba. Integrating security constraints into fixed priority real-time schedulers. *Real-Time Systems*, pages 1–31, 2016. doi:10.1007/s11241-016-9252-5.

37 Mitra Nasri, Thidapat Chantem, Gedare Bloom, and Ryan M. Gerdes. On the pitfalls and vulnerabilities of schedule randomization against schedule-based attacks. In Björn B. Brandenburg, editor, *25th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2019, Montreal, QC, Canada, April 16-18, 2019*, pages 103–116. IEEE, 2019. doi:10.1109/RTAS.2019.00017.

38 C. Pagetti, D. Saussié, R. Gratia, E. Noulard, and P. Siron. The ROSACE case study: From Simulink specification to multi/many-core execution. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 309–318, April 2014. doi:10.1109/RTAS.2014.6926012.

39 Dorottya Papp, Zhendong Ma, and Levente Buttyan. Embedded systems security: Threats, vulnerabilities, and attack taxonomy. In *13th Annual Conference on Privacy, Security and Trust*, PST, 2015. doi:10.1109/PST.2015.7232966.

40 Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching Agreement in the Presence of Faults. *J. ACM*, 27(2):228–234, 1980. doi:10.1145/322186.322188.

41 Thomas Popp, Stefan Mangard, and Elisabeth Oswald. Power analysis attacks and countermeasures. *IEEE Design & test of Computers*, 24(6), 2007. doi:10.1109/MDT.2007.200.

42 Stefan Schorr. *Adaptive Real-Time Scheduling and Resource Management on Multicore Architectures.* PhD thesis, Technical University of Kaiserslautern, March 2015. URL: https://kluedo.ub.uni-kl.de/frontdoor/index/index/docId/4008.

43 Florian Skopik, Albert Treytl, Arjan Geven, Bernd Hirschler, Thomas Bleier, Andreas Eckel, Christian El-Salloum, and Armin Wasicek. *Towards Secure Time-Triggered Systems*, pages 365–372. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-33675-1_33.

44 Joon Son and Jim Alves-Foss. Covert timing channel capacity of rate monotonic real-time scheduling algorithm in MLS systems. In *Communication, Network, and Information Security*, 2006. doi:10.1109/IAW.2006.1652117.

45 P. Sousa, N. F. Neves, and P. Verissimo. Proactive resilience through architectural hybridization. In *ACM Symposium on Applied Computing*, pages 686–690, 2006. doi:10.1145/1141277.1141435.

46 Paulo Sousa, Alysson Bessani, Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo. Highly Available Intrusion-Tolerant Services with Proactive-Reactive Recovery. *IEEE Transactions on Parallel and Distributed Systems, vol. 21, no. 4, pp. 452-465, Apr. 2010.*, 2010. URL: http://www.navigators.di.fc.ul.pt/archive/papers/ieeetpds-prrw-final-version.pdf.

47 Raphael Spreitzer, Veelasha Moonsamy, Thomas Korak, and Stefan Mangard. Systematic classification of side-channel attacks: A case study for

mobile devices. *IEEE Communications Surveys and Tutorials*, 20(1), 2018. doi:10.1109/COMST.2017.2779824.

48 A. Teixeira, I. Shames, H. Sandberg, and K.H. Johansson. Revealing stealthy attacks in control systems. In *Allerton Conference on Communication, Control, and Computing*, pages 1806–1813, 2012. doi:10.1109/Allerton.2012.6483441.

49 A. Teixeira, I. Shames, H. Sandberg, and K.H. Johansson. Distributed fault detection and isolation resilient to network model uncertainties. *IEEE Transactions on Cybernetics*, 44(11):2024–2037, November 2014. doi:10.1109/TCYB.2014.2350335.

50 Mankuan Vai, Roger Khazan, Daniil Utin, Sean O'Melia, David Whelihan, and Benjamin Nahill. Secure embedded systems. Technical report, MIT Lincoln Laboratory Lexington United States, 2016.

51 M. Völp, B. Engel, C. J. Hamann, and H. Härtig. On confidentiality-preserving real-time locking protocols. In *IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2013. doi:10.1109/RTAS.2013.6531088.

52 Marcus Völp, Claude-Joachim Hamann, and Hermann Härtig. Avoiding timing channels in fixed-priority schedulers. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*, ASIACCS '08, pages 44–55, New York, NY, USA, 2008. ACM. doi:10.1145/1368310.1368320.

53 Nils Vreman, Richard Pates, Kristin Krüger, Gerhard Fohler, and Martina Maggio. Minimizing side-channel attack vulnerability via schedule randomization. In *58th IEEE Conference on Decision and Control (CDC)*, December 2019. doi:10.1109/CDC40024.2019.9030144.

54 A. Wasicek, C. El-Salloum, and H. Kopetz. Authentication in time-triggered systems using time-delayed release of keys. In *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 31–39, March 2011. doi:10.1109/ISORC.2011.14.

55 Armin Rudolf Wasicek. *Security in Time-Triggered Systems*. PhD thesis, Technische Universität Wien, 2011.

56 C. B. Watkins and R. Walter. Transitioning from federated avionics architectures to Integrated Modular Avionics. In *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*, pages 2.A.1–1–2.A.1–10, October 2007. doi:10.1109/DASC.2007.4391842.

57 Steve H. Weingart. Physical security devices for computer subsystems: A survey of attacks and defenses. In *Cryptographic Hardware and Embedded Systems*, CHES, 2000. doi:10.1007/3-540-44499-8_24.

58 Man-Ki Yoon, Sibin Mohan, Chien-Ying Chen, and Lui Sha. TaskShuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS, 2016. doi:10.1109/RTAS.2016.7461362.

59 H. Yun, R. Mancuso, Z. P. Wu, and R. Pellizzoni. PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 155–166, April 2014. doi:10.1109/RTAS.2014.6925999.

60 Christopher Zimmer, Balasubramanya Bhat, Frank Mueller, and Sibin Mohan. Time-based intrusion detection in cyber-physical systems. In *1st ACM/IEEE International Conference on Cyber-Physical Systems*, 2010. doi:10.1145/1795194.1795210.

# We know what you're doing! Application detection using thermal data

**Philipp Miedl**
Computer Engineering and Networks Laboratory, ETH Zurich,
Gloriastrasse 35, Zurich, Switzerland

**Rehan Ahmed**
Information Technology University of the Punjab,
Arfa Software Technology Park, Ferozpur Road, Lahore, Pakistan

**Lothar Thiele**
Computer Engineering and Networks Laboratory, ETH Zurich,
Gloriastrasse 35, Zurich, Switzerland

## Abstract

Modern mobile and embedded devices have high computing power which allows them to be used for multiple purposes. Therefore, applications with low security restrictions may execute on the same device as applications handling highly sensitive information. In such a setup, a security risk occurs if it is possible that an application uses system characteristics to gather information about another application on the same device.

In this work, we present a method to leak sensitive runtime information by just using temperature sensor readings of a mobile device. We employ a Convolutional-Neural-Network, Long Short-Term Memory units and subsequent label sequence processing to identify the sequence of executed applications over time. To test our hypothesis we collect data from two state-of-the-art smartphones and real user usage patterns. We show an extensive evaluation using laboratory data, where we achieve labelling accuracies of up to 90% and negligible timing error. Based on our analysis we state that the thermal information can be used to compromise sensitive user data and increase the vulnerability of mobile devices. A study based on data collected outside of the laboratory opens up various future directions for research.

**2012 ACM Subject Classification** Security and privacy → Embedded systems security; Security and privacy → Hardware attacks and countermeasures; Security and privacy → Mobile platform security; Security and privacy → Virtualization and security; Hardware → Temperature monitoring

**Keywords and Phrases** Thermal Monitoring, Side Channel, Data Leak, Sequence Labelling

**Digital Object Identifier** 10.4230/LITES.7.1.2

**Supplementary Material** All data presented in this work, including the raw data collected from the smartphones, are available online under following links:
- `https://doi.org/10.3929/ethz-b-000418184`
- `https://doi.org/10.3929/ethz-b-000418183`

Experiment Orchestration Toolkit is available via the Experiment Orchestration Toolkit (ExOT)-project-website `exot.ethz.ch`, including the extension made in this work [28]. The specific tools used, as well as the code and configuration developed for this publications, can be found under the following links:
- `https://gitlab.ethz.ch/tec/public/exot/eengine/-/tree/pub_MAT20`
- `https://gitlab.ethz.ch/tec/public/exot/app_apk/-/tree/pub_MAT20`
- `https://gitlab.ethz.ch/tec/public/exot/app_lib/-/tree/pub_MAT20`
- `https://gitlab.ethz.ch/tec/public/exot/compilation/-/tree/pub_MAT20`

Before executing any experiments, please run the `RUNE.sh` script from the `eengine/data` directory, to reshape the data into the fitting form for this version of ExOT.

**Acknowledgements** Thanks to Xiaoxi He and Matthias Meyer for their valuable input and the discussion concerning the machine learning part of the work. Many thanks to Balz Maag for recording all the user inputs and the verification traces with both mobile phones, and Lukas Sigrist for providing access to the thermal-testbed. Thanks also to our former students Raphael, Max, Manuel and Athanasios for providing useful insights into thermal sequence data processing during their semester projects.

**Received** 2020-03-31 **Accepted** 2021-03-26 **Published** 2021-08-12

**Editor** Alan Burns and Steve Goddard

**Special Issue** Special Issue on Embedded System Security

## 1 Introduction

Due to their high computational power, mobile devices, such as smartphones and tablets, are often used for multiple purposes concurrently. Consequently, applications with different levels of security and privacy clearances reside on the same piece of equipment. For example, companies may allow their employees to use the same smartphone for business and private applications, or people with medical conditions may use their smartphone to monitor their health status. In both cases, applications performing highly-critical tasks operate beside benign applications, like games. To prevent security and privacy violations due to different applications on the same device, Operating Systems (OSs) often rely on the security paradigm of *application isolation and permission separation*. This paradigm defines that security and privacy are ensured if information transfer between applications is only possible under the oversight of the OS.

Previous work has shown that the security framework of application isolation and permission separation is susceptible to data leaks based on shared resources, such as caches [42]. While shared resources can be used to compromise the system, they can aim at increasing the efficiency of a system or providing monitored means of communication between applications. So even though shared resources might pose a security threat, we do not consider their presence in a system as a breach of the security paradigm of application isolation and permission separation.

An example for an implementation of application isolation and permission separation is Android application sandboxing[1]. Android sandboxing isolates applications and only allows applications to communicate through explicitly permitted channels. For example, applications can communicate using intents, which are subject to the scrutiny of the OS as all intents are routed through the Android system. While previous work has shown that intents can be misused to create data leaks [11], we argue that the principal design of intents does not violate the security paradigm of application isolation and permission separation.

In this work, we show how to bypass the security paradigm of application isolation and permission separation and compromise a system by providing a method that allows an adversary to determine which applications are executed on the device at specific time intervals.

The execution of applications influences the power consumption of the device and its temperature. Therefore, temperature readings may contain information on the executed application. Modern Systems-on-Chips (SoCs) typically have multiple thermal sensors to support smart power management. For example, Arm big.LITTLE SoCs often feature thermal sensor readings in both core clusters and several more for other components of the SoC. In general, these thermal measurements are exposed to the OS through an unrestricted software interface, which makes thermal data easily accessible. For instance, there are several applications on the Google Play

[1] `https://source.android.com/security/app-sandbox`

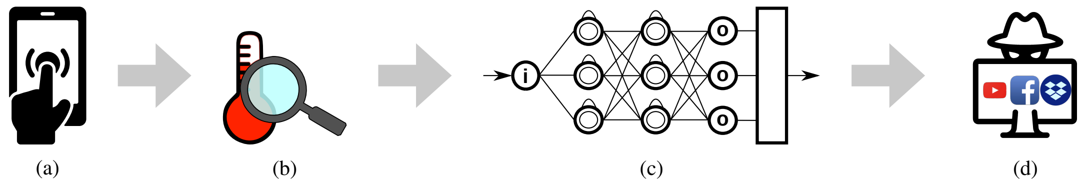


**Figure 1** (a) Different applications are executed on a mobile device depending on the user input. (b) Thermal information provided by the Operating System is collected by a third-party application. (c) Analysis of the thermal data to determine the application sequence. (d) The application sequence is used to create a usage profile or detect other applications, causing security and privacy violations.

Store that allow the thermal information of an Android device to be read, without the need for elevated privileges[2]. It has already been shown that these thermal readings may lead to security issues [6]. In particular, the thermal covert channel presented in by Bartolini et al. [1] shows the possibility of a data leak, as Ristenpart et al. [35] already stated that "Covert channels provide evidence that exploitable side channels may exist". However, as thermal information is vital to power management, it remains accessible without further permission requirements.

In this work, we present a side channel attack based on thermal sensor readings of a mobile device, as depicted in Figure 1. Different applications are executed on a mobile device based on the user input (a). These applications are not allowed to share any information without the supervision of the OS, due to the security paradigm of application isolation and permission separation. However, applications are allowed to read thermal information from the OS software interface. An adversary may deploy an application to read the thermal information (b) and analyses the thermal data (c). This allows the adversary to determine the execution sequence of other applications or detect the set of running applications (d). As this establishes an information transfer between intentionally isolated applications, such a thermal side channel violates the security and privacy restrictions imposed by the OS.

**Contributions.** Our main contributions in this work are:

1. We present a side channel attack that uses thermal data collected from mobile devices to determine patterns of application usage.
2. To the best of our knowledge, we are the first to apply machine learning techniques from time-series processing domain to determine an application execution sequence.
3. We present an extensive experimental evaluation of the thermal side channel based on real user interactions with the device, laboratory and real-world data.

## 2 Related work

Mobile devices are now present in almost all aspects of our daily life and have increased computing power. While usage information can be used to improve the functionality of mobile devices [5], this information can also be misused for malicious purposes [39]. This leads to security and privacy issues, which are well defined and broadly studied problems in current computing systems. Lampson [19] was the first to highlight the problem of *covert* and *side channels*, when he defined the *confinement problem*. The confinement problem states that an application is secure as long as it is ensured, that no information can be leaked to a third party. However, data leaks in form of

[2] e. g., Simple System Monitor (`https://play.google.com/store/apps/details?id=com.dp.sysmonitor.app`)

covert or side channels are almost omnipresent in modern computing systems, which makes their analysis even more important. In this work, we analyse a data leak, or side channel, based on the availability of temperature data.

## 2.1 Architectural data leaks

In the last years, many data leaks have been discovered and analysed, often based on architectural features of modern multicore systems. For example, the well known Spectre [18] and Meltdown [21] data leaks take advantage of the hyper-threading in Intel processors. Similarly, Rong et al. [36] showed how to compromise cloud systems by taking advantage of the so called Cloud Covert Channel based on Memory Deduplication (CCCMD). This channel has further been improved [24, 33] and specifically targeted at Intel SGX based systems [10]. Furthermore, a variety of other cache channels exploit the fact that multiple cores share the same cache [20]. However, also branch predictors [8] or random number generators [7] have been exploited as possible sources for data leaks in modern computing systems. Our work will not take advantage of architectural issues to establish a data leak, but exploit a data leak based on the thermal state of a mobile device.

## 2.2 Temperature related data leaks

Murdoch [31] showed that it is possible to identify servers in the tor network by analysing the temperature induced clock skew of the machines. The server was heated by causing a high load and was identified by concurrently analysing the timestamps of response packets of the server. This temperature induced timing side channel was further analysed and improved by Zander and Murdoch [44]. The authors minimized the clock jitter in the response packets and derived a channel capacity of approximately 20.5 bits per hour [43]. Furthermore, Ristenpart et al. [35] showed that this technique can also be applied to find other Virtual Machines (VMs) running on the same infrastructure. Other temperature related data leaks have been shown, which rely on the effects of the power management system of the devices. This includes fan speed [2], power consumption [26, 30] or operating frequency [27]. Similar to these work, we will also take advantage of the fact that modern computing devices have varying power needs and temperature profiles for different computing tasks.

Data leaks might also occur if chips are not operated in the right temperature range, as demonstrated by Hutter and Schmidt [15] on the RSA implementation on an AVR microcontroller. The authors showed that the hamming weight of the key can be leaked by correlating temperature, power and execution time, if the chip is operated beyond its specified temperature range. In this work, we will also use temperature as a medium to leak information, without the knowledge of the OS. However, we will not tamper with the environmental conditions and only rely on temperature measurements provided by the device.

## 2.3 Thermal data leaks and attacks

In addition to temperature related effects, temperature itself can serve as a medium to leak data. Islam et al. [17] presented a thermal side channel attack that allows an attacker to time power attacks on data centres more effectively. Thermal covert channels have also been extensively studied on FPGAs, where on-chip heat generators were used to transfer information out of the secure zone of the chip [22, 3, 16] or over time to the next scheduled application [40]. It has also been shown that general purpose CPUs are vulnerable to temperature based covert channels [23, 1]. Guri et al. [13] presented a method to establish such a thermal covert channel between air-gapped systems.

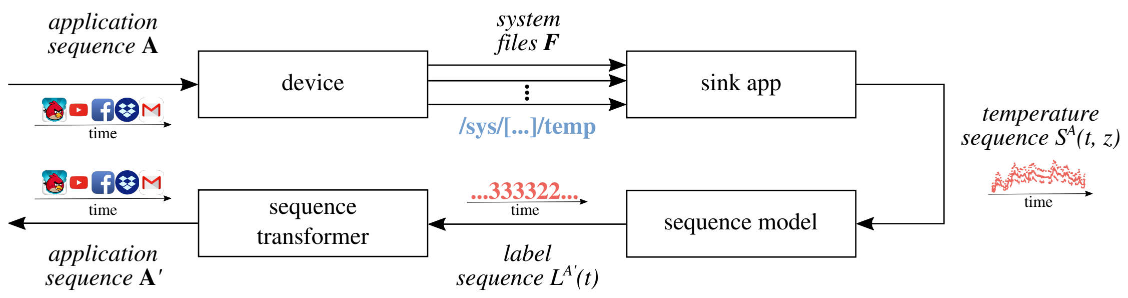


**Figure 2** The information flow during a thermal side channel attack. A user executes a sequence of different applications $\mathbf{A}$ on the device. The sink application monitors the resulting heat generation from the device by reading the respective system files $\mathbf{F}$ for the different thermal zones $z \in \mathbf{Z}$. The sink application outputs the thermal sequence $S^A(t, z)$, which is fed to the sequence model. This model generates the label sequence $L^{A'}(t)$, holding one application label per time-step. $L^{A'}(t)$ is fed to the sequence transformer, which then outputs the inferred application sequence $\mathbf{A}'$.

### 2.4 Machine learning in security applications

Fuelled by the rapid advances in the machine learning domain, techniques from the machine learning domain are more frequently applied to device security and privacy relevant topics. Machine learning methods have been employed to build detectors for malicious applications. For example, the detectors analyse the Application Program Interface (API) calls and correlate it with the permission set of applications [32]. Buczak and Guven [4] gathered many other examples for machine learning applications in the domain of cyber security, classifying all these methods in three groups:

**(i)** misuse based approaches that identify known attacks by their signature,
**(ii)** anomaly based approaches which recognise behaviour which is not considered "normal", and
**(iii)** hybrid systems, which are a mix of misuse and anomaly based approaches.

All of these techniques rely on the availability of large amount of data for training.

In our work, we will also apply machine learning techniques in the security domain. While most of the existing works take advantage of machine learning techniques for defence mechanisms, we will use machine learning from the perspective of an adversary. Furthermore, we will show how to generate a large amount of data for training, without having to deploy a complex measurement system for a long period of time.

## 3 Threat model

First, we describe the attack scenario to define the threat model of the thermal side channel attack. Second, we present the attack concept to outline the techniques used to mount a thermal side channel attack.

### 3.1 Attack scenario

We base our threat model on the scenario presented in Figure 1 and the information flow in Figure 2. A user runs a sequence of applications $\mathbf{A}$ on a mobile device. This sequence consists of an arbitrary sequential order and duration of application executions, depending on the user needs. Due to the difference in computational effort and the components utilised by different applications, the device generates different heat patterns in time and space. These heat patterns can be determined by observing the different thermal zones of a device. A thermal zone defines sensor readings for a specific part of a device, for example, a processor core. An adversary infiltrates the mobile device, for example, by disguising a thermal monitoring *sink* background service as part of a benign game. Therefore, the adversary is able to monitor the temperature sensors of the mobile device.

The sink in Figure 2 collects a temperature sequence $S^A(t, z)$, which is composed of the thermal readings of all observed thermal zones $z \in \mathbf{Z}$. The adversary analyses the temperature sequence $S^A(t, z)$ using a sequence model. As a result of the analysis, the adversary obtains an application label sequence $L^{A'}(t)$ with one label per time-step. In the final analysis step, the per-time-step label sequence $L^{A'}(t)$ is transformed into a condensed application label sequence $\mathbf{A}'$. This transformation is necessary to eliminate duplicate labels and artefacts in the per-time-step label sequence $L^{A'}(t)$. Finally, the output application sequence $\mathbf{A}'$ allows an adversary to derive further insights into the user behaviour with different security and privacy implications. These implications depend on whether the adversary does the analysis *offline* or *online*.

**Offline Scenario.** In this scenario, the adversary only deploys the sink application on the attacked device. The sequence model, as well as the sequence transformer, are implemented on a dedicated analysis device. Therefore, the sink application transfers the thermal data to the analysis device for application sequence inference. This way, the attacker can determine which applications were executed at what time and create a detailed user profile containing sensitive information. For example, the usage of medical applications would allow inferences on the medical condition of the user, or location-based applications, e. g., a regional tourism application, allow inferences about the location and activities of the user. Such a data leak would present a major privacy violation, as the profiling of the user behaviour could be performed without the user's knowledge or consent.

**Online Scenario.** In the online attack scenario, the attacker performs the application sequence inference on the attacked device in real-time. This means, in addition to the sink applications, the sequence model and the sequence transformer also have to be deployed on the attacked device. The attacker may then use the real-time information to time a targeted attack on a specific application in the secure domain. Such an attack is dangerous when considering a company that enforces the *bring-your-own-device* policy. According to this policy, employees will use their mobile devices for private and business applications. Therefore, the phone features two application domains, i. e., business and private, which are separated by virtualisation. An example of a virtualisation environment is "Android for work". Such a system is vulnerable if an attacker can retrieve information about applications in the secure domain. This can be achieved if an attacker mounts an online thermal side channel attack in the less secure domain to gain runtime information from the secure domain.

## 3.2 Concept of a thermal side channel attack

To mount the thermal side channel attack, the sink application, the sequence model and the sequence transformer need to be implemented. We base the sink application design on ExOT, presented by Miedl et al. [29], which allows us to sample the thermal zones of a device in a timed fashion and without the need for elevated privileges.

The thermal side channels establish a highly complex transformation from device usage patterns to observable temperature changes in the various thermal zones $z \in \mathbf{Z}$. In addition, the usage patterns of applications differ vastly, as do the interactions of applications with their users. Masti et al. [23] already stated that correlation based methods show very limited capabilities as thermal features identifying an application are very subtle. Therefore, we use techniques from the machine learning domains of sequence-to-sequence labelling and time-series modelling to implement the sequence model. We build our sequence model using a Convolutional-Neural-Network (CNN) and a Recurrent Neural Network (RNN), which we describe in detail in section 5. The sequence transformer is based on classical filtering algorithms and rules of condensing labels, as outlined in section 6.

It is well known that CNNs and RNNs require a tremendous amount of labelled training data to perform well. In our case, suitable data should be available to represent user interactions and to represent thermal traces from different applications on different devices and in different thermal environments. However, there is no appropriate dataset available, nor is it feasible to deploy a long-term measurement setup to gather sufficiently diverse data (user interactions, applications, devices, thermal environments, interferences) and label them correctly. Therefore, we must define a data augmentation scheme that allows us to generate a highly representative and diverse data set.

## 4 Data augmentation

The overall process to generate a large set of diverse thermal sequences is depicted in Figure 3. It contains four distinct components that will be detailed in the subsequent sections. Input to the data augmentation scheme are

**(i)** the device for which the thermal sequence needs to be generated,

**(ii)** the set of applications that will potentially run and corresponding user inputs, and

**(iii)** a dataset configuration, which contains information to configure the whole generation scheme.

The device characterisation uses measurements in order to model the thermal behaviour of the device itself. The outcomes are temperature coefficients, namely, the internal thermal, ambient heating and ambient cooling coefficients, which are used to concatenate the thermal profiles of applications and augment the data by modelling changes in the ambient temperature of the device. Note that these coefficients are determined for each temperature sensor (or zone) $z$ individually.

The purpose of the application characterisation is to record the temperature trace of a running application. This is characteristic for the application and can be used by an adversary application to spy on it. Again, the traces are collected for each temperature zone individually.

The sequence parameter generation determines (random) sequences of applications whose corresponding temperature sequences are generated by the thermal sequence generation. In order to increase the diversity of training data, it also (randomly) generates traces of ambient temperature offsets in order to model that the device is exposed to dynamically changing external temperatures.

The thermal sequence generation combines all of this information to generate a temperature sequence and its associated label sequence.

We base our thermal modelling on *Newton's law of cooling*. It states that the rate of heat loss of a body is directly proportional to the difference in the temperatures between the body and its surroundings. Therefore, it is expected that the system will experience exponential decay in the temperature difference of body and surroundings as a function of time. Note that Newton's law does not take heat transfer between individual architectural elements into account. However, as the experimental results show, this approximation is sufficiently accurate for our purpose.

The basic form of Newton's law is

$$T(t_2) = T^{\text{idle}} + (T(t_1) - T^{\text{idle}}) \cdot e^{-\beta(t_2 - t_1)} \tag{1}$$

where $\beta$ denotes the thermal coefficient, $T(t)$ denotes the temperature at time $t$, we have $t_2 \geq t_1$, and $T^{\text{idle}}$ denotes the steady-state temperature. For convenience, we introduce the temperature

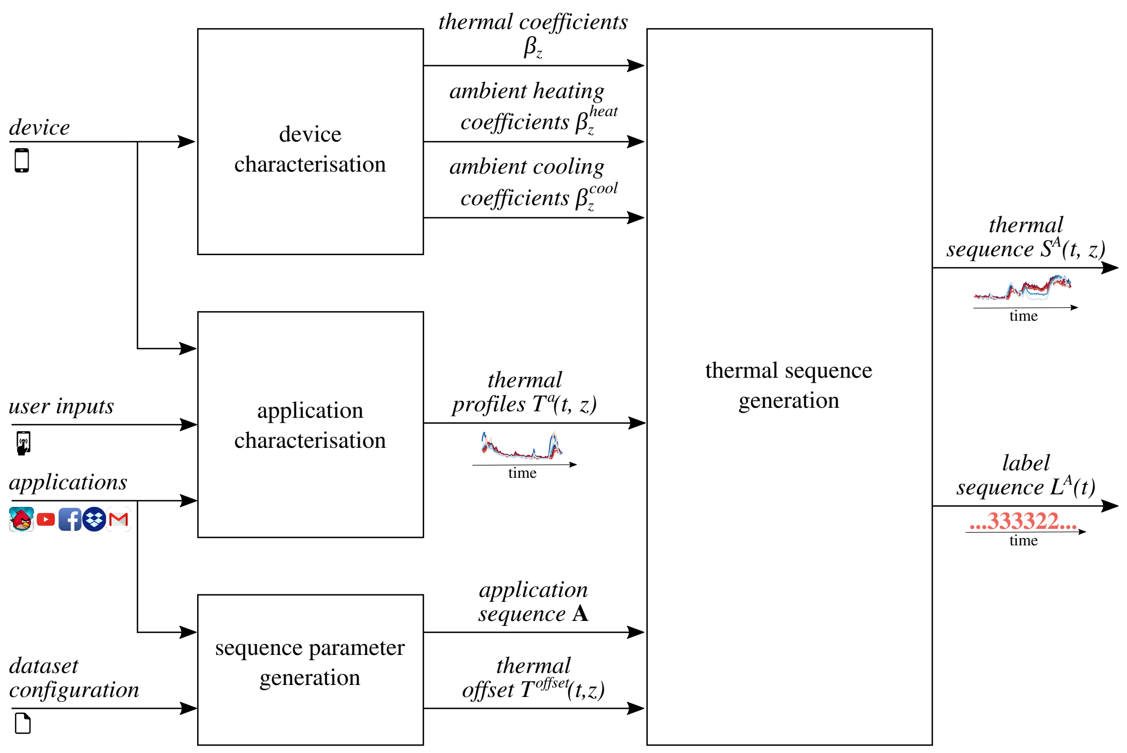

device
device characterisation
thermal coefficients $\beta_z$
ambient heating coefficients $\beta_z^{heat}$
ambient cooling coefficients $\beta_z^{cool}$
user inputs
applications
application characterisation
thermal profiles $T^a(t, z)$
time
dataset configuration
sequence parameter generation
application sequence **A**
thermal offset $T^{offset}(t,z)$
thermal sequence generation
thermal sequence $S^A(t, z)$
label sequence $L^A(t)$
...333322...

**Figure 3** Overall data augmentation scheme structure. We generate a thermal sequence and its labels based on (i) the particular device and its measured thermal characterisation, (ii) the set of identifiable applications and their thermal characterisations, and (iii) the highly-configurable data set configuration. Using diverse configurations, we generate representative training data.

function $C(\cdot)$ with

$$\Delta T(t_0 + t) = C(\Delta T(t_0), \beta, t) = \Delta T(t_0) \cdot e^{-\beta \cdot t} \tag{2}$$

It returns the temperature difference to the steady-state temperature $\Delta T(t_0 + \Delta t) = T(t_0 + \Delta t) - T^{\mathrm{idle}}$ after time $t$ and uses the initial temperature difference $\Delta T(t_0) = T(t_0) - T^{\mathrm{idle}}$, the thermal coefficient $\beta$, and the time difference $t$. In order to be able to consider different heat transfer mechanisms that mediate between heat losses and temperature differences, we can use different constant thermal coefficients in the temperature function $C(\cdot)$.

## 4.1 Device characterisation

One basic component of the data augmentation scheme is the characterisation of each device in terms of its thermal coefficients. This model will be used later on to combine the temperature profiles of two applications that run in sequence.

In order to consider different heat transfer mechanisms, we will characterise each thermal zone $z \in \mathbf{Z}$ of a given device by three thermal coefficients:

1. the internal temperature coefficient $\beta_z$,
2. the ambient heating coefficient $\beta_z^{\mathrm{heat}}$, and
3. the ambient cooling coefficient $\beta_z^{\mathrm{cool}}$.

The internal temperature coefficient $\beta_z$ describes how long the heating effect from a previously-executed application continues. To approximate this coefficient, we conduct measurements on the respective device. The device is placed in an environment with the controlled ambient temperature and kept idle before the start of the measurement. Next, a benchmark application is executed for some randomly chosen time, which increases the temperatures of the zones by some value. As soon as the application is stopped, a temperature trace is recorded. The measurement is repeated for different run-times of the benchmark. From these traces, we derive the internal temperature coefficient $\beta_z$ for each thermal zone $z$ using regression analysis of the temperature traces, i. e., fitting the temperature function $C(\cdot)$ in Equation (2) to the temperature measurements using non-linear least squares.

The two ambient coefficients for heating and cooling, $\beta_z^{heat}$ and $\beta_z^{cool}$, respectively, model the influence of increasing or decreasing ambient temperatures on a specific thermal zone. Both coefficients are determined by first moving the idle device to an environment with a different ambient temperature, then moving it back to the initial environment and taking the corresponding temperature trace. By isolating the time intervals when the device adapts to the new ambient temperature, similar to $\beta_z$, we use regression analysis to determine $\beta_z^{\text{heat}}$ and $\beta_z^{\text{cool}}$.

## 4.2 Application characterisation

We characterise an application $a$ running on a chosen device by determining the *thermal profile* $T^a(\Delta t, z)$, which describes the thermal behaviour caused by the execution of the application $a$. Here, $\Delta t$ denotes the time since the application start and $z$ denotes the observed thermal zone of the device.

To derive the thermal profile of an application, we record a thermal trace $T^a(t, z)$ during the execution of the application $a$, i. e., $0 \leq t \leq t_{\text{exec}}$. We do this measurement in the same environment as before for the internal thermal coefficients $\beta_z$. This way, we can ensure that the device has settled before starting the measurement and minimise the chance for interference from external factors.

For every application, we perform multiple measurements to derive multiple thermal profiles. This allows us to acquire a more diverse set of profiles and compensate for thermal variations caused by the measurement setup. In addition, we also record multiple user interaction patterns for each application, encapsulating typical application use cases. The user input for the different use cases of the applications is recorded and replayed, to ensure reproducibility of the raw data.

## 4.3 Sequence parameter generation

The component for sequence parameter generation provides a sequence of applications and a trace of changing ambient temperature offsets. Depending on these inputs, the corresponding thermal sequence is generated. By changing the sequence of applications and the ambient temperature offset of the device in an appropriate way, a large set of diverse thermal sequences and their associated labels can be generated. The strategy by which the sequence parameters are generated is described in the dataset configuration. In the following, we describe the functionality of the sequence parameter generation.

Based on the available applications $a$ and the dataset configuration, we generate the application sequence as an ordered list of tuples $\mathbf{A}$. The $i^{th}$ tuple of $\mathbf{A}$ is defined as $< a_i, t_i^{\text{start}}, t_i^{\text{end}} >$, where $a_i$ defines the $i^{\text{th}}$ application which is started at time $t_i^{\text{start}}$ and closed at time $t_i^{\text{end}}$. We do not consider the concurrent execution of two or more applications. Therefore, the execution intervals of individual applications are assumed to be disjoint and consecutive: $t_{i+1}^{\text{start}} = t_i^{\text{end}}$ and $t_i^{\text{end}} > t_i^{\text{start}}$. This assumption is realistic, considering the typical usage of a mobile device where a single foreground application is executed at a time. Nevertheless, one should also mention that there are also other usage patterns where music can play in the background, or two applications can be used side-by side in split-window mode.

Depending on the information in the dataset configuration, we generate the application sequence **A** either

**(A)** randomly,

**(B)** systematically to increase the number of different thermal profile sequences in the data set, or

**(C)** such that thermal profiles are randomly chosen from two sets of profiles, alternating.

Method (B) ensures that the maximum number of different thermal profiles appear in the data set. Method (C) is used, for example, if the thermal profiles can be split into two groups, i. e., thermal profiles of known and unknown applications.

The thermal offset $T^{\text{offset}}(t,z)$ simulates different environmental temperatures for different locations of the mobile device, e. g., indoors or outdoors. To generate a relative offset trace $T^{\text{offset}}(t,z)$ for each temperature $z$, the sequence parameter generation first randomly generates an initial thermal ambient thermal offset $\Delta T_0^{\text{ambient}}$ and an ambient thermal offset sequence $\mathbf{\Delta T^{ambient}}$ with $n$ tuples. The $i^{th}$ tuple of the sequence is defined as $< \Delta T_i^{\text{ambient}}, t_i^{\text{enter}} >$ and specifies that the device enters an environment with the ambient thermal offset of $T_i^{\text{ambient}}$ at time $t_i^{\text{enter}}$. The ambient thermal offset is relative to the ambient temperature in the measurement environment used for the application characterisation (see above). Based on this ambient thermal offset sequence, we derive $T^{\text{offset}}(t,z)$ using Newton's law as follows:

$$T^{\text{offset}}(t,z) = \begin{cases} \mathcal{U}(-1,1) + \Delta T_0^{\text{ambient}} \qquad \forall\, 0 \le t < t_1^{\text{enter}} \\ \mathcal{U}(-1,1) + \Delta T_i^{\text{ambient}} + C\left(T^{\text{offset}}(t_i^{\text{enter}},z) - T_i^{\text{ambient}}, \beta_z^{\text{heat}}, t - t_i^{\text{enter}}\right) \quad \cdots \\ \qquad \forall\, 1 \le i \le n \,\wedge\, t_i^{\text{enter}} < t \le t_{i+1}^{\text{enter}} \,\wedge\, T_{i-1}^{\text{ambient}} \le T_i^{\text{ambient}} \\ \mathcal{U}(-1,1) + \Delta T_i^{\text{ambient}} + C\left(T^{\text{offset}}(t_i^{\text{enter}},z) - T_i^{\text{ambient}}, \beta_z^{\text{cool}}, t - t_i^{\text{enter}}\right) \quad \cdots \\ \qquad \forall\, 1 \le i \le n \,\wedge\, t_i^{\text{enter}} < t \le t_{i+1}^{\text{enter}} \,\wedge\, T_{i-1}^{\text{ambient}} > T_i^{\text{ambient}} \end{cases} \tag{3}$$

where $t_{n+1}^{\text{enter}} = \infty$. Note that when the ambient temperature increases, we use $\beta_z^{\text{heat}}$ as the thermal parameter for the thermal behaviour model of the zone, while for decreasing ambient temperatures, we employ $\beta_z^{\text{cool}}$. $T^{\text{offset}}\,(t,z)$ not only contains the ambient offset of the thermal trace but also adds random thermal noise $\mathcal{U}(-1,1)$, where $-1$ and $1$ define the maximum amplitude of the thermal noise in °C. This is necessary, as traces generated from laboratory data is less noisy than real-world recordings.

## 4.4 Thermal sequence generation

Now that we have chosen the sequence parameters and have characterised the device, as well as the applications, we are in the position to generate a corresponding *thermal sequence*.

The first challenge is to generate application sequences based on the provided temperature profiles. The simple concatenation of these profiles following the provided application sequence **A** is not possible for three reasons: First, the final temperature of a temperature profile does not typically match the initial temperature of the subsequent application. Second, we cannot expect that an application runs from start to end. Rather, it undergoes a halting or closing phase when a context switch takes place. Finally, we must consider the changing ambient temperature offset, as well as the initial temperature offset of the thermal sequence. We will now go through these challenges one-by-one.

Before we can concatenate the thermal profiles according to the generated application sequence **A**, we have to select and crop thermal profiles to the desired length. However, we must include the thermal information of closing or halting the application in the cropped thermal profile. To this end, we empirically evaluate the time interval at the end of a thermal profile necessary to close an application. For example, a thermal profile has the length of 20 s, and we have evaluated

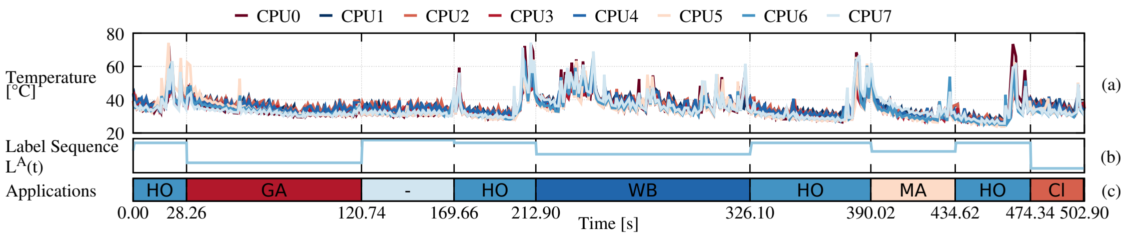


**Figure 4** Thermal sequence build from different thermal profiles collected from a Sony Xperia Z5, see Equation (6). (a) illustrates the thermal traces, (b) the label sequence $L^A(t)$ and (c) the application trace **A**. The application labels correspond to Table 1. Our data augmentation removes temperature discontinuities at application pre-emption points without distorting the thermal profiles.

that the last 4 s are used to close the application. If we now want to crop the thermal profile to a length of 12 s, we crop it to 8 s and then append the final 4 s. To ensure that there are no temperature discontinuities in the cropped thermal profile, we employ the temperature function $C(\cdot)$ as follows:

$$\overline{T}^a(t,z) = \begin{cases} T^a(t,z) & \forall\, t \le t_{crop} - t_{close} \\ T^a(t + t_{exec} - t_{crop}, z) + & \cdots \\ \quad C\left(T^a(t_{crop} - t_{close}, z) - T^a(t_{exec} - t_{close}), \beta_z, t + t_{close} - t_{crop}\right) \\ \qquad \forall\, t_{crop} - t_{close} < t \le t_{crop} \end{cases} \tag{4}$$

Here, $t_{crop}$ is the cropped length the thermal profile, i. e., the length as requested from the application sequence, $t_{close}$ is the time interval needed for closing an application, and $t_{exec}$ the total length of the thermal profile. We note that $t_{close} < t_{crop} < t_{exec}$.

Now, we will determine the thermal sequence $S^A(t,z)$ of an application sequence **A** with tuples $< a_i, t_i^{\text{start}}, t_i^{\text{end}} >$ for $1 \le i \le n_A$, while

**(i)** considering that there are no discontinues in the sequence when switching from one application to the next,
**(ii)** taking into account that the ambient temperature is changing, and
**(iii)** setting the initial temperature of the thermal sequence.

We obtain the thermal sequence $S^A(t,z)$ for each temperature zone $z$ as follows:

$$S^A(t,z) = \begin{cases} T^{\text{offset}}(t,z) + \overline{T}_1^a(t,z) & \forall\, 0 = t_1^{\text{start}} \le t \le t_1^{\text{end}} \\ T^{\text{offset}}(t,z) + \overline{T}_i^a(t,z) + C\left(S^A(t_{i-1}^{\text{end}}, z) - \overline{T}_i^a(0,z), \beta_z, t - t_i^{\text{start}}\right) \\ \qquad \forall\, 2 \le i \le n_A \ \wedge\ t_i^{\text{start}} < t \le t_i^{\text{end}} \end{cases} \tag{5}$$

Here, we have $n_A$ applications with indices $1 \le i \le n_A$, where the first application starts at time $t_1^{\text{start}} = 0$. Note that we use the cropped temperature profiles, as determined in Equation (4). We employ the temperature function $C(\cdot)$ to offset every thermal profile in accordance with our temperature model so that there are no discontinuities in the thermal trace at the concatenation points. As a final step, we derive the label sequence $L^A(t)$ corresponding to the thermal sequence $S^A(t,z)$. The label sequence $L^A(t)$ is the numerical representation of the application label for each time step.

Figure 4 illustrates the thermal sequence generated from the example application sequence **A** defined as

$$\begin{aligned} \mathbf{A} = (\ & <\text{HO}, 0.00\,\text{s}, 1.13\,\text{s}>, <\text{GA}, 1.13\,\text{s}, 4.83\,\text{s}>, <\text{-}, 4.83\,\text{s}, 6.79\,\text{s}>, \\ & <\text{HO}, 6.79\,\text{s}, 8.52\,\text{s}>, <\text{WB}, 8.52\,\text{s}, 13.04\,\text{s}>, <\text{HO}, 13.04\,\text{s}, 15.60\,\text{s}>, \\ & <\text{MA}, 15.50\,\text{s}, 17.38\,\text{s}>, <\text{HO}, 17.38\,\text{s}, 18.97\,\text{s}>, <\text{CL}, 18.97\,\text{s}, 20.22\,\text{s}>) \end{aligned} \tag{6}$$

The application labels are outlined in Table 1, and the thermal profiles were collected from a Sony Xperia Z5 smartphone and the settings outlined in section 7. The example illustrates that our data augmentation scheme is able to generate traces that look realistic and do not show any discontinuities.

## 5 The sequence model

In this section, we show how we can apply well-known methods from time-series and sequence-to-sequence modelling to mount a thermal side channel attack. In particular, we describe an implementation of the sequence model, as shown in Figure 2. Its purpose is to transform the received thermal sequences from all the thermal zones $S^A(t, z)$ into a sequence of labels $L^{A'}(t)$, i. e., the sequence of presumably running applications. We describe the chosen basic neural network architecture, as well as the training setup.

### 5.1 Neural Network architecture

The thermal side channel is characterised by complex, largely unknown and non-deterministic properties. First, the running applications can only be identified through their time-dependent usage pattern of the various components of the device such as its CPUs, GPU, memory and dedicated processing components. This usage pattern is unknown and non-deterministic, as it depends on the interaction of the user with the application, as well as data input. Second, the usage pattern leads to distributed power consumption that is converted to temperature changes and heat diffusion through an unknown thermal model of the device. Last but not least, interferences from changing ambient temperature, air flow, running software services, as well as measurement noise change the temperature pattern received by the sink application.

The transformation of a running application to a corresponding thermal sequence involves long-term and state-dependent behaviour. For example, an application can be identified by a sequence of usage patterns of the various components of the device and, therefore, memorising and identifying this sequence of usage patterns is an essential prerequisite for the sequence mode. The transfer from usage patterns to the thermal response at the thermal zones of the device also involves long-term state dependencies. In this case, the thermal energy that is diffusing. As a result, an effective sequence model needs to be able to flexibly represent state-dependent behaviour internally, i. e., it should have an internal state.

Due to this complexity of the input data, we choose a Neural Network (NN) based sequence model, which is able to learn the relation between running applications and the received thermal sequence. We compose this NN based sequence model using a feed-forward Convolutional-Neural-Network (CNN) for feature extraction and a Long Short-Term Memory (LSTM) based Recurrent Neural Network (RNN) to obtain the temporal relation between the thermal features. We give a brief overview of these two NN types in the following two sections.

#### 5.1.1 The Convolutional Neural Network

A Convolutional-Neural-Network (CNN) is one of the techniques used in time-series processing, for example, seismic data [25]. As stated by Goodfellow et al. [9, Chapter 9], a CNN implements the convolution operation to extract feature maps from the input using kernels, defined as

$$y(t) = (x * k)(t) = \sum_{i=-\infty}^{\infty} (x(i) \cdot k(t-i)) \tag{7}$$

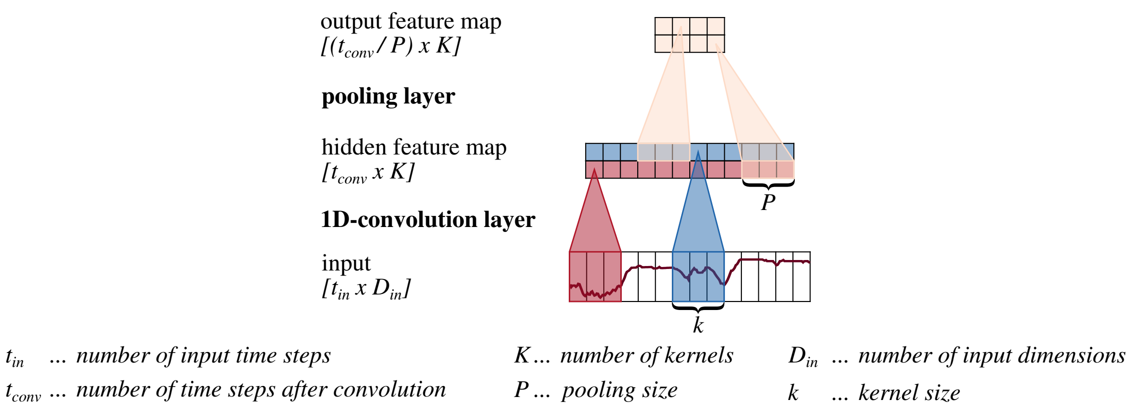


**Figure 5** A simplified example of a one-dimensional Convolutional-Neural-Network (CNN). Note that the number of time steps after the 1D-convolution $t_{conv}$ depends on the kernel size $k$ and the data padding used for the convolution.

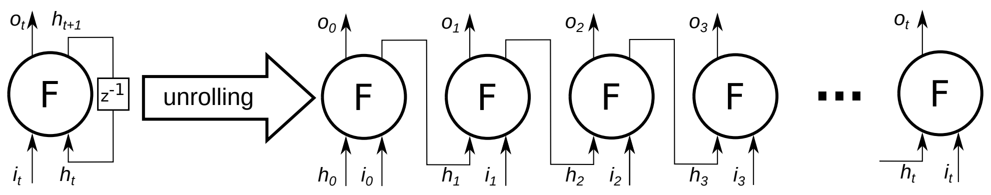


**Figure 6** A Recurrent Neural Network (RNN) cell, in simple representation (left), and unrolled over time. $i_t$ represents the input, $o_t$ the output and $h_t$ the internal state at time $t$.

where $x(t)$ is the input and $k(t)$ is the kernel. In our model, we employ one-dimensional feed-forward convolutional layers to transform the input to feature maps. Furthermore, we use pooling layers to reduce the size of the convolutional layer output. Roughly speaking, a CNN can efficiently transform our time-series data into a suitable format for further processing, while reducing the data size and thus reducing the complexity of following computation steps.

Figure 5 illustrates an example of such a CNN. The input data has a length of $t_{in}$ time steps and $D_{in}$ dimensions. Different dimensions can be, for example, different sensors that are read. The 1D-convolutional layer processes the input data to obtain the so-called hidden feature map, which has a size of $[t_{conv} \times K]$. Here, $t_{conv}$ defines the number of time steps in the data after the convolution operation, which depend on the size of the kernel $k$ and the padding method that is used. $K$ defines the number of different kernels, or filters, used by the 1D-convolutional layer, illustrated by the different colours in Figure 5. The hidden feature map is downsampled by the pooling layer using a fixed pooling size $P$ and a pre-defined function, for example, max. The pooling layer outputs the final feature map of size $[(t_{conv}/P) \times K]$.

### 5.1.2 The Long Short-Term Memory based Recurrent Neural Network

In contrast to simple feed-forward neural networks, RNNs have internal feedback loops that allow them to capture, represent and use temporal information in the input sequences. Figure 6 illustrates an RNN consisting of a single cell, feeding the internal state of the current time-step to the next time-step. For training, the RNN is unrolled over time (see Figure 6), to perform a *back-propagation* through time for computing gradients. In other words, after unrolling, the input vector dimensions of the RNN are extended by the time dimension. For example, a RNN that takes an input vector with $N_F$ feature dimensions and is unrolled $N_T$ time-steps, will take an $N_T \times N_F$ input matrix during training.

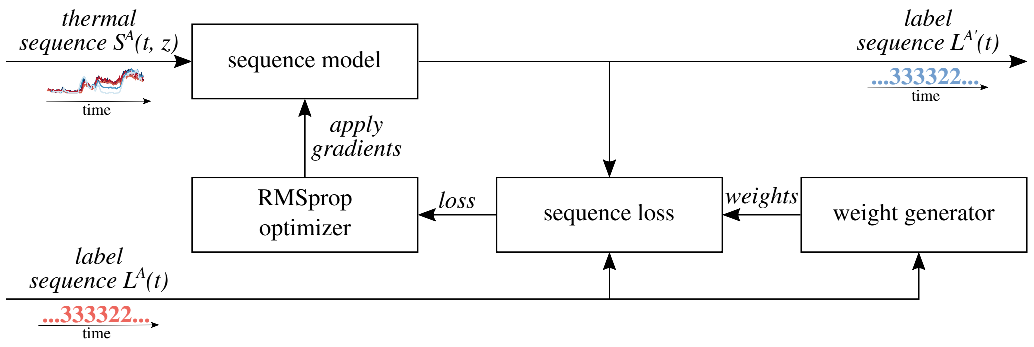


**Figure 7** Setup during the training phase of the sequence model.

The main issue of simple RNN cells is the *vanishing-gradient-problem*, which does not allow them to capture long-term temporal relations in the data [12, Section 3.2]. This problem arises because, as the weights are shared for all time steps, input values are multiplied with the same weights, which leads to exponential decay of the sensitivity of the nodes towards the input data. To compensate the vanishing-gradient-problem, Hochreiter and Schmidhuber [14] designed Long Short-Term Memory (LSTM) cells with an input, a forget and an output gate. These gates allow LSTMs to control the information flow over time (long-term memory) better and, therefore, compensate for the vanishing-gradient-problem [12, Chapter 4].

LSTMs are often implemented as bi-directional networks, which capture temporal relationship in the data in both directions. Simply speaking, bi-directional networks consist of two sub-networks, one of which traverses the data from past to future and the other from future to past [12, Section 3.2.3]. Bi-directional networks often perform better, but they are more complex due to the increased size of the network. In addition, bi-directional networks cannot be used on-line as they require information from the future to process a label output. Therefore, we will use simple uni-directional LSTM layers in our sequence model.

## 5.2 Model structure and training setup

For our experiments, we use a network consisting of

**(i)** one convolutional layer with 64 filters and a kernel size of 25 (1 s),
**(ii)** one max-pooling layer,
**(iii)** 4 LSTM-layers with 128 units each, and
**(iv)** a dense layer with as many units as labels in the experiment scenario.

The CNN reduces the amount of data fed to the LSTM layers and extracts the most important thermal features. Using the LSTM layers, we derive and memorise timing-related information from the internal thermal feature stream. Lastly, the dense layer converts the LSTM layer output to a one-hot-encoded output label vector.

Figure 7 illustrates the setup for training of the sequence model. The input for the training are the thermal sequence $S^{\mathrm{A}}(t, z)$ and the corresponding label sequence $L^{\mathrm{A}}(t)$, which we generate as described in section 4.

During training, we use the *sequence loss* from the TensorFlow 2.0 Addons `seq2seq` package[3]. This loss function allows us to weight the individual samples of the trace for two purposes:

[3] `https://www.tensorflow.org/addons/api_docs/python/tfa/seq2seq/sequence_loss`

**(i)** if the input trace length is shorter than the actual input length of the model, we use zero weights to indicate which samples should be ignored, and

**(ii)** the weights allow us to compensate if some labels appear disproportionately often in the training data to ensure the training puts the same emphasis on all labels (example-weighted training).

The *weight generator* generates the weights depending on how often a label occurs in a batch. For example, in a batch label with two labels A and B, A occurs twice as often as B. In this case, the weights are 0.5 for label A and 1 for label B. The determined loss is then fed to the *RMSprop optimiser*, an adaptive learning rate optimiser which has proven to work well for many applications [41]. We choose the RMSprop optimiser because it is widely used and has been empirically evaluated to outperform simple stochastic gradient decent in terms of training time[4]. However, we acknowledge criticism towards stochastic optimisation techniques that has been raised in the past [34], but leave an evaluation of different optimisers for future work.

To avoid over-fitting, we use dropout layers and early stopping during training. The early stopping is triggered whenever the loss on the test set does not decrease, and the per-time-step accuracy does not increase after 20 epochs of training.

## 6 Sequence transformation and performance metrics

Following the information flow, as depicted in Figure 2, the sequence model transforms the thermal sequence into a label sequence. Due to the limited view of the sequence model on the relation between the application sequence and the resulting temperature sequence, we can observe artefacts when switching between applications. Finally, in order to evaluate the difference between the initial application sequence and the predicted application sequence, appropriate metrics need to be defined.

### 6.1 Sequence transformation

After training the sequence model, we feed an example thermal sequence to obtain the label trace, as illustrated in Figure 8. The example is generated using thermal profiles collected from a Sony Xperia Z5 smartphone, using the experimental setup outlined in section 7. In this section, we highlight issues in the label trace $L^{A'}(t)$ as produced by the sequence model and provide an approach to address them. Figure 8 illustrates a thermal sequence using the applications outlined in Table 1 and thermal profiles from a Sony Xperia Z5 (see section 7 and section 8). (a) shows the thermal sequence $S^A(t,z)$ for one of the zones, (b) the output label trace $L^{A'}(t)$ from the time-sequence model, (c) the output application sequence $\mathbf{A'}$, and (d) the ground truth $\mathbf{A}$.

The literature offers a variety of different approaches for a transformation from a label per time unit to a label interval. For example, combining the RNN with a Hidden Markov Model (HMM) or using the so-called Connectionist Temporal Classification (CTC) algorithm [12]. CTC based models provide an output distribution over all possible application sequences for a given input. One can use this distribution either to infer a likely application sequence or to assess the probability of a given one. However, as such advanced approaches require a considerable amount of training and computing overhead, we resort to the following simpler two-step approach: Use a filter to avoid the jitter label and a label condensing rule to convert $L^{A'}(t)$ into $\mathbf{A'}$. We consider this simple approach sufficient for our application, as we do not need to differentiate whether

[4] `https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a`

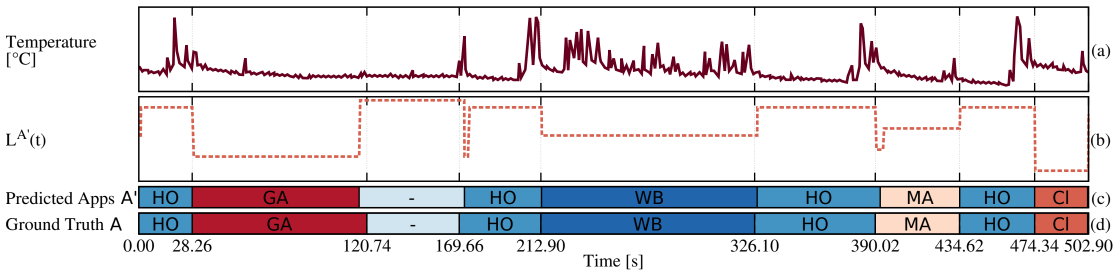


**Figure 8** A thermal sequence processing example taken from the evaluation in section 8 using thermal profiles collected from a Sony Xperia Z5. shows (a) the thermal sequence $S^A(t,z)$ for one of the zones, (b) the labelling sequence $L^{A'}(t)$, (c) the predicted application sequence $\mathbf{A}'$ and (d) the actual application sequence $\mathbf{A}$. The application Labels correspond to the labels outlined in Table 1. The plot shows that the sequence model is capable of predicting correct labels with only a small amount of timing inaccuracy. Yet, labelling artefacts at the application pre-emption points at 169.66 s and 390.02 s occur. However, the sequence transformation using a majority voting filter and label condensing is able to compensate for such labelling artefacts.

an application has been executed multiple times in a row or once for a longer period of time. Nevertheless, the parameter of the filter should match the minimal time period of an application to be recognized.

In Figure 8, the output label trace $L^{A'}(t)$ at the pre-emption points at 169.66 s and 390.02 s jumps between multiple labels. Therefore, there is a need for an additional filter to eliminate such a label jitter. We apply a simple sliding window with majority voting replacement with a window length of 25 s as the minimal length of an application execution is larger. This window size is sufficiently large to compensate the label jitter and other labelling artefacts, yet small enough to allow for a sufficient temporal accuracy of the labelling.

After determining the filtered per-time-step labelling sequence $L^{A'}(t)$, we have to apply a final transformation to derive the predicted application sequence $\mathbf{A}'$. All consecutive equal labels are combined to a single label. By tracing the start and end time of the condensed labels, we can determine when an application is executed. The example from Figure 8 shows that the network is capable of determining the correct application sequence and that the timing is reasonably accurate. The latency of the labels at the application pre-emption points can be considered normal as Graves [12] already stated that uni-directional LSTM models often set the labels with some delay. However, the example also shows that if the thermal trace does not contain sufficient thermal features, misclassifications might happen. As the gaming application (GA) seems to be idle at the pre-emption point at 120.74 s, the sequence model sets the unknown label ("-") too early.

## 6.2 Performance metrics

In order to evaluate the performance of the whole approach as outlined in Figure 2, we need to determine different metrics. First, we derive the per-timestep accuracy of the label sequences $L^A(t)$ and $L^{A'}(t)$, which are of equal length. The per-timestep accuracy is the number of equal elements in $L^A(t)$ and $L^{A'}(t)$, divided by the length of $L^A(t)$.

To get a more detailed understanding of the performance of our approach, we also evaluate the difference between the initial application sequence $\mathbf{A}$ and the predicted one $\mathbf{A}'$. The associated challenges are due to the following characteristics:

**(i)** two application sequences $\mathbf{A}$ and $\mathbf{A}'$ might have different lengths,
**(ii)** element-wise comparison of two sequences may lead to misleading results, and
**(iii)** we are interested in the temporal correctness of the predicted application sequence in addition.

For example, the element-wise comparison of the correct sequence “ABDABABAB” and predicted sequence “ABDBABAB” yields a relative error of 5/8 when taking only the shorter application sequence as a reference. Besides the problem of different sequence lengths, we can also observe that there is just one difference in the two sequences, namely the missing “A” after “D”. A metric that can handle sequences of different lengths is the relative *Levenshtein distance*, also known as relative edit distance. The relative Levenshtein distance is the minimal number of modifications that have to be applied to a sequence to be equal to another one, divided by the length of the correct sequence. Possible modifications are insert, delete and replace. For our example sequences, the relative Levenshtein distance is 1/8 as we just need to insert the application “A” after “D”. This result shows that the two sequences are rather similar in terms of the relative Levenshtein distance, which is closer to our intuition.

The second metric we use for the final evaluation is the average timing error or temporal label placement error. A naive approach would measure the time error of the label placement by calculating the *Euclidean distance* between the predicted application pre-emption points and the actual pre-emption points. However, the predicted application sequence can contain a different number of application pre-emptions than the real application sequence. Therefore, we combine the Euclidean distance with the Dynamic Time Warping (DTW) algorithm [37]. DTW will calculate the Euclidean distance between all pre-emption points reported in the predicted application sequence, with the most similar pre-emption point in the true application sequence, and report the sum of all distances calculated for the sequences. For example, let us assume the network reports three application pre-emptions at 5 s, 8 s and 11 s, while the true sequence only contains two pre-emption points at $5\,s$ and 9 s. Using DTW, the reported Euclidean distance will, therefore, be 0 s $+1$ s $+2$ s $=$ 3 s. To normalise the time error metric, we divide the value reported by the DTW algorithm by the number of actual pre-emptions in the true sequence. This would result in an average time error of 1.5 s in the example. If a predicted application sequence only contains one application and, therefore, no pre-emption point, the average timing error is $NaN$.

## 7 Target Setup

The final performance evaluation of the thermal side channel attack is based on data from real smartphones. We chose two different smartphones from two different vendors for our evaluation:

- A Samsung Galaxy S5 SM-900H based on a Samsung Exynos 5422 SoC, with Android 5.0 and 3 thermal zones; from now on referred to as S5.
- A Sony Xperia Z5 based on a Snapdragon 810 SoC with Android 7.0 and 36 thermal zones, referred to as Z5.

If not otherwise specified, the two smartphones are placed in an air-conditioned server room with approximately 22°C and are connected to the power outlet. To generate our datasets, we use a measurement setup based on the ExOT (see Miedl et al. [29]). ExOT provides building blocks for measurement applications, as well as an experiment execution and analysis flow. To read the thermal zones, we employ a sink application which reads the corresponding sysfs files[5].

In addition, for the evaluation purposes, the sink application determines the current foreground application, i. e., the ground truth. Note that this just implemented for determining the ground truth and is not part of the attack scenario at all. The foreground application is determined by querying the usage stats or activity manager, depending on the Android build. As these methods require elevated privilege levels, in a real attack the sink application is not able to determine the foreground application.

[5] On the most Unix based systems the thermal zone nodes can be accessed via the `sysfs` where `$i` is the respective thermal zone number: `/sys/devices/virtual/thermal/thermal_zone$i/temp`

**Table 1** Used applications and associated labels during the performance evaluation.

| Label | Application Name | Description |
|---|---|---|
| VM | AnTuTu | Benchmark Suite |
| CL | Dropbox | Cloud storage application |
| DO | Document Viewer | Standard document viewer |
| GA | Angry Birds Rio | Game |
| SM | Facebook | Social Media client |
| IM | Wire | Instant messaging service |
| WB | Chrome Browser | Web browser |
| MA | Gmail | E-mail client |
| LO | Google Maps | Location services |
| VI | YouTube | Video streaming service |
| HO | Launcher/Home | Android system |
| - | Blank/Unknown | Unknown application |

The sink application is implemented as an Android background service and is configured to conduct a measurement every 1 ms. However, due to the Android scheduling policy, the sampling period is longer and fluctuates. This results in an average sampling period of approximately 30 ms on S5 and 46 ms on Z5. To get equally-spaced samples for further data processing, we re-sample all thermal traces with a sampling period of 40 ms.

To minimise the data processing overhead, we embed our analysis into the existing ExOT framework to take advantage of the data processing stack. We randomly choose ambient thermal offset between −35°C and 35°C and allow multiple ambient temperature changes per batch. Table 1 outlines all possible foreground applications and the associated labels, if not defined otherwise. We selected those applications since they cover the most common use cases of a current smartphone.

## 7.1 Thermal parameters

For applying a thermal side channel attack according to Figure 2, we need to train the corresponding sequence model, see Figure 3. To be able to apply the data augmentation scheme as described in section 4, we need to characterise the device and determine the necessary thermal parameters, namely, the internal thermal, ambient heating and ambient cooling coefficients.

We also use the thermal coefficients to select the thermal zones which show high thermal dynamics, i. e., potentially carry useful information. Initial measurements on Z5 show that 24 of the 36 thermal zones provide usable thermal measurements. The other thermal zones either only provide 0°C outputs or are not affected by any application execution. Furthermore, on S5, we exclude the battery sensor as it does not show any application-dependent thermal variations. All parameters are outlined in Figure 9.

To determine the thermal coefficients $\beta_z$, we place the target devices in a testbed [38, Chapter 3], which allows us to control the temperature in a range of approximately 15°C to 50°C. For each ambient temperature, we conduct two measurements where we keep the device idle for 3 minutes and then execute a CPU benchmark[6] for either 70 or 130 seconds. The execution of the benchmark heats up the device, and we observe the cooling-off phase for 3.5 minutes, as illustrated in Figure 10.

[6] CPU Throttling Test (`https://play.google.com/store/apps/details?id=skynet.cputhrottlingtest`)

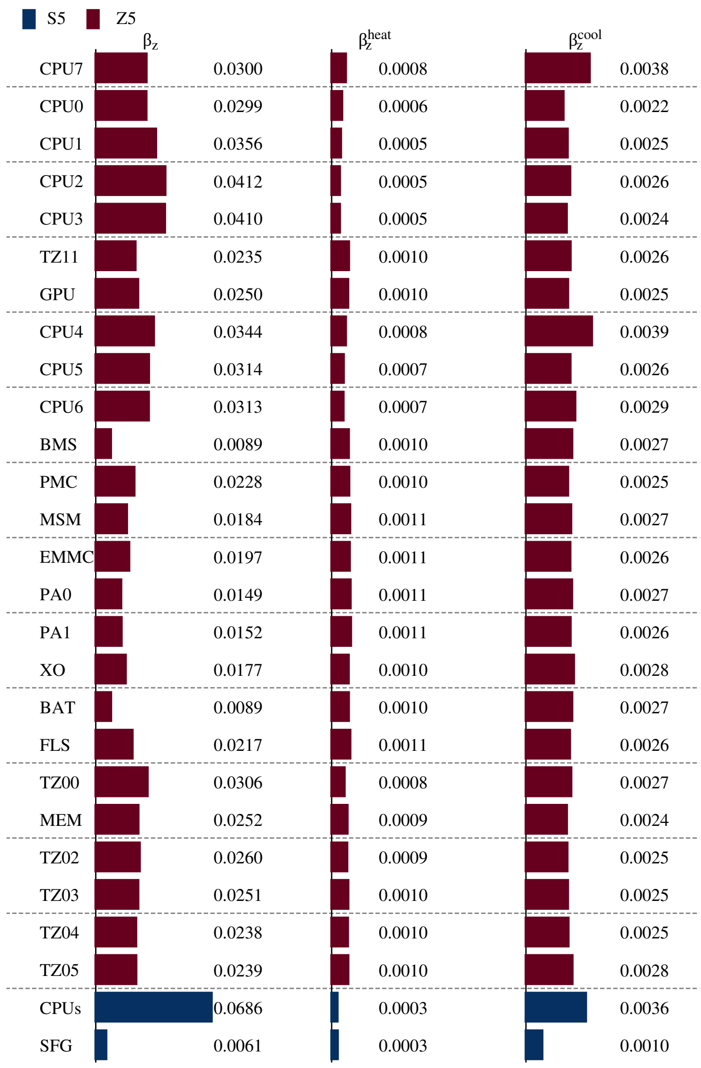


**Figure 9** Thermal coefficients for internal ($\beta_z$), increasing ($\beta_z^{\text{heat}}$) and decreasing ambient temperatures.

To determine the ambient heating and cooling coefficients, we simply move the idle phones to different locations in- and outside of an office building. From the resulting thermal trace, we isolate the resulting thermal changes and determine $\beta_z^{\text{heat}}$ and $\beta_z^{\text{cool}}$. The results in Figure 9 illustrate that the ambient thermal coefficients $\beta_z^{\text{heat}}$ and $\beta_z^{\text{cool}}$ are significantly lower than the respective $\beta_z$, on both platforms. However, due to the length of our temperature observations, the long-term influence of varying ambient temperatures cannot be neglected.

The thermal coefficients illustrated in Figure 9 show that the CPUs have the highest thermal dynamics on both smartphones S5 and Z5. Furthermore, while the internal thermal coefficients $\beta_z$ for the cores are high compared to other thermal zones, the influence of increasing ambient temperatures is rather low, as indicated by the comparison of $\beta_z^{\text{heat}}$ for different zones. Therefore, we will only use the eight CPU thermal zone readings on S5 and the single CPU cluster reading on Z5 as input for the application detection.

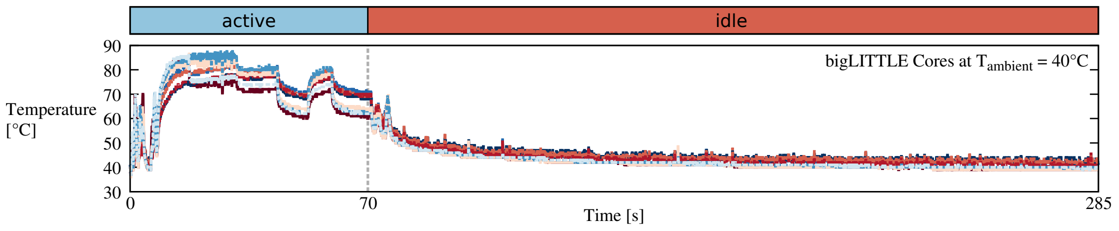


**Figure 10** The device is heated up by an active application and we derive the thermal coefficients from the measurements taken when the device is idle and cools down.

## 7.2 Preparing thermal profiles for data augmentation

In order to generate the thermal profiles $T^a(t, z)$ required by the data augmentation scheme, see Figure 3, we need user inputs to interact with the selected applications. We use the RepetiTouch Pro application[7] to record user inputs. RepetiTouch Pro also allows us to replay recorded user inputs to generate multiple thermal profiles for one application usage. In our experimental setup, the user inputs are collected by a single user, who executes 49 different use cases. Each use case defines the usage of one application in a specific manner. By defining multiple use cases for each application, we ensure that the model learns the thermal profile of an application rather than one specific use case. In order to increase the pool of thermal profiles to feed to the data augmentation scheme, we record 10 traces per use case, i. e., 10 thermal profiles per use case.

The collected thermal profile traces can contain labels which we do not specify in Table 1. This is caused by dynamic application content, for example, advertisement pop-ups. As the execution of recorded user inputs with RepetiTouch is static, such dynamic application content causes deviations from the defined use case and results in unknown application labels. Depending on the evaluation scenario, we remove parts with unknown labels and use data augmentation to ensure there are no temperature discontinuities in the thermal profiles.

Moreover, we define that the data augmentation scheme uses thermal profile snippets with a length of at least 30s. Hence, our scheme is limited to a minimal use of a single application of 30 seconds. Shorter usages of applications are not considered at this stage.

## 8 Performance evaluation

In this section, we evaluate the performance of the thermal side channel attack based on the training and test datasets defined in subsection 7.2. We divide our performance evaluation into multiple scenarios. For each scenario, we generate 28 training and 7 test batches, with a batch length of 3 hours and 45 minutes (13500 s). The augmentation scheme uses 8 thermal profiles of each use case to generate the training dataset and the remaining 2 thermal profiles of each use case as a base for the test dataset.

**No Augmentation Scenario.** In the "No Aug" scenario, we train the network using the raw data we have collected without using the proposed augmentation technique. Thermal sequences that are shorter than the specified model input are zero-padded, and we use zero weights so that the training is not affected by the padding. This results in 325 training batches, one for each application use case. As a test, we generate 7 batches by simply concatenating recorded lab traces without augmenting the ambient temperature and the dynamic offset.

[7] https://play.google.com/store/apps/details?id=com.cygery.repetitouch.pro

**Whitebox Scenario.** The training dataset for the “Whitebox” scenario, generated using the data augmentation scheme, only contains known labels. This means we remove all portions with unknown labels from the datasets as described in subsection 7.2.

**New Apps Scenario.** In the “New Apps” scenario, we evaluate the performance of the model if a new application is added to the device after training using augmented data. In contrast to the “Whitebox” scenario, we keep all unknown labels in the training dataset. Furthermore, we record thermal profiles using the AndroBench[8] application and add it to the test data. Instead of introducing new labels for applications not defined in Table 1, the unknown (“-”) label is used.

## 8.1 Expressiveness of the performance metrics

First, we intend to validate whether the chosen performance metrics are sufficiently expressive. To this end, we perform a manual evaluation by means of a visual inspection of a sample of the “New Apps” scenario, shown in Figure 11, and compare it to the performance metrics illustrated in Figure 12.

The per-time-step accuracy outlined in Figure 12 indicates that the models perform better with data from Z5 than S5, which is also supported by the trace illustrated in Figure 11. However, the relative Levenshtein distance is quite high for both platforms, as the experiments yield 0.83 for S5 and 0.70 for Z5. This indicates that the model misclassifies short thermal patterns, which are not compensated by the majority filtering. As the durations of the misclassified trace intervals are very short, they cause a higher relative Levenshtein distance while still yielding a high per-time-step accuracy. We assume that this behaviour is mainly caused by the fact that thermal patterns occurring during the execution of an application are very similar to the thermal pattern when starting other applications and, therefore, are misclassified. A possible solution to this issue could be to increase the long-term-memory of the model, to increase the amount of temporal context information that is taken into account by the model when placing the labels. The majority filter size could also be increased, but this would also increase the minimal detectable application execution length. This example shows that neither the per-time-step-accuracy nor the relative Levenshtein distance on its own are expressive regarding the performance of the model. Only when combined can these metrics be used to assess the performance of the models reliably.

The average temporal errors, shown in Figure 12 for the two platforms, S5 and Z5, are significantly lower than the length of the majority filtering window of 25 s. This indicates that the temporal error of the labelling models is mainly due to label misclassifications. This assumption is also supported by the labelling sequences illustrated in Figure 11.

## 8.2 Effectiveness of the data augmentation scheme

The experimental results depicted in Figure 12 show that training without augmentation (“No Aug”) is not able to learn a proper sequence model, in contrast to the “Whitebox” and the “New Apps” scenario.

To provide a more detailed experimental analysis of the effectiveness of the data augmentation scheme, we run an experiment where we generate the training data with a reduced set of raw thermal profiles, i. e., we use 2 instead of 8 thermal profiles per use case as a basis for the training dataset. Figure 13 illustrates the results for the “New Apps” scenario with the normal and a reduced raw thermal profile set as used for generating the training dataset. For Z5, the performance for the reduced training data scenario decreases. For S5, however, training fails completely. We assume that these two effects are caused by the following factors:

[8] `https://play.google.com/store/apps/details?id=com.andromeda.androbench2`

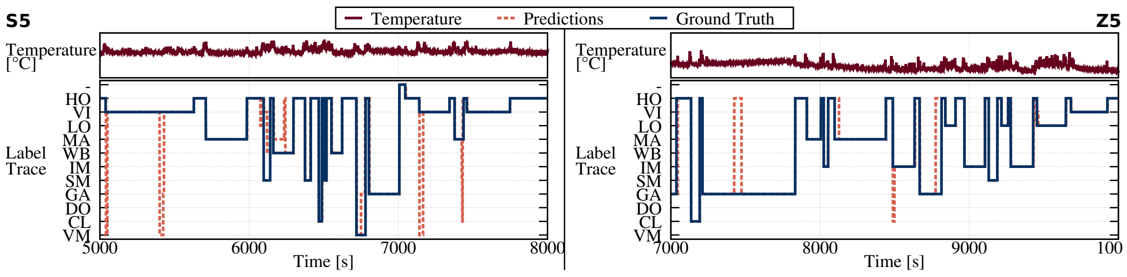


Figure 11 Traces indicate a higher amount of labelling errors on S5 than on Z5.

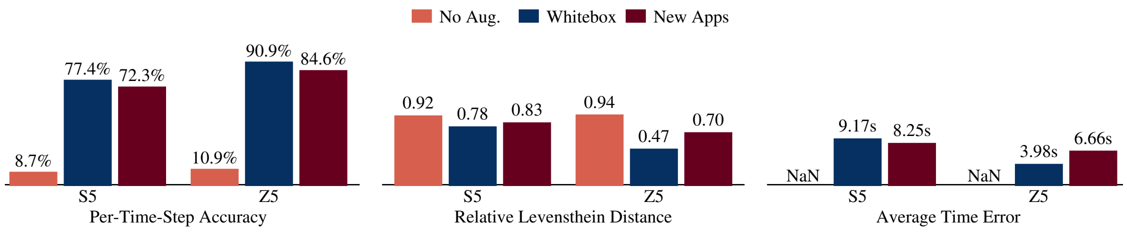


Figure 12 Training the model without augmented data is not possible. Models trained with augmented data perform well, but adding a new application cause performance degradation.

- The performance for Z5 degrades as the model does not generalise well. The augmentation scheme is not capable of generating a training data set with sufficient variance to allow the model to generalise well based on two very similar thermal profiles for each use case.
- For S5, training of the sequence model is not possible anymore, as the thermal profiles chosen for the training set are not representative. This might be caused by the fact that the measurement of the thermal profiles contains more measurement artefacts. Therefore, the smaller the set of thermal profiles, the more important their quality.

In sum, we can state that while the proposed augmentation scheme helps to generate datasets which allow for the training of the sequence model, there are some caveats that need to be considered. The quality of the training dataset highly depends on the quality of the thermal profiles. However, as there is the risk of measurement artefacts, data cleansing is necessary, as the quality of the training and test data highly depends on the quality of the initial raw data. Unfortunately, data cleansing is a process which can hardly be automated and, to some extent, will always have to be done manually.

## 8.3 Performance on different devices

The results illustrated in Figure 12 show that in general models trained for Z5 outperform the models for S5. While we mentioned in the previous subsection that the quality of the thermal profiles collected on S5 seem to be less representative, we also assume that the lower performance on S5 is caused by the lack of thermal information that we can collect. On Z5, we can use 8 sensor reading for 8 cores, while on S5 we can only get 1 reading for all 8 cores. Therefore, the amount of thermal information that we can extract is lower on S5, which lowers the performance of the sequence model.

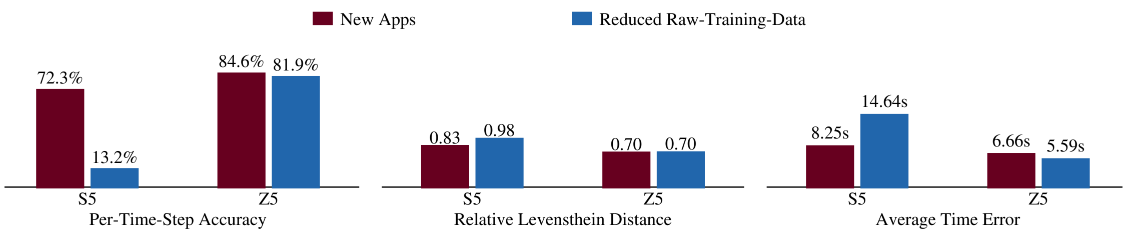


**Figure 13** Reducing the base dataset of thermal profiles for the data augmentation causes a slight performance drop for Z5, but has a substantial impact on the performance for data from S5. This is caused by the dependency of our data augmentation scheme on the amount and quality of the thermal profiles available for data generation.

## 8.4 Influence of new applications

To assess the influence of a new application in the test dataset, we compare the performance metrics for the “Whitebox” and the new app scenario illustrated in Figure 12. The metrics suggest that the model performance suffers when the test data contains applications labelled as unknown, which are not present in the training dataset. A detailed analysis of the labelling traces shows that the accuracy degradation of the per-time-step accuracy from the “Whitebox” to “New Apps” scenario is approximately the amount of unknown label in the test dataset. Therefore, we conclude that our models are not capable to properly label new applications with the unknown (“-”) label.

The models react as expected and simply label the new application with the application label, which has the most similar thermal profile. This issue can be addressed by either implementing unsupervised online learning to update the model constantly or offline re-training of the model. However, these extensions are left for future work.

## 8.5 Single application detection

For the final laboratory test, we evaluate the performance of the model when it only has to detect whether a specific application is running or not. Therefore, in the training set, we only have two labels

**(i)** the targeted application, or
**(ii)** unknown.

Figure 14 illustrates the performance metrics for the different applications, i. e., each case corresponds to a completely new learning scenario where one of the applications is known and all other applications are labelled as unknown (“-”).

Except for Chrome (“WB”), Google Maps (“MA”) and YouTube (“VI”), on both platforms single applications are detected with an accuracy above 80%, sometimes even exceeding 90%. However, we also have to consider the relative Levenshtein distance, which is above 0.5 for many test-cases and also high average timing errors. We assume that the lower performance for some applications is caused by more ambiguous thermal profiles. For example, for YouTube (“VI”), the thermal profile is very different depending on whether a video is played or the application is only opened and the search function is used, without video playback. Therefore, we conclude that it might also be useful to introduce multiple labels per application to differentiate, for example, different operations like menu and video replay.

Training a model for single application detection is harder than for the multilabel case. This is caused by the fact that the real data and the training data might be very different. While we configured our augmentation scheme to generate training data sets to contain many occurrences of the target application, in real life, a target application might not be used for a long time. However,

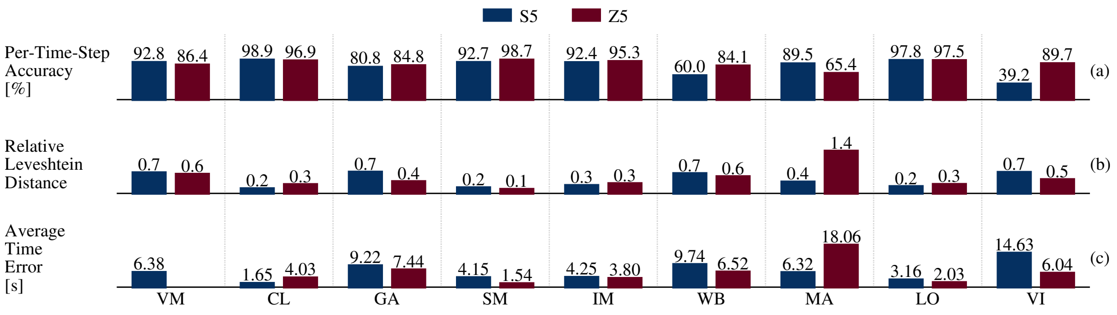


**Figure 14** Performance metrics for the detection of a single application. While the per-time-step accuracy is very high for most applications, the relative Levenshtein distance varies between 0.1 to 0.6. This indicates that the applications with a high relative Levenshtein distance have a less unique thermal profile. Furthermore, the time errors indicate that the labels are placed with an almost perfect temporal accuracy, as they are considerably smaller than the majority voting filter window of 25 s.

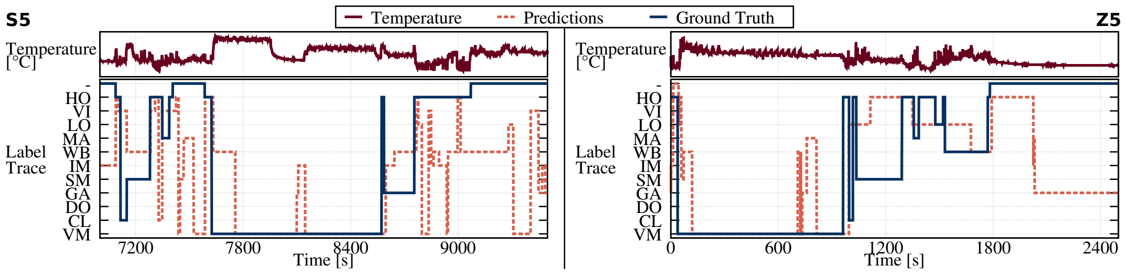


**Figure 15** The real-world traces for both smartphones, S5 and Z5. The short temporal snippets validate the performance metrics and illustrate that the models are not able to correctly label the thermal trace.

if the target application does not appear often enough in the training data, the model will not be able to learn the thermal profile of the application. On the other extreme, if the target application thermal profile is inserted too often and with an unintended periodicity, the model might learn the periodicity in which the target application appears, rather than the thermal profile. Therefore, a model needs to be trained carefully to learn the target application thermal profile without expecting this thermal profile to occur regularly.

If trained properly and when considering the observations from subsection 8.4, a model for single application detection might be more robust in a real deployment. Let us consider that the multidimensional feature space used by the RNN to classify the thermal traces is a high-dimensional Euclidean space. In such a case, the similarity of two thermal profiles can be illustrated by the distance between the two corresponding points in the feature space. The space mapped to the target application label will be small compared to the space which maps to all the other unknown applications. Therefore, if a new application thermal profile is presented to the model, its representation will most likely be mapped to the unknown label. Hence, the model performance will not degrade, i. e., the model is robust against new applications.

## 8.6 Real-world applicability

In addition to the lab generated data, we also collect a real-world trace: the same user who recorded the inputs for the laboratory setup carries each smartphone for a day. The user freely runs the applications available on the smartphones to imitate normal behaviour. As no data is recorded when the smartphone is locked, this results in trace lengths of

**(i)** 3h (10800 s) for S5, and
**(ii)** 4.5h (16200 s) for Z5.

The models trained with the "New Apps" scenario training set achieved a per-time-step accuracy of less than 20%. This means that the models are not able to detect any applications in the real-world thermal trace correctly, as outlined in the example illustrated in Figure 15. Also, models for the binary use cases were not capable of detecting the respective target application in the real-world trace.

The real world data shows more short term variability than the laboratory data. Causes for this are the variable temperature influences from the environment and the position of the phone, e.g. how the phone is held and operated. Furthermore, the laboratory traces were collected with a minimum amount of applications running in the background, while during the real world data more applications might be active and disturb the thermal profile of a foreground application. Therefore, the significance and shape of the thermal features that the model uses change, which makes it harder for the model to correctly infer the right application based on thermal data. In addition, the user interaction pattern in the real-world experiment could have deviated significantly from the typical usage patterns recorded for training. This would have resulted in significantly different thermal profiles and consequently, higher error.

## 8.7 Future directions

While the results based on laboratory data are promising and clearly show the threat potential of the thermal side channel, the tests using data collected outside of the laboratory illustrate that there are still challenges to overcome to implement the thermal side channel attack in a real environment. As the sequence model requires a large amount of data for training, we do not consider manual data collection outside of a laboratory setup a viable alternative to a data augmentation scheme. However, the data augmentation scheme needs to be refined to resemble real-world data more closely. This includes, for example, influences on the temperature of the smartphone when it is held in the hand compared to sitting on a table. In addition, the current scheme does not take into account that the set of applications running in the background changes under normal use. This influences the scheduling and core pinning of the foreground application and adds noise to the profile, whereas in our evaluation all thermal profiles for training are collected from smartphones running the same set of background services.

Furthermore, a study evaluating the relation between severity of the side channel and the amount of available thermal information would allow more insights in possible mitigation strategies. Sparse spatial information, less available sensors, or a low temporal resolution of the measurements might drastically reduce the possibility to mount the thermal side channel attack, while still providing enough thermal information to the power management system.

Another possible direction for exploration are server racks or laptops. Due to the lower mobility we expect the environmental influences to be lower. However, the cooling systems and the high utilisation of those systems might make a thermal side channel attack, as presented in this work, very challenging.

In future work, the online scenario also needs to be evaluated. This requires that the sequence model is optimised for the attacked platforms, to minimise the computation and memory footprint. Otherwise, the attack is easy to detect or it might not even be possible to deploy it, due to the limited hardware capabilities of the smartphones. Moreover, evaluating online learning methods that would allow the model to predict applications it has not seen during the initial training would further increase the capabilities of the thermal side channel attack.

## 9 Concluding remarks

In this work, we presented a data leak based on the measurement of the temperature of a mobile device using its internal sensors. We showed that it is possible to determine which applications were executed at what time. This kind of information can be acquired without the knowledge of the user and may pose a serious security and privacy violation.

We showed how to generate a fitting dataset for training a model based on real-world measurements but without the need for an extensive measurement campaign. Furthermore, we explained in detail how to build and train this time-sequence model using Convolutional-Neural-Network (CNN), Long Short-Term Memory (LSTM) and label trace filtering. In addition, we outlined an extensive laboratory study based on data from two smartphones, a Samsung Galaxy S5 and a Sony Xperia Z5. The results of the laboratory results are promising, with per-time-accuracy of up to 90% for a scenario with 11 different application labels. However, tests using data recorded outside of the laboratory setup revealed that the data augmentation scheme is not sophisticated enough to use laboratory data to generate training datasets that resemble outside use.

Lastly, we showed that it is possible to misuse this thermal information to mount a thermal side channel attack. Because a thermal side channel attack violates security and privacy constraints, action needs to be taken to mitigate this data leak before it can become a real threat.

## References

1 Davide B. Bartolini, Philipp Miedl, and Lothar Thiele. On the Capacity of Thermal Covert Channels in Multicores. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 24:1–24:16. ACM, 2016. `doi:10.1145/2901318.2901322`.

2 J. Brouchier, T. Kean, C. Marsh, and D. Naccache. Temperature Attacks. *IEEE Security and Privacy*, 7(2):79–82, March 2009. `doi:10.1109/MSP.2009.54`.

3 Julien Brouchier, Nora Dabbous, Tom Kean, Carol Marsh, and David Naccache. Thermocommunication. Cryptology ePrint Archive, Report 2009/002, 2009. URL: `https://eprint.iacr.org/2009/002`.

4 Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2016. `doi:10.1109/COMST.2015.2494502`.

5 Hong Cao and Miao Lin. Mining smartphone data for app usage prediction and recommendations: A survey. *Pervasive and Mobile Computing*, 37:1–22, 2017. `doi:10.1016/j.pmcj.2017.01.007`.

6 P Dadvar and K Skadron. Potential thermal security risks. In *Semiconductor Thermal Measurement and Management Symposium, 2005 IEEE Twenty First Annual IEEE*, pages 229–234, 2005. `doi:10.1109/STHERM.2005.1412184`.

7 Dmitry Evtyushkin and Dmitry Ponomarev. Covert channels through random number generator: Mechanisms, capacity estimation and mitigations. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 843–857. Association for Computing Machinery, 2016. `doi:10.1145/2976749.2978374`.

8 Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. Understanding and Mitigating Covert Channels Through Branch Predictors. *ACM Transactions on Architecture and Code Optimization (TACO)*, 13(1), March 2016. `doi:10.1145/2870636`.

9 Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: `http://www.deeplearningbook.org`.

10 Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache Attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security*, EuroSec'17. Association for Computing Machinery, 2017. `doi:10.1145/3065913.3065915`.

11 Michael C Grace, Yajin Zhou, Zhi Wang, and Xuxian Jiang. Systematic detection of capability leaks in stock android smartphones. In *NDSS*, volume 14, page 19, 2012.

12 Alex Graves. *Supervised sequence labelling*, pages 5–13. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. `doi:10.1007/978-3-642-24797-2_2`.

13 Mordechai Guri, Matan Monitz, Yisroel Mirski, and Yuval Elovici. BitWhisper: Covert Signaling Channel between Air-Gapped Computers Using Thermal Manipulations. In *Proceedings of the 2015 IEEE 28th Computer Security Foundations Symposium*, CSF '15, pages 276–289, USA, 2015. `doi:10.1109/CSF.2015.26`.

14 Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, November 1997. `doi:10.1162/neco.1997.9.8.1735`.

15 Michael Hutter and Jörn-Marc Schmidt. *The Temperature Side Channel and Heating Fault Attacks*, pages 219–235. Springer International Publishing, Cham, 2014. `doi:10.1007/978-3-319-08302-5_15`.

16 T. Iakymchuk, M. Nikodem, and K. Kepa. Temperature-based covert channel in FPGA systems. In *Reconfigurable Communication-centric*

*Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on*, pages 1–7, June 2011. `doi:10.1109/ReCoSoC.2011.5981510`.

**17** Mohammad A. Islam, Shaolei Ren, and Adam Wierman. Exploiting a Thermal Side Channel for Power Attacks in Multi-Tenant Data Centers. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1079–1094. Association for Computing Machinery, 2017. `doi:10.1145/3133956.3133994`.

**18** Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203*, 2018. URL: `https://spectreattack.com/`.

**19** Butler W. Lampson. A Note on the Confinement Problem. *Commun. ACM*, 16(10):613–615, October 1973. `doi:10.1145/362375.362389`.

**20** Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. ARMageddon: Cache Attacks on Mobile Devices. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, pages 549–564. USENIX Association, 2016. `doi:10.5555/3241094.3241138`.

**21** Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *arXiv preprint arXiv:1801.01207*, 2018. URL: `https://spectreattack.com/`.

**22** Carol Marsh and David McLaren. Poster: Temperature Side Channels. In *In the Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), 2007*, 2007.

**23** Ramya Jayaram Masti, Devendra Rai, Aanjhan Ranganathan, Christian Müller, Lothar Thiele, and Srdjan Capkun. Thermal Covert Channels on Multi-core Platforms. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 865–880, Washington, D.C., August 2015. USENIX Association. `doi:10.5555/2831143.2831198`.

**24** Clémentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard, and Kay Römer. Hello from the other side: SSH over robust cache covert channels in the cloud. *NDSS, San Diego, CA, US*, 2017. URL: `https://cmaurice.fr/pdf/ndss17_maurice.pdf`.

**25** Matthias Meyer, Samuel Weber, Jan Beutel, and Lothar Thiele. Systematic identification of external influences in multi-year microseismic recordings using convolutional neural networks. *Earth Surface Dynamics*, 7(1):171–190, 2019. `doi:10.5194/esurf-7-171-2019`.

**26** Yan Michalevsky, Gabi Nakibly, Gunaa Arumugam Veerapandian, Dan Boneh, and Gabi Nakibly. PowerSpy: Location Tracking Using Mobile Device Power Analysis. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 785–800, Washington, D.C., August 2015. USENIX Association. URL: `https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/michalevsky`.

**27** Philipp Miedl, Xiaoxi He, Matthias Meyer, Davide Basilio Bartolini, and Lothar Thiele. Frequency Scaling as a Security Threat on Multicore Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2497–2508, November 2018. `doi:10.1109/TCAD.2018.2857038`.

**28** Philipp Miedl, Bruno Klopott, and Lothar Thiele. ExOT Website, March 2020. URL: `https://www.exot.ethz.ch/`.

**29** Philipp Miedl, Bruno Klopott, and Lothar Thiele. Increased reproducibility and comparability of data leak evaluations using ExOT. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020. `doi:10.3929/ethz-b-000377986`.

**30** Philipp Miedl and Lothar Thiele. The Security Risks of Power Measurements in Multicores. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, pages 1585–1592. Association for Computing Machinery, 2018. `doi:10.1145/3167132.3167301`.

**31** Steven J. Murdoch. Hot or Not: Revealing Hidden Services by Their Clock Skew. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 27–36. Association for Computing Machinery, 2006. `doi:10.1145/1180405.1180410`.

**32** Naser Peiravian and Xingquan Zhu. Machine learning for android malware detection using permission and api calls. In *Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, ICTAI '13, pages 300–305, USA, 2013. IEEE Computer Society. `doi:10.1109/ICTAI.2013.53`.

**33** Danny Philippe-Jankovic and Tanveer A Zia. Breaking VM Isolation-An In-Depth Look into the Cross VM Flush Reload Cache Timing Attack. *International Journal of Computer Science and Network Security (IJCSNS)*, 17(2):181, 2017. URL: `https://researchoutput.csu.edu.au/en/publications/breaking-vm-isolation-an-in-depth-look-into-the-cross-flush-reloa-2`.

**34** Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the Convergence of Adam and Beyond, 2019. `arXiv:1904.09237`.

**35** Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 199–212. Association for Computing Machinery, 2009. `doi:10.1145/1653662.1653687`.

**36** Hong Rong, Huimei Wang, Jian Liu, Xiaochen Zhang, and Ming Xian. WindTalker: An Efficient and Robust Protocol of Cloud Covert Channel Based on Memory Deduplication. In *Proceedings of the 2015 IEEE Fifth International Conference on Big Data and Cloud Computing*, BDCLOUD '15, pages 68–75, USA, 2015. IEEE Computer Society. `doi:10.1109/BDCloud.2015.12`.

**37** Stan Salvador and Philip Chan. Toward Accurate Dynamic Time Warping in Linear Time and Space. *Intell. Data Anal.*, 11(5):561–580, October 2007. `doi:10.5555/1367985.1367993`.

**38** Lukas Sigrist. *Design and Instrumentation of Environment-Powered Systems.* PhD thesis, ETH Zurich, 2020.

**39** Raphael Spreitzer, Simone Griesmayr, Thomas Korak, and Stefan Mangard. Exploiting data-usage statistics for website fingerprinting attacks on android. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '16, pages 49–60. Association for Computing Machinery, 2016. `doi:10.1145/2939918.2939922`.

**40** Shanquan Tian and Jakub Szefer. Temporal Thermal Covert Channels in Cloud FPGAs. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '19, pages 298–303. Association for Computing Machinery, 2019. `doi:10.1145/3289602.3293920`.

**41** Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

**42** Xu, Yunjing and Bailey, Michael and Jahanian, Farnam and Joshi, Kaustubh and Hiltunen, Matti and Schlichting, Richard. An Exploration of L2 Cache Covert Channels in Virtualized Environments. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, pages 29–40. Association for Computing Machinery, 2011. `doi:10.1145/2046660.2046670`.

**43** S. Zander, P. Branch, and G. Armitage. Capacity of Temperature-Based Covert Channels. *Communications Letters, IEEE*, 15(1):82–84, 2011. `doi:10.1109/LCOMM.2010.110310.101334`.

**44** Sebastian Zander and Steven J. Murdoch. An Improved Clock-skew Measurement Technique for Revealing Hidden Services. In *Proceedings of the 17th USENIX Security Symposium*, SS'08, pages 211–226. USENIX Association, 2008. `doi:10.5555/1496711.1496726,`.