Diss. ETH No. 26941

Threat potential assessment of power management related data leaks

A thesis submitted to attain the degree of

Doctor of Sciences of ETH Zurich
(Dr. sc. ETH Zurich)

presented by
PHILIPP MIEDL
Dipl.-Ing.
Master's programme Telematics
at Graz University of Technology

born on 23. 07. 1989
citizen of
Austria

accepted on the recommendation of
Prof. Dr. Lothar Thiele, examiner
Prof. Dr. David Atienza Alonso, co-examiner

2020

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

TIK-SCHRIFTENREIHE NR. 184

Philipp Miedl

Threat potential assessment of power management related data leaks

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A dissertation submitted to
ETH Zurich
for the degree of Doctor of Sciences

DISS. ETH NO. 26941

Prof. Dr. Lothar Thiele, examiner
Prof. Dr. David Atienza Alonso, co-examiner

Examination date: June 26, 2020

DOI 10.3929/ethz-b-000417785

Für Mama, Papa & Opa.

Abstract

Modern computing systems rely heavily on power management to accomplish two main tasks: (i) efficient use of the available energy resources, (ii) prevention of the device from suffering damage by exceeding its physical limitation. The power management system tries to achieve these two goals by enacting policies that, at the same time, aim to reduce the performance penalty experienced by the user. To achieve this, the power management uses the system utilisation and device characteristics, such as thermal behaviour, power dissipation or operating frequency. Therefore, there is a link between the execution of applications and these power management related device characteristics.

Due to the high computing power available, devices are increasingly shared among multiple application domains or multiple users. For example, a smartphone might be used for private and business applications, or multiple users might reside on the same physical server. To guarantee the security of confidential data in such a shared setup, data and application-dependent information must be confined. Confidential information must not be revealed to third parties without the consent of the data owner. Therefore, researchers have increased their efforts to develop security frameworks to enforce this confinement, for example, by using virtualisation techniques. However, data leaks based on shared resources pose a major threat towards such a security framework. As the behaviour of the power management system influences all application or user domains on a device, the power management system is regarded as a potential source for such

data leaks.

While the research community has increased their focus on side and covert channel attacks, several challenges related to these attacks remain. For instance, executing a data leak analysis in a reproducible, comparable and exhaustive fashion requires substantial investments of time and engineering resources. This is due to the nature of data leaks being caused by the interplay of different system components, which makes it difficult to detect, reproduce and analyse them on different devices. Therefore, a methodology is needed to support reproducible, comparable and expressive analysis results and tools that help to reduce the effort needed to execute an exhaustive data leak analysis. Furthermore, while many data leaks have been discovered in recent years, little attention has been given to security implications of the power management in multicore systems.

In this thesis, we attempt to solve these challenges and investigate the threat potential of power management related data leaks in multicore systems. We summarise the main contributions as follows:

- We define a novel methodology to analyse covert channels exhaustively. This methodology helps to derive expressive metrics for assessing the threat potential of covert channels. Furthermore, we are the first to provide a measurement automation toolkit which implements the methodology. Due to its design, this toolkit allows us to apply the methodology to a variety of target platforms.

- We outline a novel procedure to derive upper channel capacity bounds for continuous covert channels. Furthermore, compared to previous work, we improve throughputs of thermal covert channels in multicore systems by applying a more sophisticated communication scheme.

- We are the first to analyse the power covert channel in current multicore systems exhaustively. In addition, we illustrate the derivation of upper channel capacity bounds for such discrete covert channels.
- We present a communication model and provide an in-depth analysis of the frequency covert channel. Moreover, we are the first to use a Recurrent Neural Network (RNN) for symbol decoding in a frequency covert channel setup.
- We establish a novel side channel attack based on system temperatures to extract runtime information from mobile devices. This side channel attack uses Neural Network time-sequence labelling models. Furthermore, we present a method for thermal data augmentation to reduce the necessary measurement effort to generate a suitable training data set.

The presented methods and findings are based on extensive experimental evaluations. We publish the tools used in these experiments and the acquired data along with this thesis, to support comparability and reproducibility of our results.

Kurzfassung

Moderne Rechnersysteme nutzen Energieverwaltungssysteme, um zwei Ziele zu erreichen: (i) effiziente Verwendung der verfügbaren Energieressourcen und (ii) Schutz des Geräts vor Beschädigung durch Überhitzen. Die Energieverwaltungssysteme versuchen diese Ziele zu erreichen, indem sie Strategien anwenden, die gleichzeitig darauf abzielen, die für den Benutzer erkennbaren Leistungseinbußen zu minimieren. Als Basis für die Regelungsentscheidungen von Energieverwaltungssystemen dienen die Auslastung des Geräts sowie Geräteeigenschaften, wie zum Beispiel das Temperaturverhalten, der Energieverbrauch oder die Betriebsfrequenz. Daher gibt es eine Verbindung zwischen den ausführenden Programmen eines Rechnersystems und den Geräteeigenschaften, die vom Energieverwaltungssystem beeinflusst werden.

Aufgrund der hohen vorhanden Rechenressourcen werden Geräte öfters gleichzeitig für verschiedene Aufgaben eingesetzt oder von mehreren Benutzern gleichzeitig verwendet. Zum Beispiel kann ein Mobiltelefon für private und geschäftliche Applikationen genutzt werden, oder mehrere Benutzer arbeiten gleichzeitig auf demselben physischen Rechner. Um die Sicherheit und Vertraulichkeit von Daten in solchen mehrfach genutzten Systemen zu garantieren, müssen Datenzugriffs- und Datenübertragungsbeschränkungen eingehalten werden. Daten und applikationsabhängige Informationen dürfen nicht ohne Zustimmung des Inhabers zu Dritten gelangen. Aus diesem Grund wurden verstärkt Sicherheitssysteme entwickelt, die diese Beschränkungen zum Beispiel durch den Einsatz von Virtualisierung durchsetzen. Eine der größten Bedrohungen für

solche Sicherheitssysteme stellen Datenlecks durch gemeinsam genutzte Ressourcen dar. Da das Verhalten des Energieverwaltungssystems eines Geräts alle darauf laufenden Applikationen beeinflusst, kann man es als ein potentielles Sicherheitsrisiko einstufen.

Obwohl die wissenschaftliche Gemeinschaft ein verstärktes Augenmerk auf Attacken mittels Seitenkanälen (side channels) oder verdeckten Kommunikationskanälen (covert channels) gelegt hat, bleiben einige ungelöste Herausforderungen. Zum Beispiel sind ein hoher Zeitaufwand und der Einsatz von vielen technischen Ressourcen nötig, um ein Datenleck auf eine reproduzierbare, vergleichbare und aussagekräftige Art hin zu analysieren. Dies ist darauf zurückzuführen, dass Datenlecks oft aus dem Zusammenspiel verschiedener Systemkomponenten resultieren und daher auf verschiedenen Geräten schwer zu detektieren, zu reproduzieren und zu analysieren sind. Daher besteht die Notwendigkeit für eine Methodik, welche die Ausführung von reproduzierbaren, vergleichbaren und aussagekräftigen Studien unterstützt, sowie für die notwendigen Werkzeuge, um den Aufwand für solche Studien zu verringern. Ferner wurden in den vergangenen Jahren zwar viele Datenlecks entdeckt, jedoch wurde diesen in Verbindung mit Energieverwaltungssystemen wenig Aufmerksamkeit geschenkt.

In dieser Dissertation versuchen wir diese Herausforderungen zu lösen und untersuchen das Gefahrenpotential von Datenlecks in Mehrkernarchitekturen, welche durch das Energieverwaltungssystem verursacht werden. Die wichtigsten wissenschaftlichen Beiträge dieser Dissertation können wie folgt zusammengefasst werden:

- Definition einer neuartigen Methodik zur eingehenden Analyse von verdeckten Kommunikationskanälen: Diese hilft aussagekräftige Metriken herzuleiten, welche es

erlauben, das Gefahrenpotential dieser verdeckten Kommunikationskanäle abzuschätzen. Außerdem präsentieren wir ein Softwarepaket zur Messautomatisierung, welches die Methodik implementiert. Durch das Design des Softwarepakets ist es möglich, die Methodik auf eine Vielzahl von verschiedenen Geräten anzuwenden.

- Wir zeigen eine neue Prozedur, um obere Kapazitätsgrenzen für kontinuierliche verdeckte Kommunikationskanäle herzuleiten. Ferner erreichen wir mit unserem experimentellen Aufbau von Temperaturkanälen aufgrund der verbesserten Signalkodierung höhere Datenübertragungsraten als frühere Publikationen.

- Wir analysieren erstmals den Leistungskanal in aktuellen Rechnersystemen. Zusätzlich präsentieren wir eine Methode zur Herleitung von oberen Kapazitätsgrenzen für diskrete verdeckte Kommunikationskanäle.

- Wir präsentieren ein Kommunikationsmodell und eine detaillierte Analyse des Frequenzkanals. Im Übrigen zeigen wir erstmals die Nutzung eines Rückgekoppelten Neuronalen Netzwerks (RNN) zur Symboldekodierung bei Datenübertragungen mittels des Frequenzkanals.

- Wir demonstrieren eine neuartige Seitenkanalattacke, basierend auf der Temperatur eines Geräts, mit welcher Laufzeitinformationen von Applikationen aus Mobiltelefonen extrahiert werden können. Diese Seitenkanalattacke nutzt Modelle, basierend auf neuronalen Netzwerken, zur Klassifikation von Zeitsequenzen. Außerdem zeigen wir eine Methode zur Augmentierung von Temperaturdaten, um den Messaufwand zur Bestimmung eines geeigneten Trainingsdatensatzes zu verringern.

Die präsentierten Methoden und Ergebnisse basieren auf der Auswertung von umfangreichen experimentellen Analysen. Wir publizieren die Werkzeuge, welche wir für die Experimente verwendet haben, sowie die generierten Daten gemeinsam mit dieser Dissertation, um die Vergleichbarkeit und Reproduzierbarkeit unserer Studien zu erhöhen.

Acknowledgements

With this chapter of my life coming to an end, I would like to express my gratitude to all who have accompanied me during the last years.

First and foremost, I would like to thank Prof. Lothar Thiele for giving me the opportunity to work in his research group and write my thesis under his supervision. I am very grateful for the confidence put into my work and the high level of freedom I could enjoy. I also want to sincerely thank Prof. David Atienza for agreeing to review my thesis and for being so flexible, as setting up the examination was not a straight forward process in the time of CoVid-19 restrictions.

I want to thank Davide for guiding me during my first steps in the research domain, Rehan for sharing an office for so many years and bouncing ideas every once in a while, Mirko for being my first student to supervise and working with me for more than a year and Bruno, my last student, who was an integral part of getting the ExOT software open sourced. A big thank you to all the other TECies, who made this lab such a great place to work by helping out with experiments or giving feedback to publications and talks, spending lunch time together, having Aperos, hiking and skiing days, going on research retreats, helping me move and then joining the obligatory house warming party. I also would like to thank all of my former students, who have endured me as their supervisor, as well as the administrative staff at TIK who helped with IT and organisational issues. Thanks to my colleagues in the SAFURE project, for the inputs and discussions during the project meetings and to my german and english proof-readers Lore and Sarah, for their help polishing the thesis.

Quiero dar las gracias a Dr. Fernando Sarasa Barrio para ayudarme poder vivir una vida sintiendo sano y con la

posibilidad de hacer tanto deporte como quiero. Eso estuve muy importante para mi, para tener la energía necesaria para finalizar el doctorado.

A big thanks also to my friends, who have been such a great support in the last years. To my Swiss circle, for welcoming me in their lives, for the skiing days, the wine tastings, and all the small and big get together that made life in Switzerland so much more worthwhile! An die Österreicher, die Haie, die BESTies, die Grazer, Wiener, Lambrechter und Oberwölzer, dass ihr mich nicht ganz vergessen habt obwohl ich schon länger weg bin, für all die Besuche und Treffen mit vielen lustigen Stunden zusammen! ¡A mis amigos españoles, para darme la bienvenida y todos las memorias buenas que hemos creado juntos! To my international friends, for the visits in Switzerland and welcoming me when I visited you guys!

Danke an Alex, für das aufrechterhalten des Kontakts und die Möglickeit, dass ich mich auch hin und wieder mal aussudern konnte.

Muchísimas gracias a Maria MC para ampliar mis horizontes, confiar en mi y ayudarme a mover a Suiza para el doctorado. ¡Sin ti nunca hubiera hecho eso!

Ein großes Danke an Manuel und Azra, die bereits in Graz zu einer zweite Familie geworden sind, und auf die ich auch trotz der Distanz immer noch jederzeit zählen kann! Hvala Azra za nacrt ExOT-Logo i korice disertacija!

An meine Familie zuhause in Oberwölz (und Klagenfurt), dass ich immer zu Besuch kommen darf, ihr mir immer helft wenn ich etwas brauche und die schönen Stunden die wir gemeinsam verbringen! Natürlich auch ein Dankeschön an meine Nichten und Neffen, die immer soviel Freude verbreiten wenn ich zu Besuch komme!

Danke an Nina, für deine große Unterstützung während meines Doktorats! Die vielen Diskussionen beim Abendessen

waren immer eine Quelle für neue Ideen und auf unseren gemeinsamen Ausflügen konnte ich immer wieder frische Energie sammeln!

Ein großes Dankeschön an Opa, der mir bei unseren wöchentlichen Skype-Telefonaten all die wichtigen und unwichtigen Ereignisse von zuhause erzählt, damit ich auch weiß was in der Heimat passiert. Und natürlich gilt mein größter Dank Mama und Papa, die mich immer unterstützt haben und mir somit alles ermöglicht haben was ich bisher in meinem Leben erleben durfte!

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644080.

This work was supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0025. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the Swiss Government.

Contents

List of Figures

List of Tables

Acronyms

API Application Program Interface.

AWGN additive white gaussian noise.

bps bits per second.

CDF Cumulative distribution function.

CNN Convolutional-Neural-Network.

COTS Commercial Off-the-Shelf.

CTC Connectionist Temporal Classification.

DTM Dynamic Thermal Management.

DTW Dynamic Time Warping.

DVFS Dynamic Voltage and Frequency Scaling.

ExOT Experiment Orchestration Toolkit.

FFT Fast Fourier Transform.

HMM Hidden Markov Model.

IoT Internet-of-Things.

LLC Last-Level-Cache.

LoRaWAN Long Range Wide Area Network.

LSTM Long Short-Term Memory.

MSR Model Specific Register.

NN Neural Network.

OS Operating System.

PER Packet Error Rate.

PWM Pulse-Width-Modulation.

RNN Recurrent Neural Network.

SNR Signal-to-Noise-Ratio.

SoC System-on-Chip.

TSC Time Stamp Counter.

VM Virtual Machine.

1

Introduction

Besides Moore's law [Moo98], i. e., the fact that the number of transistors in an integrated circuit doubles every two years, microprocessor designers used to rely on another important rule: the Dennard scaling. It was first described by Dennard et al. [Den+74] as follows:

> *The power dissipation of each circuit is reduced by κ^2 due to the reduced voltage and current levels [...] Since the area of a given device or circuit is also reduced by κ^2, the power density remains constant. Thus, even if many more circuits are placed on a given integrated circuit chip, the cooling problem is essentially unchanged.*

However, while Moore's law is still valid today, the Dennard scaling has broken down [Esm+11]. As transistors become smaller, side effects, like leakage currents, start to dominate and cause the power density to increase with reducing transistor area, breaking the scaling effect defined by Dennard et al. [Den+74]. Hence, boosting the performance of microprocessors

by increasing the clock frequency is not possible anymore due to thermal issues, i. e., because there are more transistors on a smaller area, the heat generated by switching these transistors cannot be reasonably dissipated anymore, resulting in the overheating of the device. Therefore, microprocessor manufacturers have shifted towards multiprocessor architectures to achieve higher computing throughput and given increased attention to thermal and power management systems to allow a demand-based use of the available thermal budget. Here, the thermal budget describes the amount of thermal energy that can be dissipated by the microprocessor within a certain time interval.

At the same time, the surge of battery powered mobile systems has further fuelled the development of sophisticated thermal and power management systems to maximise the battery life of these mobile devices. Thus, today, almost all computing devices take advantage of thermal and power management systems to reduce the power consumption and protect the device from thermal damage, while trying to minimise the performance impact noticeable by the user.

With the increasing amount of processing power provided to users by multicore systems, many of those systems are currently shared to increase the utilisation of the available computing resource. For example, a physical server might be shared among multiple users, or a hand-held device might be used for applications from different domains, e. g., private and business. However, such a shared setup can cause security issues, as Lampson [Lam73] already illustrated when presenting the *confinement problem*. Lampson stated that data could be regarded as secure as long as the system can guarantee that no information is passed, or leaked, to a third party without consent from the data owner. Therefore, system designers rely heavily on the security paradigm of permission separation and application isolation to keep applications confined and provide

a certain level of security on shared devices where third party users or applications are present on the same piece of silicon.

A popular technique to implement application separation is virtualisation, such as application sandboxing or virtual machines. While these techniques may isolate applications on the software level, there is still a chance for breaking the application confinement as long as the hardware is shared. For example, data may be leaked from a confined execution domain by observing hardware characteristics that are dependent on the utilisation pattern of the hardware, i. e., the thermal behaviour, the power dissipation and other aspects controlled by the power management systems.

Detecting such power management related data leaks is difficult, as they are often caused by the interplay of multiple system components. In addition, the mitigation of such data leaks is also a non-trivial task. This is due to the fact that mitigation may either require changes to the hardware setup or restrict the access to hardware status information, like the current device temperature, which is vital for the power management system. Therefore, if the security demands of the system allow it, the more suitable solution might be to tolerate such potential data leaks while closely monitoring them, rather than fully mitigating them. However, a confident decision on the threat potential of a data leak and whether it is tolerable in a system can only be based on an extensive theoretical and experimental analysis.

In this thesis, we outline how to conduct such an extensive theoretical and experimental analysis of a certain class of data leaks. We present a methodology that supports comparability, reproducibility and expressiveness of analysis results and introduce a software toolkit that implements this methodology. Furthermore, we illustrate how to execute such an extensive analysis for different power management related data leaks

in multicore systems and assess their threat potential using models, capacity bounds and experimental data. Moreover, we present a novel data leak based on thermal sensor readings, which allows sensor readings to leak application runtime information from current smartphones.

1.1 Background and related work[1]

The study of security issues related to privilege-separation and isolation in computing systems is a well-defined area of research. Already in 1973, Lampson [Lam73] analysed this *confinement problem* and noted the possibility of exploiting *covert* or *side channels*, i. e., observing system properties not originally intended for communication, in order to leak restricted data. The term *covert channel* is used when two colluding applications actively share information, as opposed to the term *side channel*, used when an attacker observes an unaware system with the aim of inferring sensitive information, e. g., a cryptographic key [HS14].

Covert channels can broadly be classified as *storage* or *timing* channels. In storage channels, the sending application directly or indirectly writes to a shared resource, which the receiving application reads. In timing channels, the sending application exploits the ability to influence timing properties of the system that the receiving application can observe [US 85; MP14]. In this thesis, we study both kinds of covert channels. In chapter 3 and 4, we analyse storage covert channels, while in chapter 2 and 5, we take a closer look on timing covert channels.

Moreover, in chapter 6, we present a detailed study of a side

[1] This section is based on the related work presented in [BMT16], [MT18], [Mie+18], [MKT20c], and [MAT20].

channel. In particular, we will analyse the possibility of leaking application runtime by observing the thermal behaviour of a device.

While the different types of data leaks are well defined, few previous works have discussed how to classify data leaks based on their threat potential. The U. S. Department of Defence reported in its 1985 *Orange Book* [US 85] that trusted computing environments must have "*the capability to audit the use of covert channel mechanisms with bandwidths that may exceed a rate of one (1) bit in ten (10) seconds*". However, such a classification using absolute measures might quickly become outdated, and the critical capacity numbers could look very different if such an assessment would be done today. Moskowitz and Kang [MK94] took a different approach when defining the *small message criterion*. They concluded that bandwidth and capacity alone are insufficient metrics to quantify the threat potential of covert channels, arguing that if the amount of sensitive information is very small, the capacity of the leakage channel is not an accurate measure to define the threat potential. Consequently, the threat potential of a data leak is always dependent on the scenario, as even a covert channel with very low capacity poses a substantial threat if the leakage of small amounts of highly sensitive data can compromise the system. To support such a flexible classification of the data leak threat potential, we present a methodology to evaluate data leaks and determine the capacity of various covert channels in the chapters 2, 3, 4 and 5.

1.1.1 Microarchitectural security issues

Complex processor architectures are likely to expose properties that can be exploited to create covert or side channels to leak information across security domains [WL06a]. In particular, shared microarchitectural resources are a major target for this

purpose [FPK17].

Many data leaks that have been discovered and analysed in recent years rely on the fact that parts of the cache hierarchy are shared among multiple processors [Lip+16]. For example, the well known Spectre [Koc+18] and Meltdown [Lip+18] data leaks take advantage of the fact that, due to hyper-threading in Intel processors, multiple logical cores share the same cache. Similarly, Rong et al. [Ron+15] 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 [Mau+17; PZ17] and specifically targeted at Intel SGX based systems [Göt+17]. Other previous work showed that by exploiting shared caches it is possible to disclose the existence of other virtual environments via a side channel attack [Suz+11], or that it is possible to establish covert channels between isolated applications [Xu 11; WXW15].

Branch predictors have also been exploited as possible sources for data leaks in modern computing systems [EPA16; EPA15]. In their works, the authors present how an application can manipulate the shared branch predictor table such that it is possible to establish a robust, noise-free, high-capacity covert channel.

Similar to these previous works, in this thesis, we will also take advantage of microarchitectural features to establish data leaks. In particular, in chapter 2, we will use the example of cache-based covert channels to illustrate our model to analyse covert channels in detail. However, the covert channels presented in chapters 3, 4 and 5 only take advantage of microarchitectural features related to the power management of the system.

Previous work has also aimed at modelling microarchitectural covert channels. Hunger et al. [Hun+15] introduced

a mathematical abstraction called the *bucket model*, which is capable of capturing the common characteristics of different microarchitectural side and covert channels and deriving their capacities. Contrary to the bucket model introduced by Hunger et al. [Hun+15], we will base our modelling methods and capacity derivations on well-known and generally applicable techniques from the communication and signal processing domain.

1.1.2 Power management related attacks

Recent work has shown that power management features can be used to attack devices, for example, to mount a denial of service attack by creating a hot spot on the silicon to trigger Dynamic Thermal Management (DTM) and induce performance throttling [Has+05]. In contrast, in this thesis, we will only present attacks that allow us to leak data from systems using power management related characteristics in general-purpose hardware.

Another target for the realisation of side channels is to tamper with the operating conditions of a chip. For example, Hutter and Schmidt [HS14] demonstrated that a temperature side channel able to retrieve the private key from an RSA implementation on an AVR microcontroller. They decapsulated the chip to measure the temperature directly on the surface of the silicon substrate and operated the device at 150℃, beyond its specified temperature range. They found that, under these conditions, the device leaked the Hamming weight of the processed data via the temperature side channel. They exploited this property to retrieve the private key by correlating the temperature, execution and power traces of the chip for several runs. Unlike Hutter and Schmidt [HS14], in this thesis, we will not tamper with the operating conditions of the device to establish a data leak.

Similar to the covert channels we present in chapters 3, 4 and 5, other work has illustrated how to encode information in the fan speed of a device [Bro+09a]. In contrast, rather than using characteristics from processor peripherals, we will only rely on measurements taken from the processor itself.

1.1.3 Data leaks based on thermal status information

In addition to temperature-related effects, temperature itself can serve as a medium to leak data. Islam, Ren and Wierman [IRW17] presented a thermal side channel attack that allowed 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 [MM07; Bro+09b; INK11] or over time to the next scheduled application [TS19].

Guri et al. [Gur+15] studied an indirect variant of the thermal covert channel to attack air-gapped systems. They showed that communication is possible between two nearby, air-gapped desktops by using the available temperature sensors: the sending application runs on one desktop and controls load; the receiving application runs on the other desktop and observes temperature variations caused by the heat coming from the other desktop. Work in this direction showed that communication between the isolated components is possible through a covert channel based on thermal readings, similar to the covert channel we study in chapter 3.

In previous work more closely related to our research, Masti et al. [Mas+15] presented an initial study of the thermal covert channels on multicore processors. They showed experiments that achieved a transmission rate of up to 1.33 bits per second (bps) with an error rate of 11% for a covert channel

between two neighbouring cores on an Intel Xeon-based server. Their work only looked at the covert channel from an empirical perspective. We present a new methodology that uses both experimental results and theoretical analysis to characterise covert channels in chapter 2. Moreover, in chapter 3 we analyse the family of thermal covert channels on modern multicores in detail using the methodology presented in chapter 2, and present an improved transmission scheme, in comparison to Masti et al. [Mas+15].

In contrast to the thermal status information based data leaks presented in this section, in chapter 6, we will illustrate how to use thermal status information to infer information on the observed system. This means that there are no colluding applications which actively leak data.

1.1.4 Data leaks using power measurements

Recent work has also presented several data leaks based on measuring the changes in the power dissipation of devices. For instance, Michalevsky et al. [Mic+15] presented a method that allowed location tracking of a mobile device based on the power dissipation. The authors generated a power-map of an area using power fingerprints for every location. Using this map, the authors were able to reconstruct the movement trajectory of a mobile device by analysing the power trace.

Spolaor et al. [Spo+17] showed that it was possible to extract data from a charging phone by modulating the amount of power taken in via the charger by changing the utilisation of the mobile phone's processor. The authors manipulated the charger to measure the current input to the phone and could reconstruct the leaked data stream by analysing the recorded power trace.

Unlike Michalevsky et al. [Mic+15], in chapter 4, we will use active data transmission based on the power dissipation

of a device. Furthermore, contrary to Spolaor et al. [Spo+17], we will not use any external devices to establish the covert channel but only rely on internal device measurements. We present a method to transfer data directly from one application to another within the device, which can then leak the data later via conventional communication interfaces.

1.1.5 Data leaks based on timing measurements

Many previous studies have analysed covert timing channels, which are based on injecting and monitoring timing variations on certain events. For example, Yue et al. [Yue+14] presented a covert channel where information was encoded into the inter-packet spacing of the network traffic by directly controlling the operating frequency of a device. Similarly, we will establish a covert channel by taking advantage of the timing variation of specific operations. However, we do not control the operating frequency through operations that require elevated privilege levels, nor do we decode information by reading system timestamps. Furthermore, while the above-mentioned covert channel is established within a communication network, the timing covert channels we analyse in chapter 2 and 5 are established between different applications within the same device.

A well-studied timing channel exploits the local clock skew introduced by temperature variations [Mur06; ZM08; Ris+09]. If a sending application can trigger temperature variations on a victim host, it can induce a skew in the local clock. The receiving application can observe the skew by looking at timestamps and comparing to a reference clock. This channel was exploited to reveal hidden services, for example, services running under the Tor network or to infer the topology of a public cloud infrastructure. Zander, Branch and Armitage [ZBA11] estimated the capacity of this timing channel to

be up to 20.5 bits per hour. In contrast, the timing channels we analyse in the chapters 2 and 5 do not depend on the temperature-induced clock skew, but on the execution of operations due to changes of the cache status of the data (chapter 2) or the operating frequency of the system (chapter 5).

Wang and Lee [WL06b] presented a covert timing channel which allows the communication between two otherwise isolated applications. The sending application blocks a functional unit for a certain amount of time to cause a delay in the computation of the receiving application. This is interpreted as sending a 1; to send a 0, the sending application stays idle. Similarly, Marforio et al. [Mar+12] analysed a frequency-based covert channel in which the sending application utilises the processor or stays idle, causing a change of the frequency of the core. The receiving application detects frequency changes by reading the core frequency from system files and can interpret whether a 0 or a 1 is sent. In chapter 5 we study a covert channel similar to the timing channel presented by Wang and Lee [WL06b] and the frequency covert channel presented by Marforio et al. [Mar+12].

As opposed to the covert channel used by Wang and Lee [WL06b], in chapter 5, we place the sending application and the receiving application on two different cores allowing cross core communication. Moreover, Wang and Lee [WL06b] could control the timing changes directly by blocking a functional unit. In contrast, we indirectly control the timing changes by changing the utilisation of the core so that the system scales the operating frequency. As the system may not behave as desired, our implementation takes advantage of a feedback loop that allows the sender to react to the system behaviour.

Unlike Marforio et al. [Mar+12], the frequency covert channel implementation we present in chapter 5 uses timing measurements to determine the operating frequency. We show

that such a method poses a higher threat, as it is harder to mitigate than using system readings for the implementation of the frequency covert channel, which can be easily blocked by changing the access permission to these system files. Further, as the evaluation presented by Marforio et al. [Mar+12] is purely experimental, we provide the missing theoretical analysis by presenting a capacity bound that gives an initial estimate of the threat level of such a channel.

Therefore, we can state that our work in chapter 5 presents an exhaustive analysis of a new covert channel implementation which combines and extends the two previously presented timing and frequency covert channels.

1.1.6 Machine learning in security applications

Fuelled by the rapid advances in the machine learning domain, techniques from this field 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 [PZ13]. Buczak and Guven [BG16] 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 a 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, in chapters 5 and 6, we will use machine learning from the perspective of an adversary. Furthermore, in chapter 6 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.

1.2 Challenges of data leak threat potential assessment

Despite the research efforts in the field of data leak evaluation, there remain a few challenges which have not been fully addressed by the related work presented in section 1.1. In particular, there are still challenges related to the confident assessment of the threat potential of data leaks. In this section, we give an overview of the main challenges that we focus on in this thesis.

Handling system dependencies. Data leaks are often the result of unintended side effects when using a computing system or side effects caused by the interaction of different components of the system. Therefore, evaluating data leaks requires in-depth knowledge of the analysed system and its usage. Furthermore, data leaks can be highly system-specific, and while data leaks may occur on different systems, evaluating the same data leak on different systems might not be possible using the same techniques. For example, different microprocessor architectures might not offer the same software interface to read hardware sensors, provide different sensor readings, or the interplay of different Operating Systems (OSs) with the same hardware may change the characteristics of a data leak.

Determining comparable and expressive metrics. The decision on whether a data leak can be tolerated or not

is always subject to the application scenario of the device. Therefore, an assessment of the data leak threat potential is necessary to be able to make a confident decision on a case-to-case basis. Such a threat potential assessment requires an exhaustive analysis of the data leak to obtain reproducible, comparable and expressive data leak metrics. However, due to the complexity of data leaks, such an exhaustive evaluation could be very resource-intensive.

Presenting a proof-of-concept. To verify whether a data leak poses a threat, it is necessary to present a proof-of-concept and underlay the theoretical analysis with experimental evidence. Yet, such a proof-of-concept potentially requires the use of novel technologies or techniques from different areas of research due to the complexity of data leaks. In addition, to present and evaluate a proof-of-concept implementation, large amounts of data may be necessary. This could require a complex measurement setup, or techniques for data generation if a measurement campaign is not feasible.

Coping with high analysis effort. All of the above-mentioned challenges contribute to one additional important challenge of data leak threat potential assessment: the analysis effort. In this thesis, we relate this challenge to the required time and engineering effort to execute an analysis, as executing a reproducible and exhaustive analysis to determine comparable and expressive results, as well as presenting a proof-of-concept, may involve a large investment in time. Furthermore, evaluations may not be easily portable to different systems, which can increase the necessary engineering effort for an evaluation of the same data leak on multiple systems.

1.3 Aims of this thesis

With this work, we aim to defend the following two theses:

> *To confidentially assess the threat potential of data leaks, it is necessary to conduct an exhaustive analysis that yields comparable, reproducible and expressive results.*

and

> *The dynamic power management of multicore systems can be misused to leak data. Therefore, it is necessary to consider the interplay of the power management system and executed applications, as well as the availability of power management related sensor readings, when building a security and privacy framework.*

1.4 Thesis outline and contributions

In this thesis, we try to address the challenges encountered when assessing the threat potential of power management related data leaks, as presented in section 1.2. An overview of the thesis contributions is illustrated in Figure 1.1, and in the remainder of this section, we will present the contributions of each individual chapter in detail.

Chapter 2: A holistic approach to data leak threat potential assessment. As outlined in section 1.2, we need to determine comparable, reproducible and expressive results based on an exhaustive theoretical and experimental data leak analysis to assess the threat potential of a data leak. However, the amount of effort needed for such an exhaustive analysis can be very high. Therefore, in this chapter, we propose a methodology to analyse data leaks exhaustively and describe the steps to execute the methodology in detail. Furthermore,

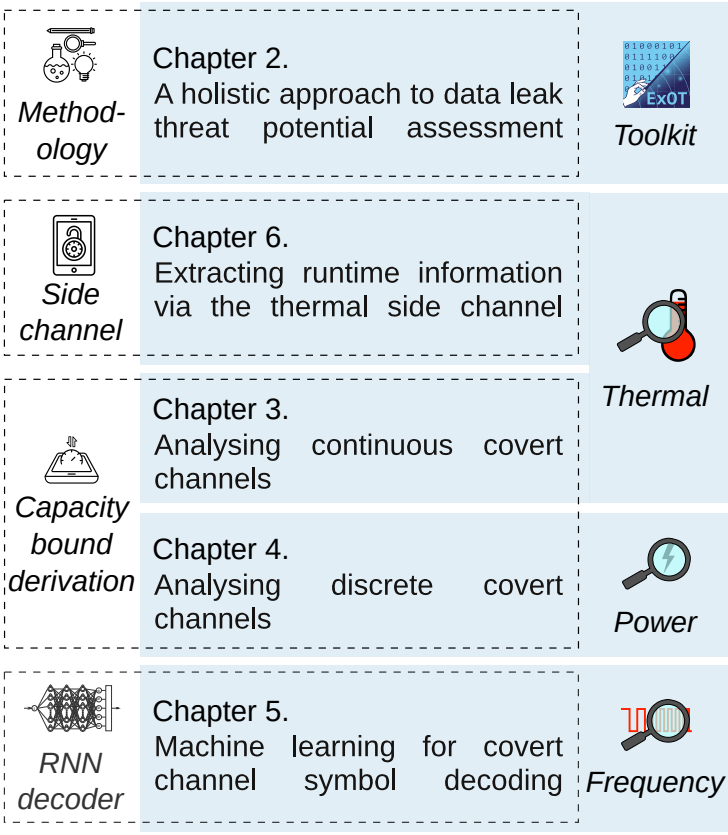


Figure 1.1: We present the methodology to assess the threat potential of data leaks and the toolkit that implements it in chapter 2. In chapter 3 and 4, we outline methods to derive capacity bounds for covert channels and present analyses of the thermal and the power covert channel. We apply machine learning techniques in chapters 5 and 6, proposing a robust signal decoder to cope with system dependencies of covert channels and introducing the novel thermal side channel attack.

we introduce a toolkit which implements the methodology and helps to decrease the engineering effort needed for an exhaustive data leak analysis and illustrate its usability by presenting an evaluation of known cache covert channels. We summarise the contributions in chapter 2 as follows:

- To the best of our knowledge, we are the first to propose a widely-applicable methodology for covert channel analysis. The methodology helps researchers to define models, metrics and experiments for a data leak analyses. This supports the repeatability, comparability and expressiveness of the analysis results.
- In addition, we present the supporting *Experiment Orchestration Toolkit (ExOT)*. Experiment Orchestration Toolkit (ExOT) implements and aims at reducing the engineering effort needed to execute the proposed methodology. ExOT is designed to be easily extended or reconfigured, such that a variety of different classes of experiments can be executed.

Chapter 3: Analysing continuous covert channels. In this chapter, we present a method to determine the capacity bound for continuous covert channels. The capacity bound reports the theoretical maximum information throughput of a channel. As the capacity bound is independent of implementation artefacts, it is considered to be a comparable and expressive metric for a data leak towards its threat potential. Furthermore, we present an exhaustive analysis of the thermal covert channel in multicore systems. Chapter 3 tackles two main challenges:

- Estimating the capacity (under controlled but realistic conditions) of the thermal covert channel to determine a comparable and expressive metric.

- Finding a communication scheme that achieves throughput results close to the channel capacity to present a proof-of-concept for the thermal covert channel.

Chapter 4: Analysing discrete covert channels. In chapter 4, we illustrate how to apply the methodology presented in chapter 2 to another class of covert channels: discrete covert channels. This shows that our toolkit and the data leak analysis methodology are applicable to a broad range of covert channels. The main contributions in chapter 4 are:

- A generally-applicable method to model and derive a tight channel capacity bound for discrete covert channels. The capacity bounds help to estimate the security threat caused by covert channels.
- We present the novel power covert channel and provide models and capacity bounds for this data leak.
- To the best of our knowledge, we are the first to show an implementation of a communication scheme that proves the functionality of the power covert channel on Intel-based platforms.

Chapter 5: Machine learning for covert channel symbol decoding. In chapter 5, we present an in-depth analysis of the frequency covert channel and give a detailed overview of the system-dependent behaviour of this data leak and how to compensate for them in an implementation. The contributions in chapter 5 are:

- We are the first to apply a formal communication model and derive an upper bound on the capacity of the frequency covert channel. Furthermore, we derive a robust communication scheme based on the formal communication model of the frequency covert channel.

- In contrast to previous work, our implementation of the frequency covert channel does not require elevated privileges for system file access. It allows the source and the sink application to communicate if they run on different cores.
- To the best of our knowledge, we are the first to present the usage of a Recurrent Neural Network (RNN) for signal decoding in a covert channel scenario.
- We present extensive experimental evidence to evaluate the feasibility of the frequency covert channel under realistic conditions for two platforms, representative for laptops and hand-held devices. Furthermore, we use our extensive experimental evaluation to provide hints for possible mitigation strategies.

Chapter 6: Extracting runtime information via the thermal side channel. Based on the findings in chapter 3, we present an implementation of the novel thermal side channel attack. This attack allows the runtime information of applications to be leaked by observing the thermal behaviour of a mobile device and is based on the use of sequence labelling models. We summarise the contributions in chapter 6 as follows:

- We present a novel side channel attack that uses thermal data collected from mobile devices to determine patterns of application usage.
- We propose a data augmentation scheme that allows to generate a training data set without the need for an extensive measurement campaign.
- To the best of our knowledge, we are the first to apply machine learning techniques from the time-

series processing domain to determine an application execution sequence.

- We present an extensive experimental evaluation of the thermal side channel based on real user interactions with the device, using laboratory and real-world data.

2

A holistic approach to data leak threat potential assessment

Data leak analyses presented in literature often lack formal methods and metrics or are not exhaustive. Due to these shortcomings, a confident threat potential assessment based on these analysis results is not possible. In this chapter, we address these issues by proposing a holistic approach for data leak analysis.

We present a novel methodology to improve *reproducibility*, *comparability* and *expressiveness* of data leak analyses and introduce the Experiment Orchestration Toolkit (ExOT). ExOT implements our methodology and aims to reduce the engineering overhead required for an exhaustive data leak analysis. Moreover, ExOT allows the application of our methodology to

This chapter is based on [MKT20c], [Klo18] and [Klo19].

Figure 2.1: In chapter 2, we present a novel methodology to improve reproducibility, comparability and expressiveness of data leak analyses. Furthermore, we introduce the Experiment Orchestration Toolkit (ExOT), which implements our methodology.

a variety of platforms and data leaks. Therefore, we use ExOT for all experimental evaluations presented in the remainder of this thesis.

Figure 2.2: A data leak where the source emits information via the channel, received by the sink and forwarded to an adversary. The hidden data transfer is either intentional (covert channel), or unintentional (side channel).

2.1 Introduction

As computing devices are getting more powerful, they are often intended for shared use to utilise the available resources fully. For example, multiple users have access to the same cloud computing infrastructure. Similarly, mobile devices are used for multiple application domains with different security clearances, such as business and private applications. Both cloud and mobile computing systems must prevent data leaks from one user or one application domain to another. Such a data leak may exist in the form of a covert or side channel – two closely related concepts. This relation is highlighted by Ristenpart et al. [Ris+09], stating that "Covert channels provide evidence that exploitable side channels may exist.". In other words, covert channels help to assess the extent to which side channel attacks might be feasible. This is important, as quantifiable metrics often cannot be determined for side channels, but for covert channels, this may be possible (see chapters 3 and 4).

We base our analysis of covert channels on the setup illustrated in Figure 2.2, which is widely used in literature. The main components are (i) the *source* application with access to confidential information, (ii) the *channel*, and (iii) the *sink* application which receives and forwards data to an adversary.

While the system in Figure 2.2 looks rather simple, an exhaustive analysis can be very costly in terms of time and engineering effort. Hence, an exhaustive analysis is not common practice. Examples for limited experimental evidence can be found in recent literature. An example of a non-exhaustive analysis is cache covert channels. Gruss et al. [Gru+16a] stated that the comparison of previously presented throughputs of different cache covert channels was not possible. The authors pointed out that all experiments had to be repeated, due to the differences in the analyses and the fact that the reported metrics where implementation dependent. This might require considerable engineering effort and highlights the importance of analysis methodologies that describe how to derive comparable metrics for data leaks, rather than implementations. These examples indicate that there is a need for a well-defined *methodology* for the analysis of covert channels.

Such a methodology would allow a confident assessment of the threat potential of a covert channel if it (i) is generally applicable to a large class of covert channels, (ii) helps to define models, metrics and experiments, as well as (iii) support the *reproducibility*, *comparability* and *expressiveness* of results. Based on the international vocabulary of metrology [20012], we define reproducibility, comparability and expressiveness as follows.

Reproducibility. An analysis is reproducible if different researchers can repeat it based on the description of the original work and derive the same conclusions.

Comparability. Comparability of different data leak analyses is assured if their results are reported using the same metrics. For example, the threat potential of data leaks cannot be compared if one is quantified using throughputs and another using capacity bounds.

Expressiveness. The expressiveness of metrics describes qualitatively how well they characterise relevant properties of the evaluated system. For example, an implementation might not reveal the full potential of a data leak. Therefore, throughput and the corresponding error rate are only expressive towards an implementation, rather than a data leak. In contrast, the channel capacity bound reports the maximum possible throughput under ideal conditions. Hence, the capacity bound is expressive towards the threat potential of a data leak.

Contributions. To the best of our knowledge, we are the first to propose a widely-applicable methodology for covert channel analysis. The methodology helps researchers to define models, metrics and experiments for data leak analyses. This supports the repeatability, comparability and expressiveness of the analysis results. In addition, we present the supporting *Experiment Orchestration Toolkit (ExOT)*. Experiment Orchestration Toolkit (ExOT) implements and aims at reducing the engineering effort needed to execute the proposed methodology. ExOT is designed to be easily extended or reconfigured so that a variety of classes of experiments can be executed. Thus, ExOT substantially exceeds the functionality of previously presented work like the Mastik toolkit [Yar16] or libflush [Lip+16; Gru+16b].

2.2 Revisiting known data leak evaluations

In this section, we review a selection of cache covert channels that have been presented in recent years. While the selection is not exhaustive, its purpose is to show the usability of the proposed methodology and ExOT.

The cache covert channels we consider in this chapter all operate on similar principles, which we briefly describe in the following paragraphs.

Memory de-duplication. In order to optimise the usage of the available memory, pages with identical content are often shared among processes or even Kernel Virtual Machine (KVM) subsystems [AEW09; YF14]. For example, two processes load the same read-only library objects. While the library objects will be mapped to the private virtual address space of each process, these private virtual addresses will point to the same physical memory location.

Inclusive Last-Level-Caches (LLCs). Many LLCs in modern computing systems are implemented with an inclusive policy. This means that the LLC always contains a copy of all the cached data in the lower-level caches. If data is evicted from the LLC, back invalidation will be performed on all lower-level caches. This means that the data is also evicted from the lower cache levels. Furthermore, many LLCs are indexed with physical addresses. Therefore, a shared library will include its addresses in the LLC when loaded and accessed.

Unprivileged cache flushing operations. Calling cache flushing operations will evict a cache line from all cache levels. These operations might even broadcast the invalidation "throughout the cache coherency domain" [Int18, p. 11-12]. As these cache flushing operations are unprivileged in both x86_64 (`cflush` [Int18]) and 64-bit Arm architectures (data cache maintenance instructions [Arm19, p. 2353]), they can easily be used for malicious purposes.

Cache-status dependent execution time. The execution time of specific operations will vary, depending on the cache status of the data, i. e., in which level of the memory hierarchy the data resides.

Covert Channel	Empirical Throughput / Error Rate					
	x86_64			ARM		
Flush+Flush	496 KB/s	/	0.84%	178.3 Kbps	/	0.48%
Flush+Reload	298 KB/s	/	0.00%	1.14 Mbps	/	1.10%
Flush+Prefetch	146 KB/s	/	$< 1\%$	N/A		

Table 2.1: Throughputs of known covert channels reported by Gruss et al. [Gru+16a]. These values are not expressive towards the threat potential of the analysed data leak, as they are highly implementation and setup dependent.

We consider three different cache timing covert channels. These cache covert channels take advantage of memory de-duplication, inclusive LLCs and unprivileged flushing operations to manipulate the execution time of certain operations in other processes or Virtual Machines (VMs).

1. The *Flush+Flush* [Gru+16a; Lip+16] channel relies on the execution time of the flush operation. This operation tends to take more time to execute if the data has been cached.

2. The *Flush+Reload* [YF14; Gru+16a; Lip+16] channel uses the time needed to execute the reload command. The reload operation will have different latencies depending on whether data has already been cached or not.

3. The *Flush+Prefetch* [Gru+16b] channel exploits the timing of the prefetch instruction. The prefetch instruction is executed faster if data is already present in the cache, i. e., is already cached.

None of the references mentioned present capacity bounds for their covert channels, while all provided throughput experiments. In Table 2.1, we report the results obtained from the re-implementation of the "Flush+" channels by Gruss et al. [Gru+16a]. The throughputs were all determined using different implementations and report different error rate metrics. Hence, they cannot be used as a measure of the general threat potential of this class of data leaks. The throughputs are only an expressive metric for a specific implementation. In the remainder of this chapter, we address the issue of expressiveness of analysis results.

Figure 2.3: Main steps of the proposed methodology for covert channel analysis.

2.3 A novel covert channel analysis methodology

Figure 2.3 illustrates the four main steps of our proposed covert channel analysis methodology. In this section, we describe these four steps and how our toolkit implements them.

2.3.1 Modelling

During the modelling phase, the covert channel needs to be formally described. A model is a necessary prerequisite for comparability and is beneficial when defining the metrics for evaluation. For example, in the case of covert channels, the formal models enable us to derive an upper capacity bound.

This phase cannot be automated, as it relies on the expertise of the researcher. However, the framework provides blueprints for two ready-to-use models, which can be applied and determine the experimentation flow. The formal models that are implemented and described in ExOT are

1. time-continuous and value-continuous, suited to describing, for example, the thermal covert channel, see chapter 3, as well as

2. time-discrete and value-discrete that can be used to model, for example, the power covert channel, see chapter 4.

2.3.2 Capacity bound derivation

The capacity bound is a metric that allows us to estimate the threat potential of a covert channel. A capacity bound is independent from the implementation artefacts, as it describes the maximum capacity of a channel achievable under ideal conditions. Hence, capacity bounds are expressive regarding the general threat potential of a data leak, rather than a specific implementation. Examples for capacity bound derivations can be found in chapters 3 and 4.

Depending on the previously established model, ExOT provides an experimentation scaffold to determine the parameters necessary to derive the capacity bounds. This includes the experiment definition and configuration, as well as the experiment generation, execution and analysis flow. In addition, ExOT controls environmental factors if the hardware setup allows it. Furthermore, ExOT provides basic building blocks for the applications generating the channel input and recording the channel output. These building blocks can be used on many different platforms, as ExOT also includes a cross-compilation suite and allows for the integration with other tool chains.

2.3.3 Experimental channel evaluation

Based on the experimental evidence provided by the evaluation, the previously established channel model and the capacity bounds are validated. An expressive metric for the threat potential of a specific implementation is provided in the form of throughputs.

The experiments are conducted under well-defined laboratory conditions with a prescribed software flow to support reproducibility. Their purpose is to understand the capabilities of the channel and the effect of external influences. The

experiment setup ensures reproducibility as the source and the sink application are well-synchronised, and external influences are controlled. In this phase, first, a simple stream of random bits is transmitted from one application to another. Simple source coding (conversion of bits to symbols) and line coding (conversion of symbols to a channel input trace) techniques are used to establish a rudimentary communication channel with minimal engineering effort. Using this configuration, possible throughputs with corresponding error rates are determined, and additional experiments can be performed to understand the covert channel better. For example, the effect of external influences or additional noise, e. g., generated by other applications, can be quantified. This approach yields insights that are useful for the development of mitigation techniques.

ExOT maintains the prescribed software flow and handles source and sink application synchronisation. Furthermore, ExOT offers a variety of source and line coding options that can be configured and applied to a randomly generated bit stream. Moreover, ExOT provides an interface to control external influences and additional applications to investigate external influences.

2.3.4 Deployment test

Using the knowledge gained in the previous steps, it is possible to evaluate the covert channel in a real-world scenario. An example of a deployment test was presented by Maurice et al. [Mau+17]. In contrast to the experimental channel evaluation phase, in the deployment test phase, the source and sink applications have to operate fully autonomously. This requires that the source and sink application also implement measures that compensate for external influences, like noise or interference. Furthermore, the application might need to implement sophisticated protocols with synchronisation methods,

packeting, bi-directional communication, error detection or error correction. Therefore, such real-world implementations can be designed in many different ways, and the throughput will often depend heavily on the invested engineering effort. The main goal of the deployment test is to show that an attack can be deployed outside of a laboratory setup. In addition to the previously determined metrics, the deployment tests can yield additional measures, like the attack footprint or the necessary implementation effort. ExOT provides application building blocks, and experimentation execution flows to reduce the engineering effort in this phase of the analysis.

Review. When applied to analyse a specific data leak, our methodology helps to determine metrics and experiments for an exhaustive evaluation. This supports reproducibility, comparability and expressiveness of results. We present examples for such applications in chapter 3 and 4. Moreover, these examples illustrate that the methodology is generally applicable to different kinds of covert channels. Therefore, our proposed methodology meets the requirements defined in section 2.1.

2.4 The Experiment Orchestration Toolkit (ExOT)

In this section, we present an overview and implementation details of our *Experiment Orchestration Toolkit ExOT*. Figure 2.4 shows the structure of ExOT and the interaction of the different components of the setup.

ExOT supports repeatability of experiments by keeping track of all relevant experiment information in a single location. The configuration and environment descriptor files written in

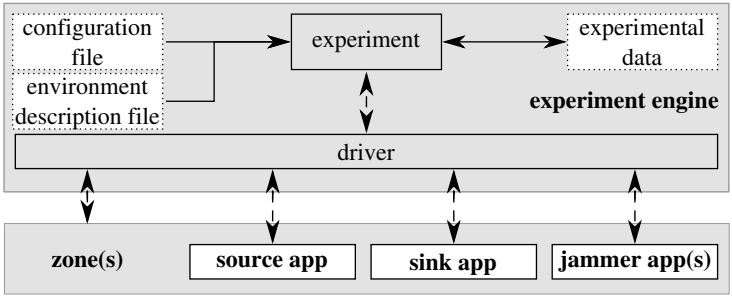


Figure 2.4: Experimental setup using the Experiment Orchestration Toolkit (ExOT). The driver is the interface to the experiment environment consisting of different zone(s) and the applications.

TOML[1] capture relevant experiment parameters. Moreover, the complete flow, including experiment generation, experiment execution and analysis of the experimental data, can be written in one Python script, which makes experiments easier to version and maintain.

The experiment environment consists of at least one environment zone, a source and sink application pair and optionally jammer applications. Jammers can be used to simulate disruptive influences on the covert channel, which is useful to either (i) understand the influence of external factors on the covert channel, or (ii) evaluate possible mitigation strategies. Any application can be instrumented as a jammer application, either custom-made applications with similar functionality as the source application or applications that can be installed on the device, like the ffmpeg video conversion tool. All applications are mapped to a zone, whereas one zone defines a certain configuration. The environment zones define

[1] *Tom's Obvious, Minimal Language* by Tom Preston-Werner, Pradyun Gedam, et al.: `github.com/toml-lang/toml`

the runtime environment of the applications. This can be, for example, the host Operating System (OS), a virtualised environment on a device or a node in a network.

The experiment engine will setup the environment by configuring the zones(s) and applications as well as copying the necessary data. It controls the experiment execution, fetches the data and cleans up the environment after the experiment execution has finished. The experiment engine also offers a variety of debug outputs in the form of log-messages during execution or plots for data preview during analysis.

2.4.1 Creating sending and receiving applications

In order to identify a possible covert channel, colluding sending and receiving applications are needed. While in the literature, the two colluding applications are usually called *trojan* and *spy*, we refer to them as *source* (*src*) and *sink* (*snk*). We choose this terminology due to the broader applicability of ExOT, which we will outline in subsection 2.4.3. In this section, we describe how ExOT can help to reduce the effort to implement source, sink and custom jammer applications.

We implemented the application library using C++17, taking advantage of modern language features. This includes the use of generic programming, compile-time code generation, templated design and inheritance without complex class hierarchies. These development paradigms ensure that the code base is extendable without too much code duplication. The library provides basic building blocks for the application, the design of which is based on the concept of process networks. We chose a process network-based design, as it provides high modularity while still maintaining a simple structure. These characteristics are helpful when developing applications for laboratory evaluation and for deployment test applications.

The three main building blocks are *interfaces*, *tokens* and

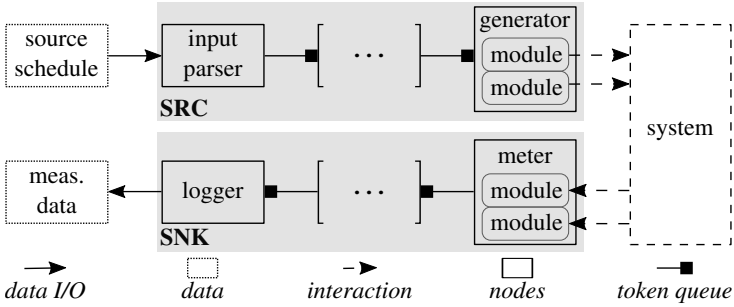


Figure 2.5: Structure of source and sink applications build with Experiment Orchestration Toolkit (ExOT). The application structure is based on process networks.

nodes. The interfaces are token queues, which are used to pass on tokens from one node to another. Tokens are data containers, the type and structure of which can be defined according to the needs of the nodes. There are three types of nodes, (i) producers, which have only an output token queue, (ii) processors, which have input and output token queues, and (iii) consumers, which have only an input token queue. The nodes are used to implement the main functionality of the applications. Basic applications, like source, sink and jammers, are built using unidirectional chains of nodes. Figure 2.5 shows a minimal abstract example of a source and sink application pair, built with the ExOT application library. A producer node, i. e., the input parser in the source or the meter node in the sink, generates tokens based on the input and passes them on to a chain of processing nodes. At the end of the process network chain, a consumer node, i. e., the generator node in the source or the logger in the sink, processes the incoming tokens and performs their respective action. The generator and meter nodes are special, as they can hold multiple modules. These modules allow a generator or a meter to interact with

multiple system components simultaneously in a timed fashion. Finally, the library also provides automatic node connectors and executors to reduce the burden on the developer.

The library also contains many different utilities to make the development of experimentation applications easier. For example:

- A simple and reliable JSON interface for application configuration.
- Logging and debug output.
- File system and Model Specific Register (MSR) handling.
- Execution, exception and signal handlers.
- Timekeeping and clocking.

In addition to the application library of ExOT, we provide a compilation suite. This compilation suite is based on docker and CMake, allowing easy cross-compilation and integration with other tool-chains, for example, the Android NDK. This enables researchers to port an analysis to different architectures easily.

2.4.2 Information flow

We base our data processing design on a layered information flow model, illustrated in Figure 2.6. Similar to the well known OSI model, information travels from the highest layer to the lowest, and then up to the highest again.

Layer 6 describes how input data is generated and how metrics are calculated from the measurement data. In layer 5 and 4, the source and line coding is defined, which is used to compress and shape the data stream depending on the channel specifications. Layer 3 describes the data format required

Layer Name	*Layer Functions*	
6 - Generate/Verify	generate bits	calculate metrics
5 - Source Coding	bits to symbols	symbols to bits
4 - Line Coding	symbol to trace	trace to symbols
3 - Raw Data Processing	output formatting	raw data to trace
2 - I/O Module	write schedule files	read meas. files
1 - Applications	utilise channel	observe channel
0 - Channel	covert information transmission	

Figure 2.6: Information flow model. Data travels from the highest to the lowest layer, gets transferred via the channel and travels up to the highest layer.

by the applications, while layer 2 defines file I/O. The two bottom layers describe the source (sending) and sink (receiving) applications and the channel.

Layers 2 to 6 are implemented as Python packages, which has the following advantages: (i) there is no need for recompilation when a new data processing scheme is tested, (ii) the implementation is platform-independent, and (iii) data checks and debugging are easy to perform.

2.4.3 Extendability and limitations

Due to its design, ExOT can be applied to a wide variety of fields and is not limited to analyses presented in this chapter. Using ExOT for additional channels, attacks or other analyses requires an extension of the experiment definitions in the Python framework. If specific deployment applications are necessary, these can be implemented using the building blocks of the C++17 library, or by extending the library.

The applicability of ExOT mostly depends on the devices the source, sink and jammer applications are run on. This dependency arises as the application building blocks provided

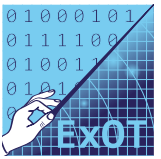


Figure 2.7: Left: Experiment Orchestration Toolkit (ExOT) logo; Right: QR-Code linking to the Experiment Orchestration Toolkit (ExOT) website

by the C++ library often depend on the architectural features of the devices. For example, the timing accuracy and maximum sampling period of the applications depends on the timing source provided by a device.

ExOT supports various devices that are based on a Linux or Android OS and require SSH or ADB capabilities of the devices. However, an extension of ExOT to other OSs or communication interfaces is possible. While including a new communication interface only requires to add a fitting driver to the Python framework, expanding ExOT to new OSs would also call for an extension of the C++17 library.

2.4.4 Availability

The Experiment Orchestration Toolkit (ExOT) is available on `exot.ethz.ch` [MKT20b] as an *open source* project. On the website, we provide further documentation of the project. This includes the following resources:

- A white paper with detailed descriptions of the underlying development paradigms.
- A wiki containing detailed manuals and "how-to" regarding the software.
- Links to the git repositories, including the source code

and application examples.

2.5 Applying the data leak evaluation methodology

In this section, we show how to apply the proposed methodology from section 2.3 using ExOT, which we introduced in section 2.4. We analyse and present an experimental evaluation for the covert channels presented in section 2.2. We determine a comparable metric, namely the capacity bound, for all of the considered covert channels. To provide experimental evidence for the validity of the models and bounds, we conduct experimental throughput evaluations. These allow a direct comparison of different implementations and platforms. In addition, we analyse the robustness of the implementations towards interference. To support the reproducibility of the analysis, all information gathered during the experiments is publicly available [MKT20a]. We perform the evaluation on

- a Lenovo T440p laptop based on an Intel i7-4700MQ running Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-58-generic x86_64), referred to as *Haswell-i7*, and
- a NVIDIA Jetson TX2 based on a dual-core Denver 2 64-bit CPU and quad-core ARM A57 cluster running Ubuntu 18.04.3 LTS (GNU/Linux 4.9.140-tegra aarch64), referred to as *ARMv8-Dev*.

2.5.1 Modelling and capacity bound derivation

We determine the capacity of the cache covert channels using the method which we outline in detail in chapter 4. This method is already implemented in ExOT as part of the

data leak analysis methodology. Furthermore, the application library of ExOT provides the basic building blocks for the measurement applications needed in this phase. Therefore, the implementation effort for this phase is reduced by using ExOT.

To derive the channel capacity bound using the method proposed in chapter 4, we need a channel model. Typically, there are multiple memory levels in a computing system, i. e., up to three cache levels, the main memory and swap. However, as already indicated in the original work [Gru+16a; Lip+16; YF14; Gru+16b], our initial experiments have also shown that it is only feasible to control and determine reliably whether data is cached or not. Considering this observation and the fact that no models of cache covert channels were given in the original work, we establish a channel model based on a state machine with two states and all possible transitions. Hence, we derive a maximum capacity of 1 bit per channel use.

To acquire a comparable metric, we need to determine the duration of one channel use. We define one channel use as (i) measuring the timing with the defined method, and (ii) resetting the memory state. To measure the channel access times, we employ an application developed using the ExOT application library. To outline the measurement procedure, the disassembly of the application for the Flush+Reload capacity channel on Haswell-i7 is shown in Listing 2.1. This application is also published as part of ExOT [MKT20b].

In the example, an address from shared memory is loaded into register `r10`. We measure the time needed to execute the operation in line 9 using two accesses to the Time Stamp Counter (TSC) at lines 5 and 11. In line 17, we reset the channel and the lines 19 to 25 are needed to calculate the time delta between the two calls to the TSC, i. e., the execution time of the instruction in line 9. The rest of the instructions, for example, `mfence`, are required to prevent the instructions in the

```
1  leaq    12(%rsp), %r10  ;; Setup
2
3  mfence                  ;; Measurement begins ...
4  cpuid
5  rdtsc
6  movl    %eax, %r8d
7  movl    %edx, %edi
8
9  mov     (%r10), %eax    ;; The measured operation
10
11 rdtscp                  ;; ...Measurement ends
12 movl    %eax, %r9d
13 movl    %edx, %esi
14 cpuid
15 mfence
16
17 clflush (%r10)          ;; Resetting the channel
18
19 salq    $32, %rsi       ;; Compute time difference
20 movl    %r9d, %r9d
21 salq    $32, %rdi
22 movl    %r8d, %r8d
23 orq     %r8, %rdi
24 orq     %r9, %rsi
25 subq    %rdi, %rsi
```

Listing 2.1: Disassembly of the measurement procedure for one channel use of the Flush+Reload cache covert channel.

measurement routine from being reordered by the processor.

We measure the channel access times for Flush+Reload, Flush+Flush and Flush+Prefetch access on either a cached or not-cached set, as shown in the example above. The measurement results shown in Table 2.2 are the average time acquired from a quarter-million measurements per channel and cache state. Using the best-case timing for each cache channel, we derive the upper channel capacity bounds.

Comparison. The formal models of the considered covert

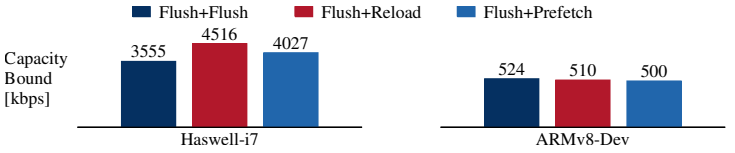


Figure 2.8: ExOT helps to derive channel capacity bounds, required by the proposed methodology. The capacity bounds allow direct comparison of different covert channels and indicate that cache based data leaks have a high threat potential.

channels enable us to derive capacity bounds, outlined in Figure 2.8. As expected, the capacity bounds suggest that the threat potential of cache based data leaks is high. However, the capacities vary by almost 10× when comparing our two devices Haswell-i7 and ARMv8-Dev. Furthermore, the throughputs reported by Gruss et al. [Gru+16a] for their ARM-core based device are higher than the capacity bounds we derived for the ARMv8-Dev platform. This indicates that the threat potential of cache covert channels is highly dependent on the specific processor architecture and, therefore, highlights the importance of data leak evaluations on different devices and architectures.

2.5.2 Experimental channel evaluation

As a final step, we conduct experiments to show how our methodology and ExOT help to validate the previously established model and the capacity bounds. Furthermore, we show how ExOT helps to assess how other applications influence the performance of the covert channels.

Our two platforms are described in the two environment descriptor TOML files, partly outlined in Listings 2.2 and 2.3. The environment descriptor files define zones for each

Covert Channel	Haswell-i7		ARMv8-Dev	
	cached	flushed	cached	flushed
Flush+Flush	281.3 ns	294.6 ns	1910.1 ns	2183.4 ns
Flush+Reload	221.4 ns	312.9 ns	1960.7 ns	2310.0 ns
Flush+Prefetch	248.3 ns	328.5 ns	2001.4 ns	2450.7 ns

Table 2.2: Time for one covert channel use depending on the cache state.

platform, i. e., `[host]` and `[combined]` respectively. In the zone settings, `model` defines what kind of computing device is used and which `cores` are mapped to the zone. In addition, further system parameters, like the available operating frequencies (`frequencies`), can be specified to ensure the experiments are reproducible. `schedule_tag` defines the string pattern that is used to identify the source application schedules used in this zone. `path_apps` and `path_data` specify where the executables of the applications are located and where the experiment engine will put the experiment data during execution. These paths are relative to the `user` home path. In the remaining lines, the necessary parameters to set up the connection to the zone are specified. We use an SSH connection with the `user="exot"` and key-authentication. If access to the zone is restricted via the network, a gateway can be used as well.

Listing 2.4 the general experiment settings from the ExOT experiment configuration file are listed. These settings are used for all remaining experiments in this chapter. In lines 30 and 31 we define the type of experiment that we want to execute and which type of channel to use. These two values determine which experiment parameters are needed and which ready-to-use analyses can be done on the experiment data. The remaining parameters are applied to all platforms and used to setup the experiment environment. `latency` defines the maximum wake-up latency of the system and is used to restrict the platform from entering deep sleep states. We use this setting to prevent timing variations in the experiment execution. `fan` defines whether a fan is used and, if possible, which fan level. We overwrite the `fan` setting for ARMv8-Dev in line 42, as this platform expect a numerical value rather than a Boolean. Furthermore, we set the operating frequency of the platforms, the sampling period of the sink application and the delays after spawning the different applications. These delays

```
[host]

# zone/platform details
model        = "Lenovo T440p"
cores        = [0, 1, 2, 3, 4, 5, 6, 7]
frequencies  = [800000,  900000, 1000000, 1100000,
↪  1300000, 1400000, 1500000, 1600000, 1700000, 1800000,
↪  1900000, 2100000, 2200000, 2300000, 2400000]
schedule_tag = "t440p"

# app and data location
path_apps    = "bin"
path_data    = "data"

# connection details
driver_type  = "SSHUnixDriver"

[host.driver_params]
ip           = "XXX.XXX.XXX.XXX" # masked value
port         = 51808
user         = "exot"
group        = "exot"
key          = "$EXOT_ACCESS_DIR/id_ed25519"
# gateway      = "exot-gateway"
```

Listing 2.2: Snippet from the ExOT environment descriptor file for Haswell-i7. The IP-address has been masked.

are used to make sure that starting the applications does not influence the measurements.

To quantify the performance of the covert channels, we divide the analysis into two phases, training and evaluation, outlined in Listing 2.5. During the training phases, we transmit a 1.5 kbit random bitstream, which we use to train the decoder decision device. As a decision device, we use a linear support vector classifier[2] In the evaluation phase, a 5 kbit random bitstream is transmitted and evaluated using the trained

[2] `sklearn.svm.LinearSVC`

```
78 [combined]
79
80 # zone/platform details
81 model        = "Nvidia Jetson Tegra TX2"
82 cores        = [0, 1, 2, 3, 4, 5]
83 frequencies  = [345600, 499200, 652800, 806400, 960000,
   ↪ 1113600, 1267200, 1420800, 1574400, 1728000, 1881600,
   ↪ 2035200]
84 schedule_tag = "tegraTX2"
85
86 # app and data location
87 path_apps    = "bin"
88 path_data    = "data"
89
90 # connection details
91 driver_type  = "SSHUnixDriver"
92
93 [combined.driver_params]
94 ip           = "XXX.XXX.XXX.XXX" # masked value
95 port         = 51808
96 user         = "exot"
97 group        = "exot"
98 key          = "$EXOT_ACCESS_DIR/id_ed25519"
99 # gateway      = "exot-gateway"
```

Listing 2.3: Snippet from the ExOT environment descriptor file for ARMv8. The IP-address has been masked.

decoder decision device. We repeat each phase 5 times and use the mean of the error rate of each repetition for further calculations, to compensate for variations introduced by the setup. Such variations in single measurements might be introduced by the OS running on the tested device, for example, sudden utilisation peaks caused by vital system services which we cannot disable. We execute both phases, training and evaluation, for multiple bitrates, so that we perform a bitrate sweep to quantify the throughput to bit error rate relation.

In the original work, the channel accesses were performed

```
[EXPERIMENT]
type    = "PerformanceExperiment"
channel = "Cache"

[EXPERIMENT.GENERAL]
latency                = 10
fan                    = true
governors              = "userspace"
frequencies            = "max"
sampling_period        = 2e-5
delay_after_spawn      = 5.0
delay_after_auxiliary = 1.0
active_wait            = true
ARMv8                  = {fan = "255", sampling_period =
↪ 0.000225}
```

Listing 2.4: Snippet from the ExOT experiment configuration file outlining the general experiment settings. These settings are used for all experiments to evaluate the three cache covert channels.

asynchronously at the highest possible rate. Therefore, the throughput could not be controlled, and it was not possible to show how the transmission rate influences the bit error rate. In contrast to the original implementation, our implementation, based on the ExOT application library, allows us to set the transmission rate and time the channel accesses. Therefore, we can perform a bitrate sweep from 6.4 kbps to 5 Mbps with a step size of 64 kbps to show how the bit error rate changes with increasing throughput. However, the timekeeping necessary to control the transmission rate limits our maximum sampling period. When we use one cache set and let the sampling rate approach 5 μs, we start to observe transmission errors due to inaccurate timing, i. e., channel accesses are not synchronised. Thus, to be able to achieve high bitrates with longer sampling periods, we use 64 cache sets in parallel, defined in the data

```
[EXPERIMENT.PHASES.trainARMv8]
bit_count    = 1500
symbol_rates = "[100] + (list(range(250, 6001, 250)))"
repetitions  = 5

[EXPERIMENT.PHASES.evalARMv8]
bit_count    = 5000
symbol_rates = "[100] + (list(range(250, 6001, 250)))"
repetitions  = 5

[EXPERIMENT.PHASES.trainHaswell]
bit_count    = 1500
symbol_rates = "[100] + (list(range(1000, 78001, 1000))  +
↪  [78125])"
repetitions  = 5

[EXPERIMENT.PHASES.evalHaswell]
bit_count    = 5000
symbol_rates = "[100] + (list(range(1000, 78001, 1000))  +
↪  [78125])"
repetitions  = 5
```

Listing 2.5: Snippet showing the measurement phases settings from the ExOT experiment configuration file.

```
[EXPERIMENT.LAYERS]
src = {name = "BitsetCoding", params={bitset_length = 64}}
lne = {name = "MultiN", params={N=64}}
rdp = {name = "DirectActivation", params={}}
io  = {name = "TimeValue", params={timebase='ns'}}
```

Listing 2.6: Snippet from the ExOT experiment configuration file presenting the data processing layer settings. These settings are used for all experiments to evaluate the three cache covert channels. As we use 64 cache sets in parallel, `bitset_length` for the source-coding layer `src` and `N` for the line-coding layer `lne` are set to 64.

```
70 [ENVIRONMENTS.Haswell.APPS]
71 src = {executable = "generator_cache_read_st", zone =
   ↪ "host"}
72 snk = {executable = "meter_cache_ff",          zone =
   ↪ "host"}
```

Listing 2.7: Snippet from the ExOT experiment configuration file outlining the application zone mapping for Haswell-i7. While all experiments use the same source application, a different sink application is used depending on the evaluated cache channel. Both Haswell-i7 and ARMv8-Dev use applications based on the same source code from the ExOT C++ library but compiled for the respective processor architecture.

processing layer settings outlined in Listing 2.6. The cache lines we used have a spacing of 64 sets on Haswell-i7 and 16 sets on ARMv8-Dev. This cache channel parametrisation allows us to sample every 22 μs on Haswell-i7 and 225 μs on ARMv8-Dev. We determine the sampling rate on both platforms using the worst-case time per channel use for one set from Table 2.2, plus a small security margin. The application settings, including the cache parameters, are outlined in Listings 2.7, 2.8 and 2.9, while the sampling rate is defined in Listing 2.4. For further information regarding the application parameters, please refer to the ExOT project [MKT20b].

After fixing the parameters for the communication channel setup, we use ExOT to generate and execute the necessary experiments. ExOT allows us to do this using a single python script, as shown in Listing 2.11. This supports reproducibility and allows for experiment versioning. Figure 2.9 illustrates the results for the evaluation of bitrate sweeps without interference in the upper plots. On Haswell-i7, this configuration allows us to establish transmissions with less than 1% bit errors at almost 200 kbps using Flush+Flush, 300 kbps with Flush+Reload and

```
[ENVIRONMENTS.Haswell.src]
generator.host_pinning       = 3
generator.should_pin_host    = true
generator.cores              = [ 0 ]
generator.should_pin_workers = true
generator.host_policy        = "round_robin"
generator.host_priority      = 97
generator.worker_policy      = "round_robin"
generator.worker_priority    = 98
generator.use_busy_sleep     = true
generator.busy_sleep_yield   = false
generator.use_huge_pages     = true
generator.shm_file           = "/dev/hugepages/8"
generator.set_count          = 64
generator.set_increment      = 64

logging.append_governor_to_files        = false
logging.async                           = true
logging.async_size                      = 4096
logging.log_level                       = "debug"
logging.provide_platform_identification = false

schedule_reader.reading_from_file = true
```

Listing 2.8: Snippet from the ExOT experiment configuration file showing the source application settings for Haswell-i7. These settings are specific to the used source application and are described in detail in the ExOT project wiki [MKT20b]. We use the same settings for ARMv8-Dev, except that `generator.set_increment=16`.

around 500 kbps for Flush+Prefetch. The significant increase of errors around 1 Mbps is caused by timing issues of our implementation. These lead to inaccurate synchronisation of the source and sink application and to higher errors, as at higher throughputs there are less samples per symbol. On ARMv8-Dev the channels allow throughputs of around 15 kbps using Flush+Flush, 40 kbps with Flush+Reload and

```
98  [ENVIRONMENTS.Haswell.snk]
99  logging.append_governor_to_files        = false
100 logging.async                           = true
101 logging.async_size                      = 4096
102 logging.log_level                       = "debug"
103 logging.provide_platform_identification = true
104 logging.timestamp_files                 = false
105 logging.rotating_logs       = false
106 logging.rotating_logs_count = 10
107 logging.rotating_logs_size  = 104857600
108
109 meter.host_policy       = "round_robin"
110 meter.host_pinning      = 7
111 meter.should_pin_host   = true
112 meter.host_priority     = 95
113 meter.log_header        = true
114 meter.start_immediately = false
115 meter.use_busy_sleep    = true
116 meter.busy_sleep_yield  = false
117
118 cache.use_huge_pages = true
119 cache.shm_file       = "/dev/hugepages/8"
120 cache.set_count      = 64
121 cache.set_increment  = 64
```

Listing 2.9: Snippet from the ExOT experiment configuration file illustrating the sink application settings for Haswell-i7. These settings are specific to the used sink application and are described in detail in the ExOT project wiki [MKT20b]. We use the same settings for ARMv8-Dev, except that `cache.set_increment=16`.

Flush+Prefetch, for less than 1% bit errors. The discrepancy of the results in comparison with the original work (see Table 2.1) can be explained by the differences between the implementation and platforms. In addition, the experimental results show a high degree of difference between the capacity bound and the achieved throughputs. This is caused by the

```
123 [ENVIRONMENTS.Haswell.APPS.ffmpeg]
124 executable         = "ffmpeg"
125 type               = "standalone"
126 start_individually = true
127 zone               = "host"
128 args               = [ "-y", "-loglevel", "error",
    ↪ "-stream_loop", "-1", "-i", "media/video.mp4", "-c:v",
    ↪ "libx264", "-b:v", "1000k", "-f", "null",
    ↪ "/dev/null",]
```

Listing 2.10: Snippet from the ExOT experiment configuration file outlining the jammer application settings for Haswell-i7. The settings used for ARMv8-Dev.

limitations of our implementation, namely the limitation of cache sets and sampling rate we can use, rather than channel effects. Moreover, the capacity bounds are not very tight, as they are based on the optimistic assumption that all channel accesses can be done with the best case time reported in Table 2.2. Therefore, we still consider the experimental evidence sufficient to validate the capacity bounds.

Interference. To evaluate the performance of the covert channels under the influence of other applications, we repeated the sweeps while running a jammer application. ExOT allows us to integrate the jammer application into the experiment execution by adding the parameters to the respective experiment configuration file. Yet, ExOT maintains a controlled experiment environment and guarantees a timed execution of the source, sink and jammer application. To cause additional utilisation of the cores and cache operations, we started video encoding using ffmpeg one second prior to the transmission. We configured ffmpeg to run an infinite loop to convert a 9.56 minutes long animated mp4 video[3] and pipe the output to `/dev/null`.

[3] `peach.blender.org`

The results of the evaluation sweeps with interference are illustrated in the lower plots in Figure 2.9 and show that the channels suffer from interference. In addition, the robustness of the cache channels can be increased by using multiple cache lines for redundant transmissions, rather than increasing the throughput. Therefore, we consider the cache covert channels to be robust and ultimately to pose a considerable threat in multicore systems.

Lessons learned. The data leak evaluation we have presented in this section highlights three points:

- Throughput results can vary to a great degree for different implementations.
- It is necessary to derive capacity bound as a metric for comparing data leaks rather than the throughput to compare implementation.
- There is a need for experimental evidence to validate the models and metrics.

These three points highlight the need for an extensive experimental evaluation, which can be very costly. However, the evaluation of the cache covert channels has also illustrated that we can reduce the resource overhead necessary for such an extensive experimental study by following the proposed methodology (see section 2.3) and taking advantage of ExOT. Furthermore, ExOT supports the repeatability of the experimental study, as all vital data is gathered in one location and, therefore, is easy to publish [MKT20a].

```
# Instantiate, generate and write the experiment
channel = ChannelFactory()(
  config["EXPERIMENT"]["channel"])
experiment = ExperimentFactory()(
  config["EXPERIMENT"]["type"], config=config,
  channel=channel)
experiment.generate()
experiment.print_duration()
experiment.write()

# Execute sequentially in all environments.
for env in environments_to_execute:
  experiment.execute_in_environment(env,
  ↪  phases[env]['normal'], resume=False)
experiment.write()

# Analyse the experiment
for env in phase_mappings:
  tmp_phase_mapping=dict(
    [tuple(phase_mappings[env][to_evaluate])])
  experiment.calculate_performance_metrics(
    phase_mapping=tmp_phase_mapping
    envs=[env],
    reps=[],
    **analysis_args[env],
  )

# Show the performance metrics...
experiment.performance_metrics.head()

# ...and the aggregated performance metrics
experiment.aggregate_performance_metrics()

# Serialise the experiment with the calculated performance
↪  metrics
experiment.write()
```

Listing 2.11: Snippet from the ExOT experiment generation, execution and analysis script. All channels are evaluated using similar simple scripts.

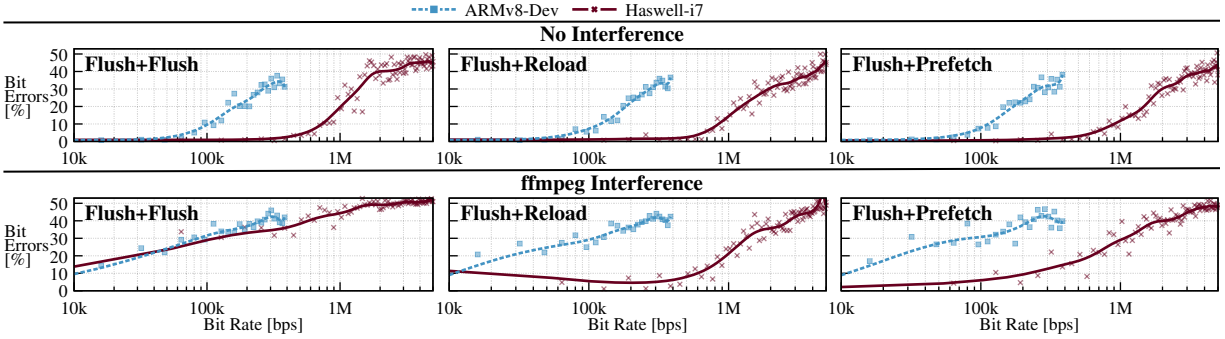


Figure 2.9: ExOT allows the evaluation of different channels on different platforms and different scenarios with low overhead. Without interference, the bit error increases similarly for all three cache covert channels. The throughputs of the Flush+Flush covert channel are more deteriorated by the ffmpeg interference. In general, higher throughputs can be achieved on Haswell-i7.

2.6 Summary

In this chapter, we proposed a *methodology* for covert channel evaluation and presented the *Experiment Orchestration Toolkit (ExOT)*. Our methodology helps researchers to define models, metrics and experiments for data leak analyses. This supports the reproducibility, comparability and expressiveness of covert channels analysis results. The Experiment Orchestration Toolkit (ExOT) implements the methodology and aims to reduce the engineering effort needed for an analysis and improve the reproducibility of experiments. The source code and documentation of ExOT are *publicly available* for download on `exot.ethz.ch` [MKT20b].

We showed the effectiveness of our methodology and ExOT by evaluating three variations of cache covert channels. In our evaluation, we captured different aspects of the data leaks by determining capacity bounds as well as empirical throughput and error rates. Furthermore, we showed that ExOT allows researchers to evaluate external influences on the covert channel with low overhead. Such an extensive analysis and the resulting comparable metrics will help researchers to understand which data leaks need immediate action and which ones can be tolerated.

3

Analysing continuous covert channels

In the previous chapter, we introduced a methodology that describes how to conduct an exhaustive data leak analysis to assess its threat potential. In this chapter, we show how to apply this methodology to a value and time continuous covert channel.

We present an extensive theoretical and experimental analysis of the thermal covert channel in multicore systems. We show how to model and derive upper channel capacity bounds for such continuous covert channels. Moreover, we define an advanced communication scheme for the covert channel based on our channel model and evaluate it on three different devices, using the Experiment Orchestration Toolkit (ExOT) presented in chapter 2. Furthermore, we briefly analyse the thermal covert channel regarding its robustness towards external influences.

This chapter is based on work presented in [BMT16], [MT20] and [Sel17].

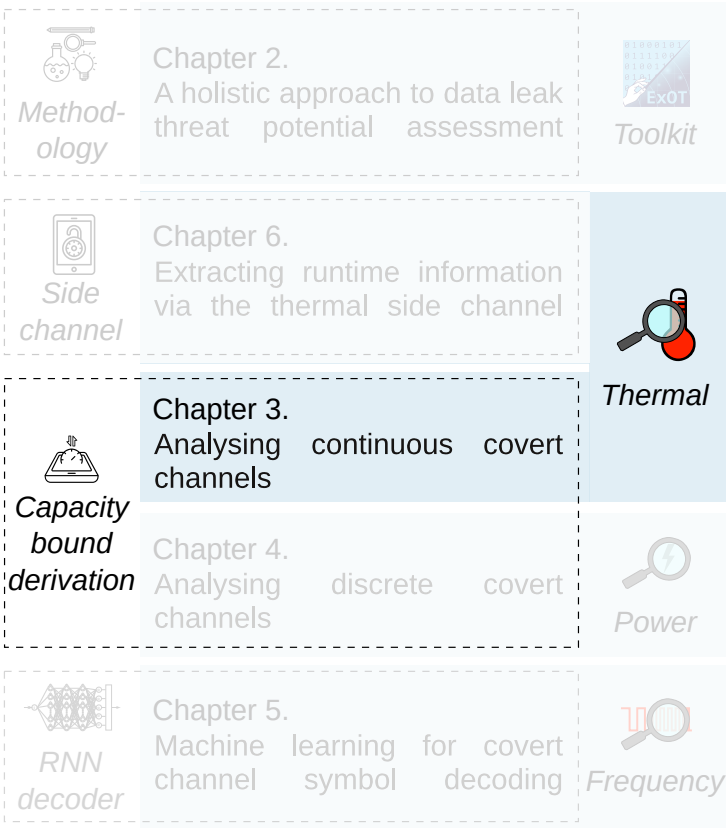


Figure 3.1: In chapter 3, we show how to apply our data leak analysis methodology on value and time continuous covert channels. We derive upper channel capacity bounds and present an advanced communication scheme for the thermal covert channel in multicore systems.

3.1 Introduction

After the breakdown of Dennard Scaling [Esm+11], power density grows with increasing integration in CMOS technology. Due to this effect, switching too many transistors at the same time generates more heat than can be dissipated, possibly damaging the chip by exceeding the maximum safe temperature. While hardware-driven Dynamic Thermal Management (DTM) [BM01] can avoid damages and ensure integrity, it resorts to techniques that severely impair performance, such as sharp speed throttling. For this reason, most current multicores expose a software interface to the temperature sensors to enable smarter thermal management policies that gracefully impact performance and avoid triggering hardware DTM. For example, Intel Core processors expose one sensor per core. Similarly, Arm big.LITTE Systems-on-Chips (SoCs) expose multiple sensors for their cores. These sensors are easily accessible on laptops or desktops running Windows or Linux through simple tools that export temperature information to userspace processes. Additionally, we verified that user-installed apps could access temperature sensors on Android-based smartphones and tablets without requiring any specific permissions.

Temperature sensors are a valuable asset for thermal management, but they can represent a security breach in *privilege-separated* or *sandboxed* systems. A widespread example of such systems is an Android-based smartphone, where each app has access to data and resources based on user-granted system permissions. Another example is sandboxing in state-of-the-art browsers, where each tab runs in an isolated process with restricted permissions [RBP09].

Recent research [Mas+15] provides evidence that temperature sensors can be used to implement a *covert channel* [Lam73] that allows applications that are otherwise isolated to commu-

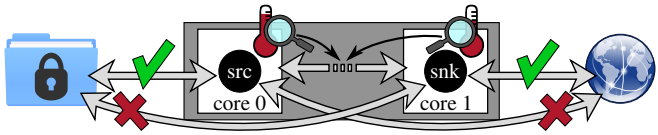


Figure 3.2: The source app (src) has access to restricted data, but no network access; the sink app (snk) has no access to the restricted data but has network access. A compromised source app can leak sensitive data to the sink app through the thermal covert channel, violating the security paradigm of application isolation and privilege separation.

nicate and possibly leak sensitive data. For instance, consider the dual-core system depicted in Figure 3.2. A source (src) app runs on core 0 and has access to sensitive data that is only stored locally, but it does not have network access. A sink (snk) app runs on core 1 and can freely communicate over the network but has no rights to access the sensitive data. In theory, privilege-separation should disallow communication between the two applications and keep the sensitive data secured, even in the presence of a compromised source app and a malicious sink app. However, if the sink app can read the on-chip temperature sensors, communication is possible through the thermal covert channel, regardless of application isolation.

If the system load is low, the source app can exploit the *sleep-states* [Bar15] used in current multicores to save energy and increase battery life to influence the temperature of its core predictably and, due to heat transfer, the temperature of the nearby cores. When the source app is active, its core wakes up and dissipates heat, thus raising the temperature. When the source app is idle, its core goes back to sleep and the temperature drops. At low load, the other cores are mostly in sleep-mode and do not introduce much noise. The source

app exploits this effect to encode a message into its execution trace; the sink app can retrieve the message by decoding the temperature trace it reads from the on-chip sensors. In section 3.2, we specify in more detail our threat model, while section 3.3 illustrates how we model this covert communication channel.

Previous work [Mas+15] presents an empirical study of the *1-hop channel*, i. e., when the sink app can read the temperature of the core physically next to the one where the source app runs. This study outlines experiments that achieve a throughput of up to 1.33 bits per second (bps) with an error rate of 11% on an Intel Xeon-based server. This result demonstrates the feasibility of communication on the 1-hop channel at low rates. Yet finding the actual channel capacity and the achievable rates and evaluating different platforms remain challenging open questions. We need to answer these questions in order to understand the possible entity of this threat in current systems.

Contributions. In this chapter, we present and exploit a new methodology that mixes theoretical and experimental analysis to tackle two main challenges:

1. Estimating the capacity (under controlled but realistic conditions) of the thermal covert channel; and

2. Finding a communication scheme that improves previous throughput results towards the channel capacity.

Both for estimating the channel capacity and for evaluating the throughput of our communication scheme, we use experimental data collected from diverse multicores representative of server systems, laptops, smartphones and automitive applications, compared to the single server platform studied in previous work. Section 3.4 illustrates our experimental setup. We estimate that the capacity can be in the order of 50 bps for

the 1-hop channel and in the order of 300 bps for the *same-core channel*, i. e., when the sink app can read the temperature of the core where the source app runs (section 3.5). Moreover, we show a communication scheme that achieves rates of more than 10 bps on the 1-hop channel and more than 100 bps on the same-core channel, with less than 1% error probability (section 3.6). This result is much higher than the maximum rate of 1.33 bps on the 1-hop channel with 11% error probability achieved in previous work [Mas+15] with a naïve communication scheme.

3.2 Threat model

We are interested in the scenario introduced in the example of Figure 3.2. Without loss of generality, we assume that the sink app only records a temperature trace by reading the sensors and later sends it to the attacker over the network; message decoding is done offline by the attacker. Thus, the sink app is mostly idle and only periodically wakes up to read the sensor.

We target current multicore devices, which implement per-core sleep states to extend battery life. On these devices, the Operating System (OS) puts idle cores to sleep and, when sleeping, cores consume little power and produce almost zero heat. On Intel Core processors, when scheduling the idle thread, the OS calls the `mwait` instruction to switch the current core from the active state to a lower *c–state* and save power. For instance, the `C1-HSW` state, which implements clock-gating on the Haswell generation of these processors, brings most of the power savings for a cheap wakeup latency of 2 μs [Bar15]. Switching to deeper c-states saves more power but implies a higher wakeup latency, up to hundreds of μs. ARM big.LITTLE multicores implement a similar, but simpler, hierarchy, where the `C1` state implements clock-gating. Assuming no scheduling

artefacts, even a costly wakeup latency of 200 μs only puts a loose upper bound of 5 KHz on how fast the source app can switch.

We note that the current devices that we target are idle or lightly-loaded most of the time (e. g., a smartphone resting in a pocket or a laptop only running a text editor). Thus, the source and sink app can wait for the system load to be low before starting to use the covert channel, so as to avoid interference. We briefly evaluate the impact of background load in subsection 3.6.3, but we leave a more detailed study of interference to future work. In this chapter, we focus on bounding the channel capacity and studying achievable rates in controlled, yet realistic, conditions that enable repeatability of our experiments. We also present a minor study outlining the capabilities of the thermal covert channel in a realistic scenario. Thus, we set the environment to limit interference and noise as much as possible (section 3.4); subsection 3.6.3 presents a study of the sensitivity of our results to departure from this controlled environment.

Finally, we note that current multicores, e. g., Intel Core processors or ARM multicores, generally feature one temperature sensor per core and that these sensors are easily accessible by userspace processes or apps. For instance, on Linux, `lm_sensors` exports a simple command-line interface; on Windows, CoreTemp offers a graphical interface. While setting up these tools might require administrative rights (e. g., `# sensors_detect`), they are commonly installed on client devices. On Android devices, the temperature sensors are even easier to access for the apps: we verified that the CPU-Z app (v. 1.15), available on the Google Play Store, requires no system permissions to be installed and it reports several temperature measurements on a Nexus 4 running Android 5.0.2. Moreover, once the sensors are exposed, any app can read all sensors

through the userspace interface, regardless of on which core it runs.

3.3 Communication channel model

We study a family of *storage covert channels* [US 85; MP14] where a source and a sink app share a multicore processor and covertly communicate through the on-chip temperature sensors. Assuming that the source app runs on core n, we can define at least as many channels as there are temperature sensors. Similarly to previous work [Mas+15], we consider one sensor per core and a floorplan with cores in a linear array, as commonly found on multicores with a moderate number of cores[1]. While the actual floorplan of our experimental platforms is not documented, the results we obtain are compatible with this assumption and our definitions can be adapted to a more general topology. Since the sink app is mostly idle and, on current systems, usually has access to all the sensors, it is not important on which core it runs; we merely assume that it runs on a different core than the source app. As Figure 3.3 illustrates, when the sink app reads the temperature of core n (the one where the source app runs), we have the *same-core channel*. Similarly, we have an *m-hop channel* when the sink app reads the temperature of a core m hops away from core n, i. e., core $(n \pm m)$.

We expect the same-core channel to have the highest capacity, as the thermal resistivity of silicon degrades the signal for the m-hop channels. In fact, the sink app can simply record a trace for each sensor and send all the data to the attacker,

[1]For example the 4th generation Intel Core Processors, see `http://images.anandtech.com/doci/7003/Screen%20Shot%202013-05-31%20at%207.59.16%20PM.png`

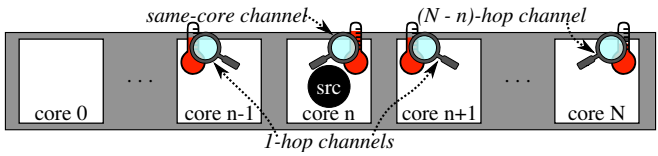


Figure 3.3: The sink app can establish several channels, depending on the physical location of the temperature sensor it reads with respect to the location of the source app.

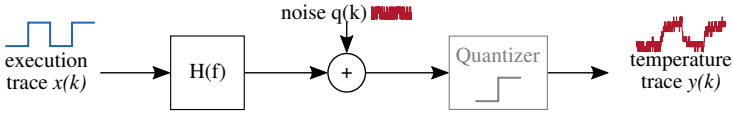


Figure 3.4: Discrete linear channel model with transfer function $H(f)$ from the execution trace $x(k)$ to the temperature trace $y(k)$, with additive noise $q(k)$. In our analysis, we neglect the quantiser.

who could always exploit the same-core channel. Studying the m-hop channels is, however, still interesting since system virtualisation may restrict the sink app to have visibility only over the sensor of its local core(s).

We consider the discrete-time channel model of Figure 3.4. The input to the channel is $x(k)$, the execution trace of the source app. At each instant k, $x(k) = 0$ if the source app is idle and $x(k) = 1$ if it is active. The output of the channel is $y(k)$, i. e., the temperature trace from the corresponding sensor. Similar to previous work [LDC02; SAS02; Rai+12], we use the linear block with transfer function $H(f)$ to model the temperature variations at the sensor caused by the execution trace. The additive noise $q(k)$ models thermal noise and any disturbances from other apps or the OS. The quantiser block models the fact that commercial processors offer a coarse sensor resolution, e. g., 1 °C on our platforms. Explicitly considering the quantiser

might increase the model accuracy, but adds a non-linear component, which is complex to analyse. For this reason, in our analysis, we ignore the quantiser and consider a linear approximation of the system. Our results (Sections 3.5 and 3.6) indicate that this approximation is reasonable.

Thanks to the model of Figure 3.4 (excluding the quantiser), we can employ the powerful tools available for the analysis of discrete linear dynamic systems for estimating the channel capacity (section 3.5). Additionally, we refer to this model to design the experiments that evaluate the throughput achieved with our transmission scheme (section 3.6).

3.4 Experimental setup

We base our analysis on experimental data collected from two diverse and representative hardware platforms:

1. A Lenovo ThinkPad T440p laptop, featuring a quad-core Intel Core i7-4710MQ processor clocked at 2.5 GHz running Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-58-generic x86_64)

2. A server rack based on a 3rd generation Intel Xeon E5-2690 octa-core processor clocked at 2.9 GHz, and running Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-137-generic x86_64).

3. An Odroid-XU3 board, featuring a Samsung Exynos 5422 SoC, including an Arm big.LITTLE processor with two quad-core clusters of A7 and A15 cores, respectively. The big cluster is clocked at 2.1 GHz, and the device runs Ubuntu 18.04.2 LTS (GNU/Linux 4.14.111-158 armv7l)

4. A NVIDIA Jetson TX2 based on a dual-core Denver 2 64-bit CPU and quad-core Arm A57 cluster. The Arm cluster is clocked at 2.04 GHz and the device is running Ubuntu 18.04.3 LTS (GNU/Linux 4.9.140-tegra aarch64).

In the rest of the chapter, we refer to platform 1 as Haswell-i7, to platform 3 as ARMv7-Mobile and to platform 4 as *ARMv8-Dev*. Haswell-i7 is representative of current business laptops; ARMv7-Mobile is representative of hand-held devices (it has the same SoC as the Samsung Galaxy S5 SM-900H smartphone); ARMv8-Dev is representative of multicore systems used in, for example, automotive applications. We use the platforms to analyse the channels for capacity estimation and to evaluate a communication scheme that achieves higher rates than previous work, in all cases, we use the following experimental setup. Additionally, we reproduce previous results [Mas+15] on our three platforms and evaluate our communication scheme on a third Sandy-Xeon platform (section 3.6).

3.4.1 System settings

On all platforms, we install Ubuntu 18.04.3 and use the `/dev/cpu_dma_latency` interface of the Linux kernel to limit the maximum wakeup latency to 10 μs. With this setting, the deepest *c-state* for Haswell-i7 is limited to `C1E-HSW` and `C1E-SNE` for Sandy-Xeon, with a wakeup latency of 10 μs. The deepest sleep state for ARMv7-Mobile and ARMv8-Dev is `C1`, with a wakeup latency of 1 μs[2].

On Haswell-i7, the temperature sensors are refreshed every 1 ms [Int15]. We were not able to find the sensors refresh period for ARMv7-Mobile on the SoC documentation. To determine this parameter, we collected several traces with a varying

[2]We retrieve wakeup latencies from the `sysfs` interface exposed at `/sys/devices/system/cpu/cpu$i/cpuidle/state/$n/latency`.

```
# set maximum fan speed level
echo 'level 7' > /proc/acpi/ibm/fan
```

Listing 3.1: Commands necessary to set the maximum fan speed level on Haswell-i7.

```
FAN_PATH = /sys/devices/platform/pwm-fan/hwmon/hwmon0
# Disable automatic mode
echo 0 > ${FAN_PATH}/automatic
# Set maximum fan pwm
echo 255 > ${FAN_PATH}/pwm1
```

Listing 3.2: Commands necessary to set the maximum fan speed level on ARMv7-Mobile and ARMv8-Dev

system load, using 1 ms as the sampling period. We noticed that the temperature only changed every 5 ms, which we took as the sensor refresh rate for this platform. Based on these characteristics, we set the sampling period to $T = 1$ ms for Haswell-i7 and Sandy-Xeon; $T = 5$ ms for ARMv7-Mobile and ARMv8-Dev. Therefore, the Nyquist frequency of our discrete system is $0.5/1\,\text{ms} = 500$ Hz for Haswell-i7 and Sandy-Xeon, while its 100 Hz for ARMv7-Mobile and ARMv8-Dev.

To favour repeatability, we run all experiments in a controlled, yet realistic, environment. We set all devices in an air-conditioned server room with an ambient temperature of ≈ 23 C° and, for both, we fix the fan speed to the maximum level (see Listing 3.1 and Listing 3.2) and set the clock frequency of active cores to the maximum, i. e., 2.5 GHz for Haswell-i7, 2.1 GHz for the big cores on ARMv7-Mobile and 2.035 GHz on ARMv8-Dev. In order to avoid scheduling artefacts, we run the source and sink app with the `SCHED_FIFO` scheduling class at highest priority by using the `pthread_setschedparam()` interface and pin the source app to one core by using the `pthread_setaffinity_np()` interface. During all experiments, the

system is idle except for the source and sink apps and the default system services of the Ubuntu installation.

For all three, the four cores of Haswell-i7, the four big cores of ARMv7-Mobile and the four Arm cores in ARMv8-Dev, we assume a linear floorplan, as shown in Figure 3.3. While the actual floorplan of the two platforms is not documented, our results are compatible with this assumption. We run the source app on the third core in the array, i. e., on core 4 on Haswell-i7, which has eight virtual cores with two-way hyper-threading, and on core 6 on ARMv7-Mobile, where cores 0 to 3 are the LITTLE cores and cores 4 to 7 are the big cores. On ARMv8-Dev, we run the source app on core 4, as cores 1 and 2 are the Denver cores. In the rest of the chapter, we only count the four physical (big/Arm) cores, starting from 0; thus, for all platforms, we say that we run the source app on core 2 and record the temperature traces from cores 0 to 3. On ARMv7-Mobile, we run the source app on the big cores since the LITTLE cores provide no temperature sensors and they do not sensibly affect the measurements on the big cores. This setup allows us to analyse one same-core channel (when looking at the temperature trace of core 2), two different 1-hop channels (when looking at either core 1 or core 3), and one 2-hop channel (when looking at core 0).

On Haswell-i7, we exploit hyper-threading and run the sink app with four parallel threads on the odd-numbered virtual cores; each thread reads the temperature of its core from the `/dev/cpu/$i/msr` interface. On ARMv7-Mobile and ARMv8-Dev, all the sensors are exposed via the sysfs in the respective directory `/sys/devices/virtual/thermal/thermal_zone$i/temp`, where `$i` is the thermal zone; on ARMv7-Mobile we run the sink app on the first LITTLE core and on ARMv8-Dev on the first Denver core. This setup avoids timing interference between the source and the sink app.

```
volatile double a = 2.0e0;
volatile double b = 2.0e0;

/* Heavy floating-point work loop. */
while (flag) {
  a *= b;
  b -= a;
}
```

Listing 3.3: Source app worker threat tight loop, used to keep cores active.

Unless differently specified, we use these settings in all our experiments. Since a real attack would not benefit from this controlled environment, in subsection 3.6.3 we analyse the sensitivity of our results to variations to these settings.

3.4.2 Reference apps

We base our reference apps on the Experiment Orchestration Toolkit (ExOT) application library, presented in subsection 2.4.1. Snippets 3.4 and 3.5 illustrate their main functionality.

The source app replays the execution trace that is passed via the token queue and consists of one host threat and one worker threat per core. The source app host threat (Listing 3.4) processes the tokens from the input queue and synchronises the worker threats using barriers. The generator host enables all workers which, depending on the subtoken, keep their core active (1) or stay idle by calling the sleep function (0). After sleeping for the specified time, the generator disables all workers, waits for all workers to report to the second rendezvous point and starts processing the next token. When activated, the workers execute a tight loop similar to the

```
while (!local_state_->is_stopped()) {
  // read token from the input queue
  in_.read(token);

  // Decompose the input token into duration and
  // subtoken. subtoken defines which cores are
  // active and which are idle. Worker threats
  // access the subtoken_ via shared memory.
  duration_ = std::get<duration_type>(token);
  subtoken_ = std::get<subtoken_type>(token);

  // Core routine:
  // (1) Workers start processing the subtoken...
  // (2) ... after the first rendezvous point.
  // (3) The host thread sleeps for the duration.
  // (4) Host stops the workers from processing.
  enable_flag_ = true;      // (1)
  barrier_.wait();          // (2)
  timer_.sleep(duration_);  // (3)
  enable_flag_ = false;     // (4)

  // Correct for timing offsets
  timer_.update_offset();

  // Second rendezvous point to synchronise all
  // workers threads
  barrier_.wait();
}
```

Listing 3.4: Stripped-down code for the reference source app host thread. We use multiple worker threads in addition to the host thread to utilise the different cores and employ barriers to synchronise these different worker threads.

one of the popular `cpuburn` stress-test[3] to keep their respective core active. The tight loop of the source app worker threat is outlined in Listing 3.3. To ensure proper timing of the execution

[3] `patrickmn.com/projects/cpuburn`

```
auto until  = [this]() {
  return !global_state_->is_stopped();
};
auto action = [this]() {
  out_.write(measure());
};

// If configured, pin the meter host threat to
// a specific core
if (conf_.should_pin_host) {
  exot::utilities::ThreadTraits::set_affinity(
    conf_.host_pinning);
}

// Set the scheduling used for the meter threat
exot::utilities::ThreadTraits::set_scheduling(
  conf_.host_policy,
  conf_.host_priority);

debug_log_->info("[meter] running on {}",
  exot::utilities::thread_info());

// Wait for the signal to start measuring
while (!global_state_->is_started()) {
  std::this_thread::sleep_for(
    std::chrono::milliseconds{100});
}

// Run `action` till `until` returns false.
timer_.run_every(conf_.period, until, action);
```

Listing 3.5: Stripped-down code for the reference sink app host thread. The host thread configures the measurement threads and a timer to ensure the measurement threads perform a measurement with the period `conf_.period`.

trace replay, the source app uses the Time Stamp Counter (TSC) of the system on Haswell-i7 and Sandy-Xeon. On ARMv7-Mobile and ARMv8-Dev we use the hardware performance

counter to generate timestamps, due to the lack of a TSC. Additionally (not shown in Listing 3.4), the source app keeps track of the overall elapsed time and keeps adjusting internal timer to avoid drifting apart due to jitter in the execution.

The sink app (Listing 3.5) samples the temperature sensors every $T\mu$s ($T = 1000$ for Haswell-i7 and ARMv8-Dev, $T = 5000$ for ARMv7-Mobile) and keeps a pre-allocated in-memory, which it dumps to a file after the application is stopped. Similar to the source app, the sink app uses barriers to synchronise all reading threats using barriers (not shown in the code). We register a signal handler to set the `global_state->is_started()` to start the measurement and `global_state->is_stopped()` flag at the experiment end. After the experiment end, we retrieve the log file and analyse it offline.

3.4.3 Platform characterisation

Before starting the thermal covert channel evaluation, we need to determine which thermal readings are available on the different platforms. In this study, we only focus on sensor readings within the processing core.

Haswell-i7 offers sensor readings for every logical core. However, the readings for two logical cores mapped to the same physical core are identical. Therefore, we have used four sensor readings on Haswell-i7, one for each physical core.

Similarly, ARMv7-Mobile offers thermal sensor readings for every big core. In addition, one sensor reading for the GPU is also available, which we do not consider in this chapter.

In contrast, ARMv8-Dev only supplies one sensor reading for all of its Arm cores, as outlined in Table 3.1. The sensor reading for all Arm cores reports the maximum temperature

Zone	Name	Sensed Location(s)
0	BCPU-therm	Centrally in each A57 CPU
1	MCPU-therm	Centrally in each Denver CPU
2	GPU-therm	Within the GPU
3	PLL-therm	Placed adjacent to PLLX
4	Tboard_tegra	Temperature of the board
5	Tdiode_tegra	Temperature on die near GPU

Table 3.1: Thermal zones of ARMv8-Dev as reported by Nvidia. If multiple sensors are located in one zone, the max measurement value of all sensors is reported.

measured in all of the cores[4]. In order to keep the analysis scenario consistent, we consider this sensor reading to be the temperature reported for each of the four Arm cores and do not consider the other thermal sensors.

On Sandy-Xeon, we are able to get one measurement for each of the 16 cores in the system. Therefore, we ran a preliminary experiment with a simple active/sleep pattern of the source application pinned to core 2 to determine which cores are of interest. The resulting trace for each core is illustrated in Figure 3.5 and show that the initial temperature for the cores 0 to 7 is around 45˚C, while for cores 8 to 15 its around 35˚C. We assume that this is caused by the separation of the cores into two sockets (or clusters), whereas the cores 0 to 7 form socket 1 and are in close physical vicinity. Moreover, knowing that the Sandy-Xeon processor has only 8 physical cores[5], we would expect that two cores in each cluster show the same temperature trace. However, only the temperature

[4] https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra%2520Linux%2520Driver%2520Package%2520Development%2520Guide%2Fpower_management_tx2_32.html%23wwpID0E05H0HA

[5] https://ark.intel.com/content/www/de/de/ark/products/64596/intel-xeon-processor-e5-2690-20m-cache-2-90-ghz-8-00-gt-s-intel-qpi.html

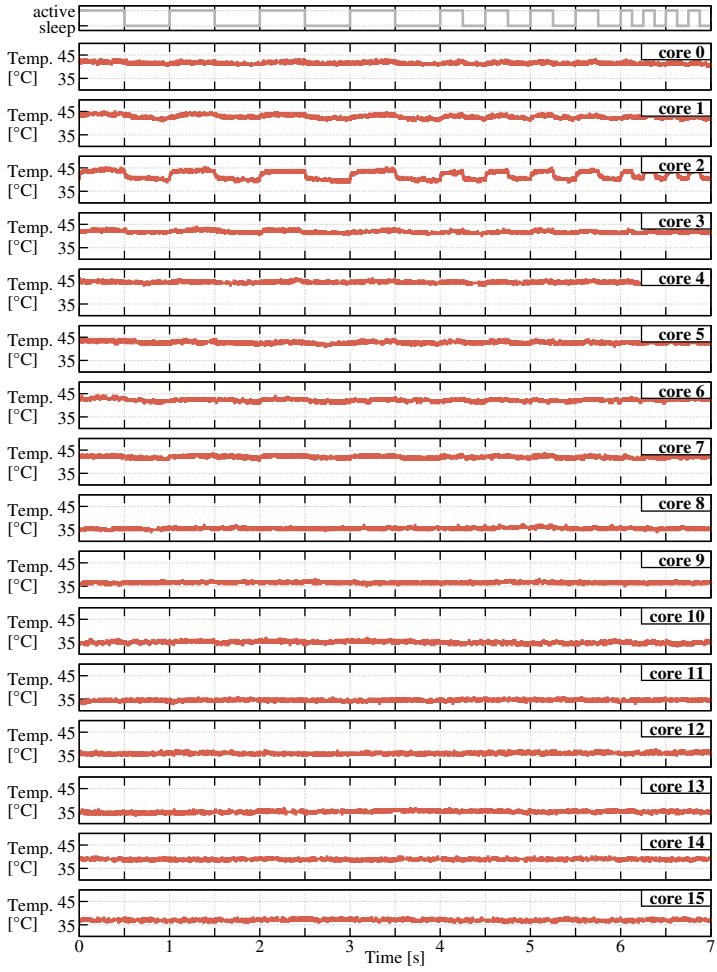


Figure 3.5: Temperature traces for Sandy-Xeon when the source app executes the trace in the top plot on core 2. We identify the two core clusters (or sockets) with cores 0-7 and 8-15. The thermal traces do not allow us to determine which logical cores are mapped to the same physical core.

on core 2 closely follows the execution trace of the source application, heating up when the source is active and cooling down when the application is idle. Therefore, other than for Haswell-i7, we cannot determine the logical core mapping simply by analysing the temperature trace. Unfortunately, we were not able to acquire any further information on the logical core mapping nor on how the thermal sensor readings are processed before they can be read through the Model Specific Registers (MSRs). Hence, we simply choose the cores 0 to 4 for the further analysis, assuming these thermal sensor readings are taken from the four different physical cores in socket 1.

Figure 3.6 shows the results of the same preliminary experiment run on Sandy-Xeon and also for the three other platforms. We use these results to characterise the temperature range and dynamics of our platforms. On all platforms, the source app runs on core 2 with the execution trace shown in the top plots. The execution trace is an active/sleep square wave with 50% duty cycle and varying frequency, with 4 periods each at 1 Hz, 2 Hz, and 4 Hz. The bottom plots report the resulting temperature traces for cores 0 to 3, i. e., for the same-core channel (core 2), the two 1-hop channels (cores 1 and 3), and the 2-hop channel (core 0). The dynamic temperature range for each platform and each core is indicated by the tics on the y-axis of each plot and highlighted in Table 3.2.

For ARMv8-Dev, all temperature traces are the same, as the system only reports one temperature reading for all cores. Therefore, there is no difference between the same-core, 1-hop and 2-hop channel. Furthermore, we note that despite the narrow dynamic temperature range of only 1°C, due to the sensor resolution of 0.5°C it is possible to reconstruct the input wave from the temperature trace for the whole experiment.

For the other three platforms, the sensor resolution is 1°C and the same-core channel resembles the response of a low-

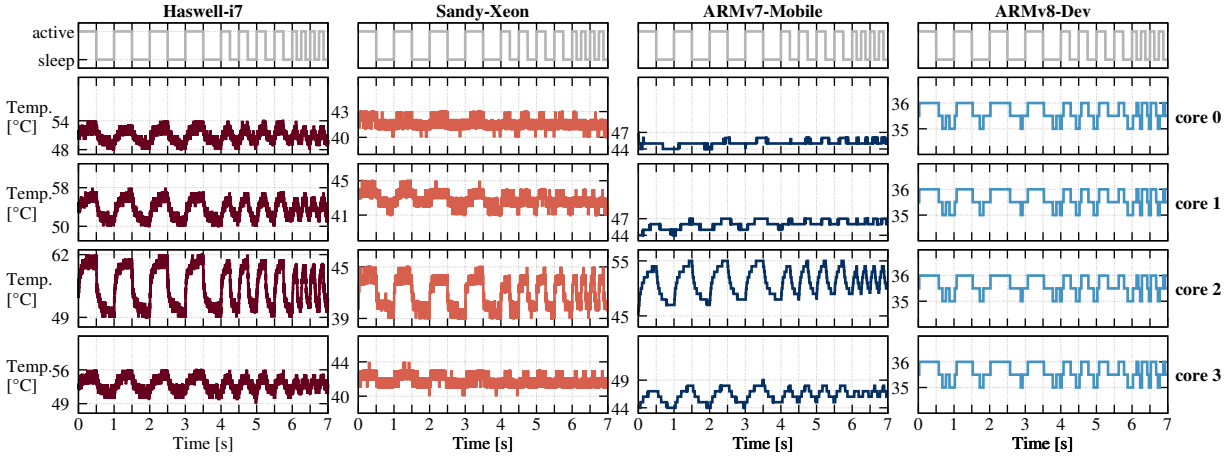


Figure 3.6: Traces from Haswell-i7, Sandy-Xeon, ARMv7-Mobile and ARMv8-Dev when the source app executes on core 2; the top plot shows the active/idle execution trace of the source app, the other plots show the temperature traces from the four cores. The dynamic temperature range is indicated by the tics on the y-axis for every plot. In general, the thermal dynamics are higher on the two Intel-based devices than on the Arm-based ones.

Platform	core 0	core 1	core 2	core 3
Haswell-i7	6.0 ℃	8.0 ℃	13.0 ℃	7.0 ℃
Sandy-Xeon	3.0 ℃	4.0 ℃	6.0 ℃	4.0 ℃
ARMv7-Mobile	3.0 ℃	3.0 ℃	10.0 ℃	5.0 ℃
ARMv8-Dev	1.0 ℃	1.0 ℃	1.0 ℃	1.0 ℃

Table 3.2: Dynamic temperature ranges measured on the different platforms for the experiment of Figure 3.6.

pass filter that oscillates between a high and a low value with a smoothed version of the input wave. Therefore, similar to ARMv8-Dev, for Haswell-i7, Sandy-Xeon and ARMv7-Mobile it is also viable to restore the complete input waveform from the temperature trace.

As expected, the execution trace is harder to infer from the 1-hop channels for the platforms Haswell-i7, Sandy-Xeon and ARMv7-Mobile. This is due to the increased distance of the corresponding sensors from the heat source. Moreover, the 1-hop channels show a different amount of attenuation and distortion for the different platforms. The trace from core 1 looks "better" than core 3 for Haswell-i7 and Sandy-Xeon, while the opposite is true for ARMv7-Mobile. Finally, Haswell-i7 shows much less attenuation for the 2-hop channel than Sandy-Xeon and ARMv7-Mobile, for which the temperature trace is basically flat, making the input trace impossible to reconstruct.

Haswell-i7 shows the least attenuation across the different channels, which is also reflected in the dynamic temperature ranges. For each channel, Haswell-i7 has a dynamic temperature range that is at least 2°C wider than the same channel on the other platforms. For the same-core channel, ARMv7-Mobile has a wider dynamic range of 10 °C, while the dynamic range on Sandy-Xeon is just 6 °C. However, for the 1-hop

Platform	Process	Package Area	TDP	Form	Cooling
Haswell-i7	22 nm	1200.0 mm^2	47 W	Laptop	Medium Fan
Sandy-Xeon	32 nm	2362.5 mm^2	135 W	Rack	Big Fans
ARMv7-Mobile	28 nm	213.0 mm^2	< 20 W	SBC + Case	Small Fan
ARMv8-Dev	16 nm	4350.0 mm^2	< 15 W	SBC	Medium Fan

Table 3.3: Platforms hardware specifications that influence the thermal behaviour. TDP, the Thermal Design Power, is the average power dissipated by the device at the base operating frequency with all cores active.

channels the dynamic range degrades less on Sandy-Xeon, where it is 4 °C for both core 1 and core 3, compared to the dynamic ranges reduction to just 3 °C and 5 °C, respectively, measured on ARMv7-Mobile. On the 2-hop channel, the temperature trace is basically flat for both platforms, Sandy-Xeon and ARMv7-Mobile. This different behaviour depends on the floorplan, fabrication characteristics and cooling system of the four platforms, outlined in Table 3.3.

Intuitively, on all platforms and for all channels, the dynamic range shrinks as the frequency of the input increases. As a notable example, the temperature trace of core 3 of ARMv7-Mobile shows significant variations as long as the input frequency is 1 Hz, but the signal is quickly lost when the frequency increases (from time 4 s on).

Finally, another important difference between the platforms lies in the incidence of noise in the temperature traces. The traces from the Intel based platforms Haswell-i7 and Sandy-Xeon present a sensible amount of noise, with the temperature constantly oscillating by 1 °C. Instead, the traces from the Arm based platforms ARMv7-Mobile and ARMv8-Dev show almost no noise and have an accentuated staircase-like quantisation effect, probably due to internal filtering in the sensors, which have a slower refresh rate compared to Haswell-i7 (5 ms versus 1 ms). The lack of noise on ARMv7-Mobile and ARMv8-Dev accentuates the signal attenuation at higher frequencies and further distance since temperature variations are only observable if the actual temperature varies across a quantisation boundary; otherwise, variations are hidden by the quantisation. Despite this difference, we found that we were able to stick to the linear channel model of Figure 3.4 for the platforms in our study to estimate the channel capacity (section 3.5).

The heterogeneity in the behaviour of these platforms

makes them good candidates for the source of representative data for our study of the capacity bounds (section 3.5) and for the evaluation of our communication scheme (section 3.6).

3.5 Capacity estimation

In the 1985 *Orange Book* [US 85], the U. S. Department of Defence reports that "*a covert channel bandwidth that exceeds a rate of one hundred (*100*) bits per second is considered high*" and that covert channels with "*maximum bandwidths of less than one (*1*) bit per second are acceptable in most application environments*". While these numbers might look somewhat different if estimated today, the 1.33 bps transmission rate with 11% error probability achieved by Masti et al. [Mas+15] for the 1-hop channel seems too low to be considered a threat in practice. Still, much higher rates with much lower error probability are possible when considering the same-core channel or a better communication scheme, as we show in section 3.6. In order to evaluate whether these channels can or cannot be a security threat, we need to find a reliable estimation of their capacity C, i. e., we need to find the upper bound on the rate of communication achievable through them with arbitrarily small error probability [Sha01; CT06].

Following Shannon's seminal work [Sha01], researchers extensively studied ways to determine the capacity of a wide range of channel models [CT06]. Yet even with this vast theoretical literature available, estimating the capacity of a physical channel remains very challenging: it requires using an appropriate model and retrieving quantitatively accurate measurements of the channel parameters, despite the noise and limited precision. We tackle this challenge by leveraging the simple model described in section 3.3 and determining its

transfer function $H(f)$ through carefully designed experiments based on the experimental setup described in section 3.4.

3.5.1 The theoretical approach to channel capacity bounds

The first step towards determining a good estimate of the channel capacity C is finding a suitable mathematical expression to compute it based on observable parameters. One of the simplest expressions for the channel capacity is

$$C = B \log_2 \left(1 + \frac{S}{N}\right) \text{ [bps]} \tag{3.1}$$

given by the Shannon-Hartley theorem [CT06], reported in Equation (3.1). The theorem gives the capacity C for the ideal, additive white gaussian noise (AWGN), band-limited channel with bandwidth B and Signal-to-Noise-Ratio (SNR) S/N.

Since Equation (3.1) applies exactly only to an ideal, band-limited, channel, we first need to verify whether we can reasonably approximate our channels this way. If this approximation is possible, we can determine the bandwidth B and the SNR S/N of our channels based on experimental measurements and use these values to estimate the capacity. To find the bandwidth, we try to fit a discrete-time dynamical system model to match the dynamics of the channels. For instance, we were able to fit the same-core channel of ARMv7-Mobile with a discrete-time model with six poles and four zeros [Hel+04]. Figure 3.7 shows the measured and modelled step response for this channel (left) and the magnitude Bode diagram of the corresponding model (right). The step response plot visually shows how well the model can predict the measured behaviour of the channel, while the Bode diagram

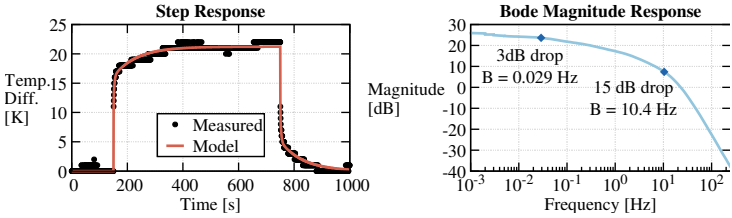


Figure 3.7: Step response of the same-core channel on ARMv7-Mobile; the input is 1 in the interval $[150, 750)$ s, 0 elsewhere.

shows the asymptotic approximation of the frequency response in logarithmic scale. In an ideal band-limited channel, we would expect the bode diagram to have a *rectangular* shape that lets a band of frequencies pass and blocks all the rest of the spectrum. While the model fits the step response well (the normalised mean-squared-error is 4.7%), its Bode magnitude plot does not allow to easily define the bandwidth B. On the one hand, the commonly used cut-off frequency at the 3 dB drop (shown in Figure 3.7) does not seem to be a good choice to determine the bandwidth in this case since the magnitude keeps decreasing slowly up to about 10 Hz, where there is a clear knee. On the other hand, using the frequency at the knee for the bandwidth would be rather arbitrary as well since the amplitude is far from constant up to there, with a 15 dB drop. Moreover, looking at the preliminary experiment of Figure 3.6, we notice that there is a significant attenuation when increasing the input frequency, even just from 1 Hz to 4 Hz. Therefore, using a fixed SNR value for the whole passband would not be accurate. We omit the step responses and Bode diagrams for the other channels and for Haswell-i7, Sandy-Xeon and ARMv8-Dev, as similar considerations apply in those cases. From these observations, we conclude that Equation (3.1) is not adequate to estimate the capacity of our channels since we are not able to estimate the required parameters reliably.

While using the Shannon-Hartley theorem is not effective in our case, we can leverage a different approach to find the capacity [CT06; VPL10]. We can search among all the possible input patterns $x(k)$ for the one that has the frequency characteristics that make the most information pass through the channel. In other words, we need to find the best allocation of the input power $\hat{S}_{xx}(f)$ across the frequency spectrum. If we can find this ideal allocation $\hat{S}_{xx}(f)$, we can use results from the information theory literature to compute the channel capacity. The key observation in this method is that we can only allocate as much power as we are able to put into our input signal, i. e., we have a power cap p_0 on how much power we can input into our system. The general approach to determining $\hat{S}_{xx}(f)$, and thus C, subject to a power cap p_0 is known as *water-filling* [CT06; VPL10]. The *water-filling* technique is based on the assumption that the optimal input spectrum is the one that allocates power such that the sum of the noise and the signal power is constant over the whole channel spectrum, so that there is more power from the signal in the parts of the spectrum with high SNR. We study two different solutions based on this technique. First, we consider the *classic* solution [VPL10], which considers the constraint p_0 on the average input power. Second, we analyse a *constrained-input* solution [HO92] that explicitly considers the extra constraint that the input to our channels is a binary value (active/idle).

Classic water-filling approach. The classic water-filling technique allows us to compute the capacity of channels with arbitrary transfer function $H(f)$ and additive Gaussian noise $q(k)$, not necessarily white [CT06; VPL10]. If we can estimate the power spectrum of the channel $S_{hh} = |H(f)|^2$ and of the noise S_{qq} then, given a cap p_0 on the average input power, we can derive the channel capacity according to Equation (3.2) [VPL10, Eq. (6.15)]. The capacity C_b is

determined by the *spectral power allocation* $S_{xx}(f)$, which cannot exceed the power cap p_0, as Equation (3.3) states. We can maximise the expression in Equation (3.2) and determine the capacity by intelligently shaping the power allocation S_{xx} so that more power is allocated at those frequencies with better SNR. This ideal allocation $\hat{S}_{xx}$ can be determined with a *water-filling* procedure [CT06; VPL10], which we do not describe in detail here.

$$C_b = \max_{S_{xx}} \left\{ \int_{\mathscr{F}} \log_2 \left(1 + \frac{S_{xx}(f) \cdot S_{hh}(f)}{S_{qq}(f)} \right) df \right\} \text{ [bps]}, \tag{3.2}$$

$$\text{under the constraint that } \int_{\mathscr{F}} S_{xx}(f) df \leq p_0 \tag{3.3}$$

As we will show in subsection 3.5.2, we are able to estimate S_{hh} and S_{qq} for our channels. Thus, we can use the water-filling procedure on Equation (3.2) to estimate the capacity C_b. We expect C_b to be an upper bound on the real capacity C because the classic water-filling approach does not consider the more stringent constraint that our input is required to be a binary value. In order to evaluate how much more stringent this constraint is, we use an additional result from the literature to compute a tighter upper bound on the real capacity.

Constrained-input water-filling. In a 1992 paper, Heegard and Ozarow [HO92] studied the capacity of *saturation recording*, i. e., the capacity of storage systems such as tape recorders or optical disks. While this problem has, in general, little to do with our study, it has the same *saturation* constraint on the channel input: input values can only be either 0 or 1. This shared property allows us to leverage their expression for an upper bound C_a on the channel capacity C [HO92, Eq. (11)]. We report this result (with minor notation changes) in

Equation (3.4). C_a depends on the value of the power spectrum of the channel S_{hh} and the parameter λ over A_λ, which is the set of frequencies $f \in (-\infty, \infty)$ for which $\lambda \cdot S_{hh} \geq 1$. The parameter

$$C \leq C_a = \max_{\lambda} \left\{ \frac{1}{2} \int_{A_\lambda} \log_2 \left(\lambda \cdot S_{hh}(f) \right) df \right\} \text{[bps]}, \tag{3.4}$$

λ must be maximised subject to the constraint of Equation (3.5), which ensures that the SNR does not exceed the ratio of the power cap p_0 over the noise power N_0. These equations assume that the noise is white, i. e., that the noise has a constant power spectrum $S_{qq} = N_0$ across the frequency range A_λ. Since, in

$$\frac{1}{2} \int_{A_\lambda} \left(\lambda - \frac{1}{S_{hh}(f)} \right) df \leq \frac{p_0}{N_0} \tag{3.5}$$

our channels, S_{qq} is not constant, we use this constrained-input solution only after applying a whitening filter to the spectra so that S_{qq} can be assumed constant; subsection 3.5.3 explains this technique in more detail. Finding the λ that maximises Equation (3.4) subject to Equation (3.5) follows a water-filling procedure again.

3.5.2 Determining the power spectra

To use the water-filling methods, we need to find reliable estimates for the power spectra of the noise and our channels on our two platforms. Computing reliable estimates from experimental data is challenging mainly due to (i) the limited temperature resolution (1 K) of the sensors, (ii) the noise (on Haswell-i7), (iii) the quantisation effect (on ARMv7-Mobile), and (iv) the saturation constraint on the input.

Noise spectra. S_{qq} is easier to estimate than S_{hh} since the input constraint does not play a role in this case. For all platforms, we simply record a 120 s long temperature trace for each channel, with the system idle except for the sink app, which records the traces and the default system services. Then, we compute the power spectral density $S_{qq}(f)$ over the frequency range $[0.5, f_m]$ Hz for each channel, with $f_m = 250$ for Haswell-i7 and $f_m = 100$ for ARMv7-Mobile, which is limited by the lower sampling rate. After subtracting the mean value from the temperature traces, to remove the DC component, we get the spectra through Fast Fourier Transforms (FFTs) [CLW69] of each temperature trace. To improve the accuracy of our analysis, we use Welch's method [Wel67] and a Blackman-Harris window [AST89]. Welch's method is commonly used to minimises the variability in the calculation of the power spectral density, i. e., the noise in the power spectrum, compared to standard Fourier analysis. The Blackman-Harris window is designed to minimise the side-lobes in the frequency domain and, therefore, the influence of neighbouring frequencies on each other. We report the resulting high-resolution noise spectra in Figure 3.9, together with the channel spectra S_{hh}, which we illustrate next.

Channel spectra. Determining S_{hh} is more challenging because of the constraint on the input. This constraint basically restricts the variety of input signals that we can use to rectangular waves of different frequency, similar to the one we used in the preliminary experiment of Figure 3.6. Our approach to determining S_{hh} consists of designing a set of experiments $\{\mathscr{E}_f\}$ where experiment $\mathscr{E}_f$ gives us an estimate of the value of the channel power $S_{hh}(f)$ at frequency f. We go into detail in the example of Figure 3.8, which illustrates how we determine S_{hh} for the 1-hop channel of core 1 of Haswell-i7. The data used to draw Figure 3.8 come from five separate experiments

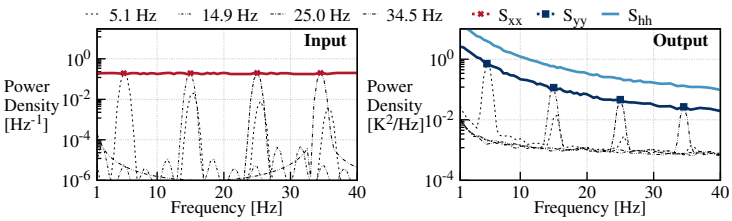


Figure 3.8: Input (left) and output (right) spectra from core 1 of Haswell-i7 for the five experiments $\mathscr{E}_f$ at the frequencies f reported in the legend. We use the spectra peaks to build S_{xx} and S_{yy}; then, $S_{hh} = S_{yy}/S_{xx}$. The y-axis is in logarithmic scale.

$\mathscr{E}_f$, with $f \in \{5.1, 14.9, 25.0, 34.5, 45.5\}$ Hz. Each experiment $\mathscr{E}_f$ consists of using a modified version of the source app to excite the system with a square wave at frequency f and computing the power spectra of the input and the output, which are superimposed in the left and right plots of Figure 3.8, respectively. To compute these spectra, we use the same FFT-based method that we use to compute S_{qq}. The spectra from experiment $\mathscr{E}_f$ show a peak at frequency f, which is where most of the power is allocated. We take these peaks as the values of the input S_{xx} (blue circles in Figure 3.8) and output S_{yy} (green triangles in Figure 3.8) power spectra. Then, we can simply compute the power spectrum of the channel S_{hh} as the sample-wise output-over-input ratio S_{yy}/S_{xx}. Figure 3.9 reports the values of the S_{hh} spectra that we derive with this methodology for the four channels on our two platforms, along with the noise spectra S_{qq}.

3.5.2.1 Additional notes on the experiments $\{\mathscr{E}_f\}$

Each experiment $\mathscr{E}_f$ lasts 120 s on Haswell-i7 and Sandy-Xeon, while on ARMv7-Mobile and ARMv8-Dev it lasts for 600 s, so that we collect the same number of samples (120 k) for all

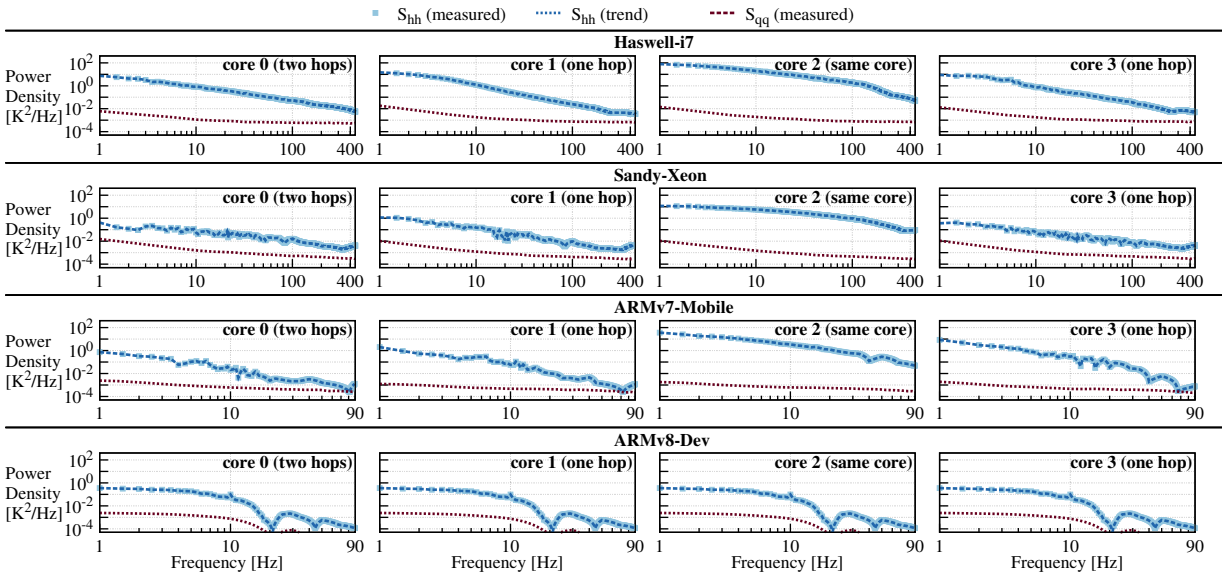


Figure 3.9: Power density spectra S_{hh} for the four channels measured on our platforms. The crosses are measured values, and the red solid line is the Bezier trend for S_{hh}. The dotted grey lines are the spectra of the noise S_{qq}. Both axes are in logarithmic scale.

platforms. The longer experiments on ARMv7-Mobile and ARMv8-Dev also help to make sure that we can actually observe enough variations in the temperature traces to build a meaningful spectrum (recall the accentuated quantisation effect on ARMv7-Mobile and ARMv8-Dev that was discussed in subsection 3.4.3). Finally, for all the channels, we only keep the S_{hh} points up to the frequency f where $S_{hh}(f)$ drops at or below the noise level $S_{qq}(f)$.

We determine the frequency range $\{f\}$ for the experiments $\{\mathscr{E}_f\}$ to reduce measurement errors as much as possible. We only use frequencies that, at the sampling period of either 1 ms (Haswell-i7) or 5 ms (ARMv7-Mobile), have an integer number of samples per period of the square wave. We start from 0.5 Hz and proceed in steps of either 0.2 Hz or one fewer sample per period, whichever yields the largest step. The crosses in Figure 3.9 are located at these frequencies along the x-axis. In total, we evaluate 180 different frequencies for Haswell-i7 and Sandy-Xeon; 78 different frequencies for ARMv7-Mobile and ARMv8-Dev.

Due to the constraints on the input, we use square waves as an approximation of sine waves, which would be the most appropriate waveform to concentrate the input power at the corresponding frequency. In practice, the non-ideal characteristics of our channels (particularly, the c-state sleep/wakeup latency) ensure that our *logical* square waves are really steep ramps that approximate a sine wave well enough. In fact, the spectra of Figure 3.8 clearly show the peaks at the fundamental frequencies, with some negligible harmonics.

One way to approximate sine waves better on the input would be to use active/sleep Pulse-Width-Modulation (PWM) at a rate r much higher than the frequency corresponding to the sampling time T we use (i. e., $r \gg 1\text{KHz}$). In this way, it is possible to obtain different power levels and to generate a

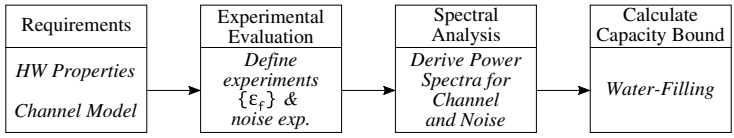


Figure 3.10: Summary of the steps necessary to determine upper channel capacity bounds for complex value and time continuous covert channels.

sampled sine wave. Since the c-state and scheduling latencies are fast enough to do so, we actually implemented this PWM approach in a modified version of the source app. However, we found that the results were not significantly different; thus, we decided to stick with the "square" waves.

3.5.3 Computing the capacity bounds

We can finally compute the two capacity bounds C_b and C_a, with the classic and constrained-input water-filling methods, respectively. Figure 3.10 summarises the necessary steps for deriving the capacity bounds. Since we work with discrete spectra, we accordingly adapt the equations of subsection 3.5.1 to use summations instead of integrals and to consider the discretisation intervals along the frequency range. While the noise spectra S_{qq} already come with a high frequency resolution, the S_{hh} spectra are more coarsely quantised, as the crosses in Figure 3.8 show. To simplify the computations, we linearly interpolate all the spectra on a regular frequency grid with 0.1 Hz spacing.

Classic water-filling. This method can handle non-white noise spectra S_{qq}, which is the case in our measurements (see Figure 3.9). We define the input power cap p_0 as the maximum input power measured in our experiments $\{\mathscr{E}_f\}$,

see Equation (3.6). To find C_b, we compute the ideal power

S^e_{xx} ... Input power spectrum of experiment $e \in \{\mathscr{E}_f\}$

$$p_0 = \max_{e \in \{\mathscr{E}_f\}} \left(\int_f (S^e_{xx}) \right) \tag{3.6}$$

allocation $\hat{S}_{xx}$ by iteratively refining the value of the parameter λ until the condition of Equation (3.5) is met (almost) with equality (with a maximum error of 10^{-6}).

Constrained-input water-filling. In order to compute C_a, Equation (3.4) assumes that the additive noise is white, with constant power density $S_{qq} = N_0$ across the relevant frequency range. However, our measured S_{qq} spectra vary significantly across the frequency range we are interested in. To address this issue, we apply a whitening filter to all spectra. Assuming the

$$S_{WF} = \frac{S_{qq}}{N_0} = \frac{S_q q}{\overline{S_{qq}}} \tag{3.7}$$

$$\hat{S}_{hh} = \frac{S_{hh}}{S_{WF}} \tag{3.8}$$

$$\hat{S}_{qq} = \frac{S_{qq}}{S_{WF}} \tag{3.9}$$

$$\hat{N}_0 = \overline{\hat{S}_{qq}} \tag{3.10}$$

constant noise power N_0 as $\overline{S_{qq}}$, we determine the spectrum of the whitening filter S_{WF} as shown in Equation (3.7). We now apply this whitening filter S_{WF} to our channel to get the whitened channel spectrum $\hat{S}_{hh}$ and the whitened noise spectrum $\hat{S}_{qq}$ as illustrated in Equation (3.8) and (3.9). The white noise level $\hat{N}_0$ is defined in Equation (3.10). We adopt

the Equations (3.4) and (3.5) as illustrated in Equation (3.11) and (3.12), to determine the capacity of the channel. Given

$$C = \max_{\lambda} \left\{ \frac{1}{2} \int_{A_\lambda} \log_2 \left(\lambda \cdot \hat{S}_{hh}(f) \right) df \right\} \text{ [bps]} \tag{3.11}$$

$$\frac{1}{2} \int_{A_\lambda} \left(\lambda - \frac{1}{\hat{S}_{hh}(f)} \right) df \leq \frac{p_0}{\hat{N}_0} \tag{3.12}$$

the global power cap p_0, which we determine in the same way as in the classic water-filling case, we compute the optimal allocation to the sub-bands based on their width and their noise level [VPL10, Chap. 6.5]. Finally, we consider one sub-band at a time and independently compute the capacity in an iterative way, similar to the classic case. To compute C_a, we sum the resulting capacity in all the sub-bands.

Capacity bounds. Figure 3.11 shows the capacity bounds C_b (upper) and C_a (lower) that we compute with the classic and constrained-input water-filling methods, respectively. In general, the observations of subsection 3.4.3 seem to comply with the determined capacity bounds, i. e., the channel on core 3 is better than the one on core 1 for ARMv7-Mobile, while the opposite is true for Haswell-i7 and Sandy-Xeon. As expected, $C_b > C_a$ and the bound for the same-core channel is the highest all platforms and both methods. Also, the trend that capacity bounds reduce drastically for m-hop channels with $m > 0$ compared to the same-core channels is consistent among all platforms except ARMv8-Dev. Here, ARMv8-Dev is a special case, as it supplies the same sensor reading for all of the cores, as outlined in section 3.4.

While for Sandy-Xeon and ARMv7-Mobile the capacity bound decreases with increasing hop count, the 2-hop channel on Haswell-i7 yields a higher channel capacity bound than the

1-hop channels. This can be explained by closely reviewing the spectra for Haswell-i7 illustrated in Figure 3.9. While the signal power reduces slightly when comparing core 0 to *core* 1 and 3, the noise power is significantly lower on core 0.

These results do not exclude that the same-core channel might be a security threat, with C_a well above 100 bps for three of the four platforms. While the bounds for the 1-hop channels and channels on ARMv8-Dev are all below 100 bps, they are still much higher than our initial expectations based on previous research. In section 3.6 we show a transmission scheme able to increase previous results on transmission rates notably.

3.6 Transmission scheme and achieved rates

The transmission scheme that Masti et al. [Mas+15] used to evaluate the 1-hop channel is based on *ON-OFF* keying: the source app is active to transmit a 1 and it goes idle to transmit a 0. A major issue with this simple scheme is that the average load level depends on the input message: a message with several *ones* (respectively, *zeros*) in a row will leave the source core active (respectively, idle) for a long time compared to the symbol duration, causing the average temperature to drift up and down. This drift of the operating point unpredictably changes the temperature dynamics over time, making the channel non-stationary and the decoding more complicated. This issue, coupled with the simplistic edge-detection decoding method they used, could explain the poor performance that they measured compared to the capacity bounds that we derived in section 3.5. In this section, we evaluate a simple communication scheme that overcomes this issue.

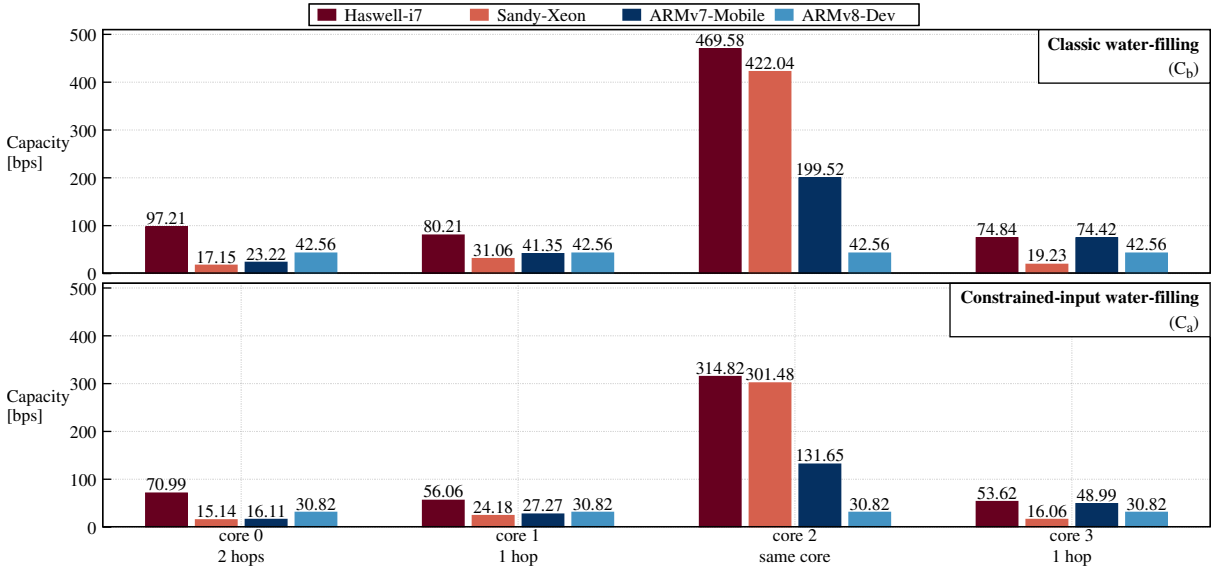


Figure 3.11: Upper bounds C_b (top) and C_a (top) on the channel capacity C. Except for ARMv8-Dev, the capacity degrades drastically for m-hop channels with $m > 0$, compared to same-core channels. The constrained-input water-filling yields tighter capacity bounds.

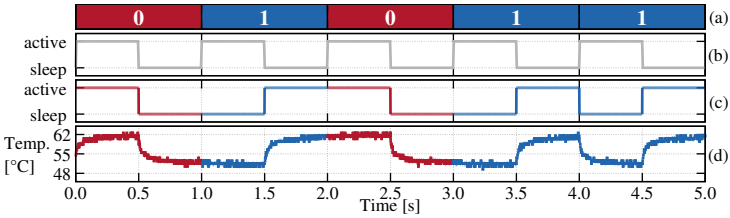


Figure 3.12: An input message (a), encoded onto the 1 Hz clock (b), gives the execution trace (c), which leads to the temperature trace (d) on the same-core channel of Haswell-i7.

3.6.1 Encoding and decoding scheme

A simple way to keep the channel in the dynamic range during communication is to encode the input message so as to maintain, on average, a constant load. To do so, we use square waves with a 50% duty cycle as a *clock* signal onto which we *encode* the input message.

Message encoding. We generate the execution trace of the source app with the *Manchester encoding* scheme [Tan02], as Figure 3.12 illustrates for a 5-bit message and a 1 Hz clock. A *one* in the message is encoded into an unmodified clock signal in the execution trace; a *zero* becomes a 180° phase-shifted clock signal in the execution trace. The resulting execution trace leads to temperature traces oscillating around a roughly constant average, as Figure 3.12 (d) shows for the same-core channel on Haswell-i7. The transmission rate directly depends on the frequency of the clock signal since the trace carries 1 bit of information per period of the clock, i. e., r bits per second (bps) for a r Hz clock.

Message decoding. Message decoding happens offline (see section 3.2) from the temperature traces recorded by the sink app. The first step of decoding is determining the phase of the clock signal. For simplicity, we synchronise our experiments

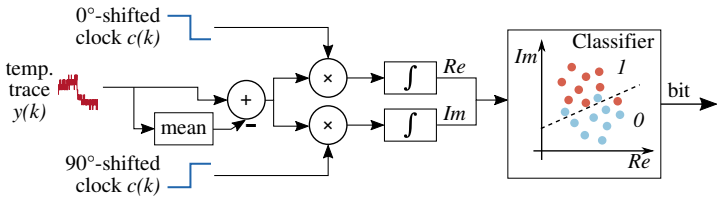


Figure 3.13: Block diagram of our bit-wise decoding scheme.

so that the beginning of the temperature trace coincides with the beginning of the message. In a real attack, where this synchronisation would not be possible, the source app could send a known preamble that the sink app could use to detect the clock phase. Once the clock phase is detected, it will not change during an experiment, since our source and sink app are designed to not accumulate clock skew (see subsection 3.4.2). To proceed with decoding, we look at each clock period, i. e., at each bit, separately. As Figure 3.13 shows, for each bit, we first get a 0-mean signal by subtracting its mean temperature. In this way, the decoding is robust against long-term temperature variations due to environmental changes. We decode the resulting trace with traditional signal-processing techniques [VPL10]. We first multiply the trace with a 90° and a 0° phase-shifted clock signals and we integrate over the two resulting signals ($\int$ blocks in Figure 3.13). The two resulting numbers are the real (*Re*) and imaginary (*Im*) parts of a representation of the bit in the complex plane $\mathbb{C}$. To classify each bit as a 1 or a 0 in this signal space, we use a linear support vector classification[6] previously trained on data from the same platform.

[6]We use the sklearn python module using a data pipeline consisting of a standard scaler and a LinearSVC.

3.6.2 Performance evaluation

To evaluate our transmission scheme, we encode several random messages onto clock signals at different frequencies and we use our source and sink app to transmit and record these messages on our two platforms, configured according to the reference setup of section 3.4. We decode the temperature trace from each channel with our classifier. As the performance indicator, we use the error probability, as measured through the empirical bit error probability, i. e., the relative number of misclassified bits. We only report raw transmission rates and error probabilities and do not evaluate error correction strategies. We leave such study to future work.

Error probability at increasing rates. As a first test, we generate a 1500 bit and a 5000 bit message and we evaluate the error probability of our channels at increasing transmission rates, from 1 bps in 1 bps steps up to 100 bps, and 5 bps steps for higher rates. For each channel, we use the 1500 bit message to train the classifier, which we evaluate on decoding the 5000 bit message. In a real attack, the source app could first transmit a known message that the sink app could use for training the classifier and then the actual information, which the sink app could decode with the trained classifier. Figure 3.14 shows the resulting error probability (measurements and Bezier trends) for the four channels on our two platforms. For the two x86_64 based platforms Haswell-i7 and Sandy-Xeon, the same-core channel shows very few errors ($\ll$ 1%) up to $\approx$ 90 bps; the error plot in logarithmic scale in Figure 3.15 illustrates this. The error probability is much higher on the two Arm-based platforms, ARMv7-Mobile and ARMv8-Dev. While on ARMv7-Mobile the error stays below 1% up to approximately 30 bps, ARMv8-Dev only allows low error transmission up to 5 bps. At increased rates, the errors increase more slowly on the x86_64 based platforms, where we achieve $\approx$ 175 bps at 10% error

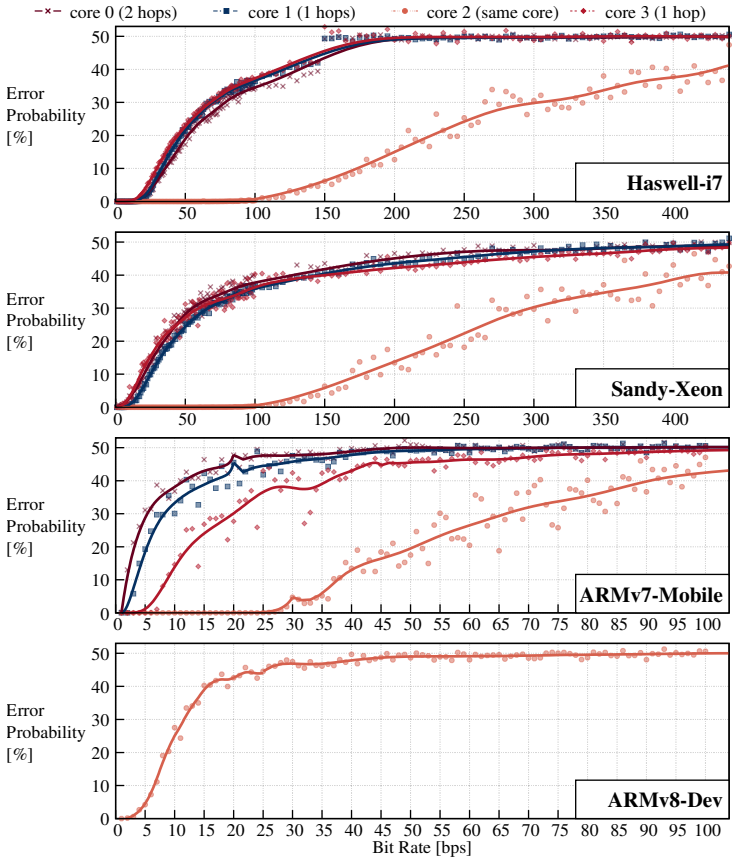


Figure 3.14: Error probability on decoding a 5000 bit random message for the four channels on all platforms. The x86_64 based platforms Haswell-i7 and Sandy-Xeon perform similarly, while the performance is notably worse on the Arm-based platforms ARMv7-Mobile and ARMv8-Dev.

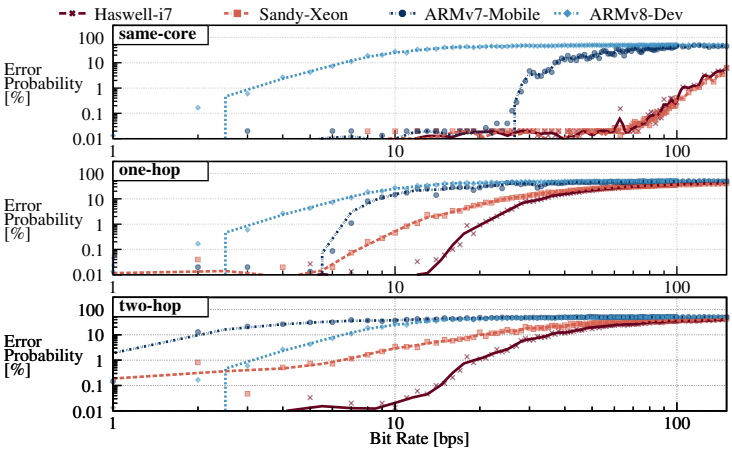


Figure 3.15: Logarithmic illustration of Figure 3.14; showing one channel type per plot. For the 1-hop channels, for Haswell-i7 and Sandy-Xeon core 1 is illustrated and for ARMv7-Mobile core 33.

probability, than on the Arm-based platforms. On ARMv7-Mobile, the transmission rate is $\approx$ 40 bps, and $\approx$ 7 bps on ARMv8-Dev, at the same error level, because on the Arm-based platforms signals with higher frequency suffer from more attenuation. Haswell-i7 also show better performance for the 1-hop and 2-hop channels, where the error probability remains below 1% up to $\approx$ 10 bps and hits the 10% level between 40 bps and 50 bps. On Sandy-Xeon, the 1-hop and 2-hop channels perform slightly worse than on Haswell-i7 up to the 10% error probability threshold, after which the performance is comparable. We assume this is caused by the higher signal attenuation for lower frequencies on Sandy-Xeon, compared to Haswell-i7. Another interesting effect is that on Haswell-i7 and Sandy-Xeon, the 2-hop channel does not perform much worse than the 1-hop channels. Instead, on ARMv7-Mobile

the error probability immediately increases steeply, and the performance gaps are more evident, with the 2-hop channel showing several errors already below 5 bps. These results relate to the stronger quantisation effect and the higher attenuation for these two channels on ARMv7-Mobile.

While directly comparing these results with the capacity bounds of Figure 3.11 is not rigorous since we are not considering the overhead of error correction, we can observe that the Arm-based platforms ARMv7-Mobile and ARMv8-Dev generally perform worse than the x86_64 based platforms Haswell-i7 and Sandy-Xeon when compared to the capacity bounds. In fact, for the capacity study, we hid the negative effects of quantisation on ARMv7-Mobile and ARMv8-Dev through longer experiments (see subsection 3.5.2), while our transmission scheme is oblivious to this effect. A better transmission scheme for ARMv7-Mobile and ARMv8-Dev might leverage temperature observations in the source app in order to tune the duty cycle of the clock signal so as to bring the average temperature at a quantisation boundary, thus making the small 1 K variations visible and reducing the errors at high rates.

Direct comparison with previous work. To evaluate our transmission scheme against the naïve ON-OFF keying scheme used by Masti et al. [Mas+15], we provide a direct comparison. We both evaluate our scheme on the exact same platform they used[7] and implement the ON-OFF keying scheme in our framework to evaluate it on Haswell-i7, Sandy-Xeon, ARMv7-Mobile and ARMv8-Dev. Figure 3.16 shows all these results for the 1-hop channel. For all platforms, we plot the best of the two 1-hop channels. The solid lines show the results we obtained with our scheme on the four platforms (Haswell-i7,

[7]The same type dual-socket server with two Intel Xeon E5-2690 multicores clocked at 2.90 GHz as Sandy-Xeon

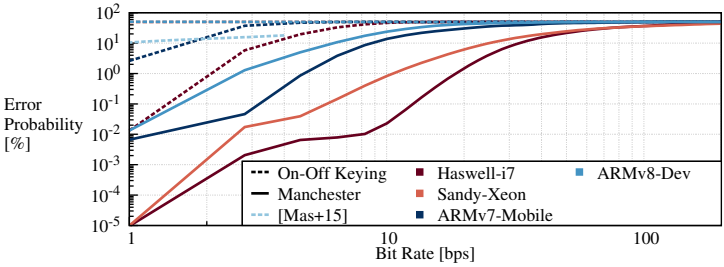


Figure 3.16: Direct comparison with Masti et al. [Mas+15, Tab. 1] for the 1-hop channel. The solid lines show the results with our scheme (see subsection 3.6.1), the dashed lines show the results reported by Masti et al. [Mas+15] on Sandy-Xeon and the results we obtained using their same scheme (ON-OFF keying). Our scheme outperforms their scheme on all platforms.

Sandy-Xeon, ARMv7-Mobile, and ARMv8-Dev). The dashed lines show the results that we obtained on the four platforms by implementing the ON-OFF keying scheme, as described in Masti et al. [Mas+15]. However, in contrast to the edge detection decoder used by Masti et al. [Mas+15], we use a threshold detection. For each symbol duration, we determine the median temperature and then feed it to a previously trained linear support vector classifier to decode the symbol. In addition to our experiments, in Figure 3.16 we also report the original results from this previous work [Mas+15, Tab. 1], which only covers a smaller range of bit rates. As Figure 3.16 shows, on Sandy-Xeon, our results with ON-OFF keying are very close to the original ones. Our scheme achieves 10 bps at about 0.1% error probability, while ON-OFF keying does not go below 1% error probability at 1 bps. On all platforms except Haswell-i7, establishing communication with ON-OFF keying proves virtually impossible due to symbol coding scheme, our

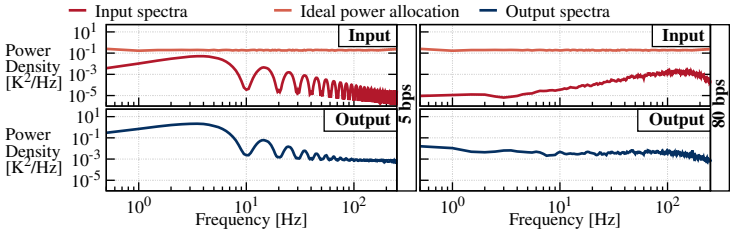


Figure 3.17: Input and output (same-core channel on Haswell-i7) power spectra of the evaluation sequences at 5 bps and 80 bps, compared to the ideal water-filling power allocation.

scheme proves more robust and obtains better results than on Sandy-Xeon. From this extensive comparison, we conclude that our transmission scheme ensures much better performance than ON-OFF keying in all cases.

Spectral efficiency. To get a feeling of whether our scheme could be further improved, Figure 3.17 compares the input (green, top) and output (red, bottom) power spectra of the 5000 bit evaluation sequences at 5 bps and 170 bps with the ideal water-filling power allocation S_{xx} for the same-core channel on Haswell-i7. The comparison is purely indicative since the water-filling solution only gives an upper bound on the capacity of our channels (see subsection 3.5.1), but it is interesting nonetheless. On the one hand, the 5 bps input spectrum allocates much power at low frequency, resulting in very little distortion in the output spectrum in that area, which is where most information is encoded. On the other hand, the 170 bps input spectrum shifts most of the power at a higher frequency, leading to visible distortion in the output spectrum due to the noise which, as Figure 3.9 shows, is stronger at lower frequencies. A better scheme should have a levelled input power allocation across the spectrum. Finding such a scheme, despite the limitations on the input, is an interesting

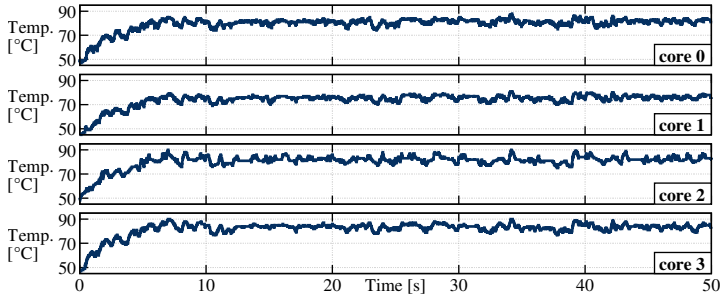


Figure 3.18: The temperature of ARMv7-Mobile increases drastically until the Dynamic Thermal Management (DTM) throttles the device. Ultimately this leads to an OS freeze, which makes it impossible to perform experiments with heavy interference using the ffmpeg application.

challenge for future work.

3.6.3 Sensitivity to environmental conditions

To assess the robustness of the thermal covert channel, we evaluate how variations in the environmental conditions affect the error probability on our channels. First, we perform a general interference evaluation across all platforms. Therefore, concurrent with the transmission, we employ the ffmpeg application to generate interference, similarly to chapter 2. We configure ffmpeg to run an infinite loop to convert a 9.56 minutes long animated mp4 video[8] and pipe the output to `/dev/null`.

On ARMv7-Mobile the high load of the ffmpeg application causes a drastic increase in the core temperatures, such that the DTM is triggered. This behaviour is illustrated in Figure 3.18. Unfortunately, as a final consequence, the ARMv7-Mobile

[8] `peach.blender.org`

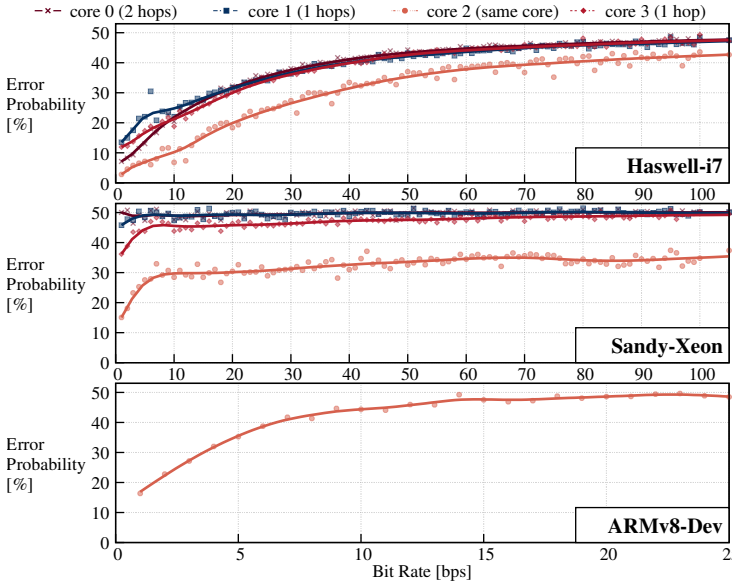


Figure 3.19: Error probability on decoding a 5000 bit random message for the four channels on all platforms with a concurrently running interference application.

platform freezes and further data collection is not possible. Therefore, we exclude ARMv7-Mobile from the interference study.

Figure 3.19 illustrates the error probability on decoding a 5000 bit random message with heavy ffmpeg interference. Compared to the results without interference in Figure 3.14 the performance of the thermal covert channel notably degrades on all platforms. While on Sandy-Xeon and ARMv8-Dev, no transmissions with error probabilities lower than 10% are possible, on Haswell-i7 the same-core channel allow transmission of up to 10 bps for the same core channel. Judging from these results, a thermal covert channel transmission

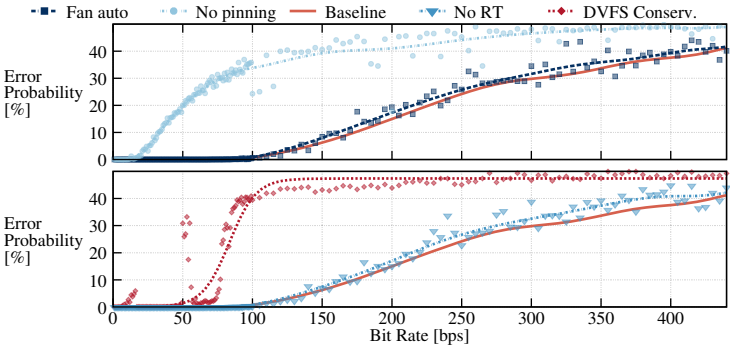


Figure 3.20: Sensitivity of the error probability to using automatic fan speed, not pinning the apps to cores, no real-time scheduling, or the conservative Linux DVFS governor.

would still be possible under noisy conditions on Haswell-i7. Therefore, we perform a more detailed study of the different environmental influence factors on for the same-core channel on Haswell-i7, as a representative case.

We identify four important parameters that, in a real attack, would not be fixed as in our experimental setup (section 3.4) and we evaluate the sensitivity of our results to variations of these parameters. Figure 3.20 shows how the error probability is affected when changing these four parameters in the experimental setup:

1. Setting the fan speed to automatic (Fan auto);
2. not pinning the apps to a specific core (No pinning);
3. using the default, Linux scheduling policy (`SCHED_OTHER`) instead of the high-priority `SCHED_FIFO` (No RT);
4. letting the ondemand Linux DVFS governor change the frequency of the cores (DVFS Ondemand.).

These four parameters have different impacts on our baseline results, represented in Figure 3.20 by the solid light blue line.

Automatic fan speed. Using a variable, automatic fan speed highly affects the channel and makes it more chaotic. This result is intuitive, as the fan controller is designed to keep the temperature on a low constant level, hindering the possibility to encode data in temperature variations.

Conservative DVFS governor. Enabling DVFS has a strong effect on the communication channel, which becomes highly unstable and chaotic. This result is due to the fact that the active frequency of the cores largely determines the active power consumption, and thus temperature. Notice, however, that since, on many platforms, multiple active cores run at the same frequency, load-level based frequency scaling (which the Linux conservative governor implements) might enable another covert channel, where the sink app observes frequency variations induced by the source app. We explore such a covert channel in chapter 5.

No real-time priority. Similar to variable fan speed, real-time priority affects the error probability only for rates faster than $\approx$ 100 bps. The additional errors are due to increased jitter in the timing of the source and sink apps. Figure 3.21 further investigates this effect by analysing the jitter in the state transitions of the source app when running a 100s random trace with our baseline setup (pin, rt) and when dropping real-time priority (nort) or thread pinning (nopin). We repeat the experiments with different levels of system load, which we simulate using the `stress-ng` app[9]. At low load, dropping real-time priority causes the jitter to increase to $\approx 80\,\mu$s in $\approx 50\%$ of the transitions; the sink app is similarly affected in the

[9]We generate the system load using the bash command `stress-ng -c 0 -l $LOAD`, where `$LOAD` is the system load in percent.

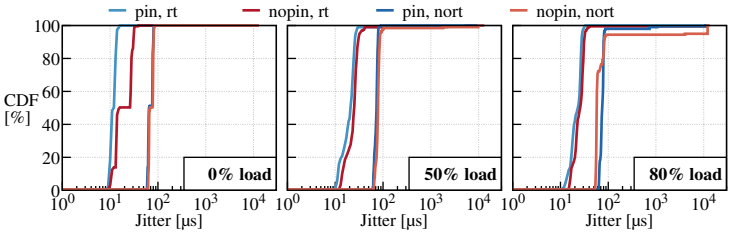


Figure 3.21: Cumulative distribution functions of the transition jitter of the source app on Haswell-i7 with or without real-time scheduling ([no]rt) and thread pinning ([no]pin) and with different background load.

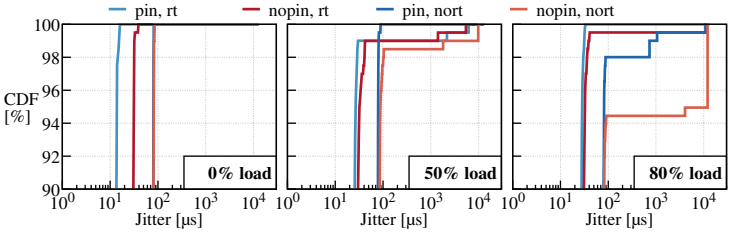


Figure 3.22: Zoom in illustration of Figure 3.21.

precision of its sampling rate. Figure 3.20 shows that this effect only starts impairing the performance of our scheme at rates faster than ≈ 160 bps. Figure 3.22 illustrates that the jitter is higher at increased load, but it does not exceed 0.1 ms for 94% of the transitions at 80% load for the nopin, nort case; thus, error correction should still enable communication at low rates even with system load.

No thread pinning. When the source and sink apps are not pinned to a specific core, the different channels effectively move with the source app. As an example, Figure 3.23 shows part of a trace from Haswell-i7 where the source app, which is transmitting a 1 bps data signal, migrates between cores 0 and

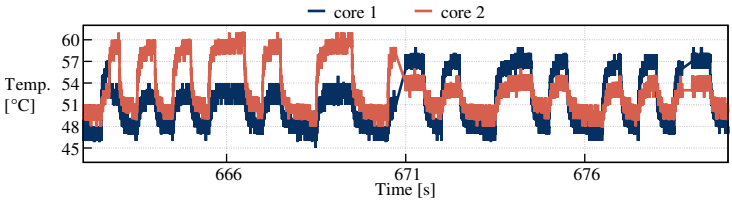


Figure 3.23: Traces from cores 0 and 1 of Haswell-i7; the source app is not pinned. The source application is migrated from core 0 to 1 at ≈ 671 s.

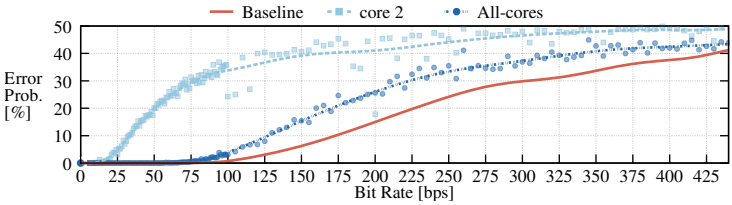


Figure 3.24: Same-core vs. all-cores channel comparison with no pinning on Haswell-i7.

1. Initially, reading the temperature from core 2 corresponds to a same-core channel, while it becomes a 1-hop channel at time ≈ 671 s, when the source app migrates to core 1. Furthermore, Figure 3.23 illustrates that at the time of the core migration, measurement artefacts may occur and the thermal dynamics might be different for different cores. As Figure 3.24 shows, if the sink app always observes the same core (core 2 on Haswell-i7 in this case), the error probability without thread pinning will sensibly increase compared to the baseline since the channel type keeps changing. However, there is a simple way to work around this issue. Since the sink app can always read the temperature of all the cores, we can simply look at the *all-cores channel*, which is the sum of the temperatures from all cores. As Figure 3.24 shows, the all-

cores channel has performance comparable than the same-core channel up to 80 bps. For higher rates, more errors occur due to an increased amount of measurement artefacts and presumably lower thermal dynamics due to the use of different cores by the source application.

We conclude that our communication scheme is robust to disabling thread pinning and, to some extent, to dropping real-time priorities and having background system load. The most sensitive parameters are varying fan speed and enabling the DVFS governor, which makes communication impossible with our scheme but might enable a different covert channel when all cores share the same active frequency.

3.6.4 Deployment test

As a final step, we set up a realistic scenario to perform a deployment test. We intend to leak the *id_rsa* private SSH key file which contains ASCII characters translating into 13432 bits.

Setup. We employ a Lenovo ThinkPad T460s with an Intel Core i7-6600U CPU with two physical cores, each running two hyph-threads, resulting in four logical cores. The laptop runs Ubuntu 15.10 as a host OS and as a virtual machine client in an Orcale VM VirtualBox. The virtual machine is configured to emulate two cores and is able to fully utilise the host CPU (execution cap of 100%). Contrary to the controlled laboratory setup, the laptop uses the `powersave` frequency governor and the completely fair scheduler (CFS). The source application is located within the virtual machine, while the sink application is running on the host OS. For our evaluation, we let the laptop sit on an office table without actively using it. Thus, we do not need to differentiate between busy and idle periods to start a transmission. We deem this scenario to be valid, as laptops are often kept active in offices during the night or the weekend.

Signal coding. We empirically evaluated the transmission parameters on the target setup. We found that a Manchester encoding that uses a duty cycle of 70% outperforms the standard version with 50% duty cycle, which we used in the laboratory setup. Here, duty cycle defines the ratio of the symbol duration in which the source application worker threats are active. Furthermore, we fixed the transmission rate to 17 bps.

Application synchronisation. We assume that the system clock of the virtual machine and the host operating system are synchronised. Therefore, we define a rendezvous points between src and snk every 30 minutes at the .00 and .30 hour mark, independent of the system utilisation.

Data coding. We zero pad the data to 13440 bit and split it into 840 packets; each packet contains a 2.7 bit header pulse, 16 bits data load and 4 bits for error-detection. The header pulse is generated by the source application by fully utilising the cores for the duration of 2.7 bits. We use this header pulse to forces the system to scale the operating frequency to a higher level, which increases the thermal dynamics due to the higher power dissipation of the cores. Furthermore, the header pulse as an additional indicator for the message start and is used in the data processing to synchronise the thermal stream before decoding it. Including all the overhead, the data that needs to be leaked totals to 19068 bits, which takes approximately 18 minutes and 42 seconds to transmit.

Results. After having received 9 complete packet sets after approximately 2 hours and 49 minutes, we were able to successfully recover the SSH key file from the thermal data. Hence, such an attack can be carried out during the night when the target device is not active. This deployment illustrates the feasibility of a thermal covert channel attack under realistic conditions.

3.6.5 Threat mitigation

As we reported in Sections 3.1 and 3.2, the on-chip temperature sensors that enable the thermal covert channels we studied are easily accessible by user-level apps on current multicore systems. A technically simple way to block the potential threats coming from these channels is to restrict access to the temperature sensors to trusted code. If temperature information needs to be made available to user apps (e.g., a CPU temperature monitor), viable mitigation strategies include increasing the refresh interval from milliseconds to seconds or minutes and reducing the sensor resolution, thus directly limiting the capacity of the thermal covert channels. While mitigating this threat is not technically challenging, it requires shipping security patches to a huge base of affected devices running different versions of different system software stacks. Our aim with this chapter was to raise awareness of the potential threat that current systems are exposed to and provide a quantitative study that can be used as a base to decide what actions to take in order to mitigate this threat.

3.7 Summary

In this chapter, we analysed a family of covert channels where a source app induces temperature variations on a multicore processor, and the sink app observes these changes through the on-chip temperature sensors. Our two main contributions with this chapter are providing upper bounds on the capacity of these channels and showing a transmission scheme that improves previous results on communication rates by more than 20×. Based on experimental data from two diverse platforms representative of ch3:item:laptops and smartphones, we derived capacity bounds by leveraging information theory

and spectral analysis. Based on our results, we cannot exclude the possibility that these channels might be a security issue, as the capacity could be in the order of 300 bps for the same-core channel. We presented a transmission scheme based on Manchester encoding that sensibly improves the performance of previous work and studied the sensitivity of our results to non-ideal conditions. With this scheme, we were able to achieve rates of more than 100 bps on the same-core channel and more than 10 bps on the 1-hop channel, with less than 1% error probability. Furthermore, we show that it is possible to establish a thermal covert channel to successfully leak an SSH key in a realistic scenario, where one application is running inside a virtual machine and the other running on the host system.

4

Analysing discrete covert channels

In the previous chapter, we illustrated how to use the methodology presented in chapter 2 to analyse continuous channels, like the thermal covert channel. However, some data leaks are established based on quantities that are discrete. As discrete quantities are not necessarily able to be differentiated over time and may only take values from a finite set, different models and capacity bound derivation methods are needed for their analysis. In this chapter, we present such a model and the corresponding channel capacity bound derivation method.

In addition, we present the novel power covert channel, which takes advantage of power readings provided in current Intel CPUs. We perform a detailed theoretical and experimental analysis in which we determine a channel model, derive the channel capacity bounds and present an exhaustive

This chapter is based on work presented in [MT18] and [Tho17].

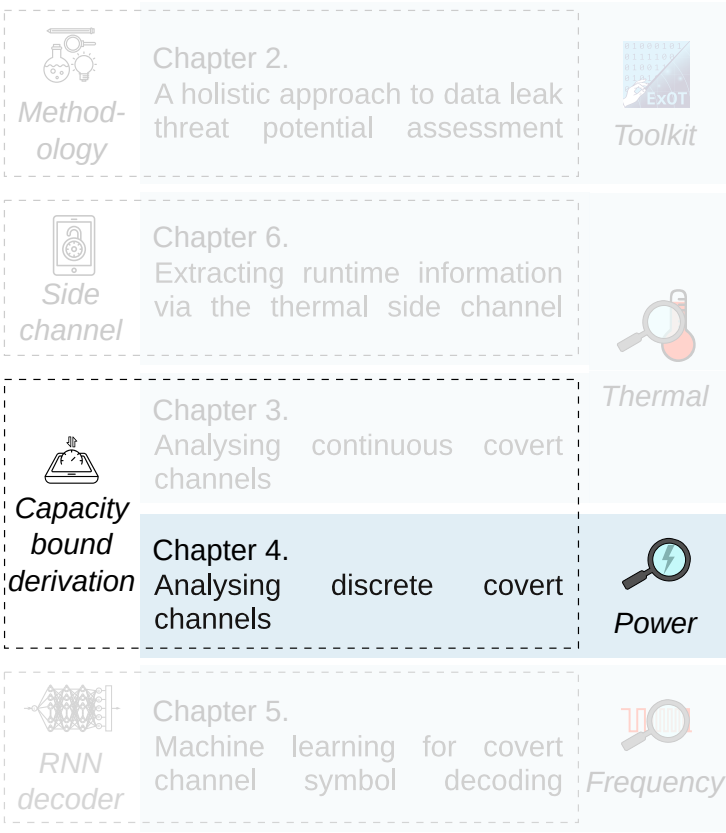


Figure 4.1: In chapter 4, we apply our data leak analysis methodology on value and time discrete covert channels. We illustrate how to model such covert channels and derive upper channel capacity bounds. Moreover, we present the novel power covert channel, which takes advantage of power readings provided by Intel CPUs.

experimental evaluation.

4.1 Introduction

Today, general-purpose and embedded multicore processors face two major challenges: (i) preventing overheating in the face of increasing maximal power density, and (ii) keeping a low average power dissipation despite the need for a high quality of service and large variability of the processing load. To deal with these problems, different power management schemes have been developed and deployed in computing systems. Implementations range from simple hardware-based Dynamic Thermal Management (DTM), which impose a high-performance penalty on the system, to sophisticated solutions relying on software implementation.

An example of a more sophisticated method is sleep states, which are specific power-saving modes that drastically reduce the power dissipation using, for example, clock gating or flushing and turning off the cache. Another popular technique, often deployed in addition to sleep states, is Dynamic Voltage and Frequency Scaling (DVFS). DVFS allows the Operating System (OS) to change the operating frequency and voltage of the processor cores according to the system utilisation. This demand-based control allows the system to reduce the power dissipation with little or no noticeable performance impact. On some architectures, DVFS also allows overclocking of the processor cores to deliver a higher performance for a limited time.

Yet, to prevent the system from exceeding its physical limitations when overclocking, the power management system requires more detailed information than the common performance counters. Such detailed information is offered by many processors, for example, in the form of temperature or power measurements. However, in a system where applications with different security clearances share the same cores, these measurements can be misused to compromise the system's

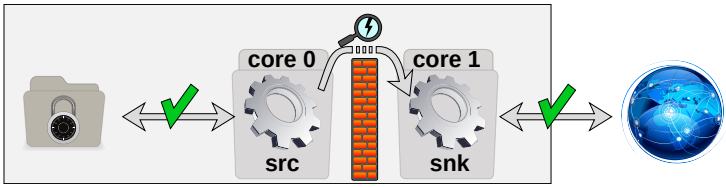


Figure 4.2: The source application (src) has access to restricted data, while the sink application (snk) has access to the communication interfaces. Although source and sink are isolated from each other, they manage to establish a covert channel by observing the processor power dissipation. This compromises the security paradigm of permission separation and application isolation.

security framework.

In this chapter, we investigate the potential security threats that arise from accessible processor power information by using power measurements on Intel cores. In particular, we consider a scenario similar to the one presented in Figure 4.2. Here, a simple dual core system is running only the basic OS services besides our two attack applications, the *source* application src and the *sink* application snk. The source application has access to highly sensitive information, i. e., cryptographic keys, but it cannot use any communication interfaces. In contrast, the sink application has no access to highly sensitive information, but it has access to the communication interfaces to send data to third-party servers. The two applications are isolated from each other, complying with the security paradigm of privilege separation and application isolation. In case of low system load, the source application may take advantage of the power management system to encode information by modulating the power dissipation. Now, the sink application is capable of logging the power dissipation and decoding and

forwarding the information to a third-party server for analysis. This constitutes the so-called *power covert channel* and violates the security paradigm of privilege separation and application isolation.

Contribution. In this chapter, we present a detailed study of a covert channel based on processor power information: the power covert channel. Our contributions are:

1. A generally applicable method to model and derive a tight channel capacity bound for discrete covert channels. The capacity bound helps to estimate the security threat caused by covert channels.

2. We present the novel power covert channel and provide models and capacity bounds for this data leak.

3. To the best of our knowledge, we are the first to show an implementation of a communication scheme that proves the functionality of the power covert channel on x86_64 based Intel platforms.

4.2 Power management in Linux

The power covert channel depends on variations of the power dissipation of the device cores. As we base our experimental setup on Linux systems, we will take a brief look into the two main parts of the Linux power management: the `cpufreq` and the `cpuidle` subsystem. We note that although we execute all our experiments on Linux, power measurements can also be read on other OSs like Windows or Mac OS. For example, Intel allows us to read power measurements using the Intel Power Gadget API[1].

[1] `https://software.intel.com/en-us/articles/intel-power-gadget-20`

cpufreq. The cpufreq subsystem is responsible for power management during the active time. Its main purpose is to scale the operating frequency and voltage according to the system utilisation, based on a so-called governor policy [PS06]. In our experimental evaluation, we will use the *acpi-cpufreq* driver, which offers various governor policies. Among the most notable governor policies are the *conservative* and the *userspace* governor. The conservative governor automatically scales the operating frequency of the cores stepwise up or down, depending on whether the utilisation is above or below certain thresholds. In contrast, the userspace governor allows direct control of the operating frequency from the userspace via *sysfs* nodes. For more detailed information on the cpufreq subsystem, please refer to chapter 5.

cpuidle. The cpuidle subsystem controls the sleep mechanisms of the device. Similar to cpufreq, the behaviour of cpuidle is also based on governor policies [PLB07]. Whenever a core is not utilised, cpuidle decides to which sleep state, also called *C-state*, that core is sent. Deeper C-states have a higher power saving effect but also have a higher exit latency. The standard implementation of cpuidle offers two governor policies, the *ladder* and the *menu* governor. While the ladder governor selects the C-state with a step-wise approach, moving down from the shallowest to the deepest C-state, the menu governor is more sophisticated. The menu governor selects the deepest possible C-state by evaluating various parameters, like the expected core sleep time, latency requirements or the last C-state used by the core.

Power Measurements. In order to optimise the power dissipation and maximise the performance of the processor up to its physical limits, power measurements are needed in addition to thermal and system utilisation measurements. The granularity of the power measurements depends on the

platform. For instance, the platforms used in this chapter provide only one power measurement for the entire multicore processor. An example of the use of power measurements for performance optimisation is the *Intel Turbo Boost*. It allows processor overclocking in case of high-performance need and sufficient overclocking budget. The overclocking budget is determined using power and thermal measurements, which are used to determine how long a processor can be operated at a frequency higher than specified rated operating frequency without overheating.

4.3 Channel model

To apply the methodology we presented in chapter 2 to the power covert channel, we first need to define a channel model. The proposed system abstraction model in Figure 4.3 is composed of three parts: the Input Stage, the power covert channel and the Output Stage.

Input stage. The encoder performs channel coding to convert the input bitstream b_i with bits $b_i[k]$ into the input symbol trace x, containing symbols $x[k]$. One symbol can, for example, represent a specific power dissipation of the processor. The set of feasible symbols depends on the granularity of the power measurements, i. e., the number of observable levels. The input symbol trace x is transferred to the power covert channel.

Power covert channel. In the proposed system model, the source and the sink applications, `src` and `snk`, are part of the power covert channel. Input to `src` is the input symbol trace x generated by the encoder component. The source application `src` converts the input symbol stream x to a utilisation trace x_U with core utilisations $x_U[k]$ and applies these utilisations at runtime. The power trace y_p with power values $y_p[k]$ is obtained as

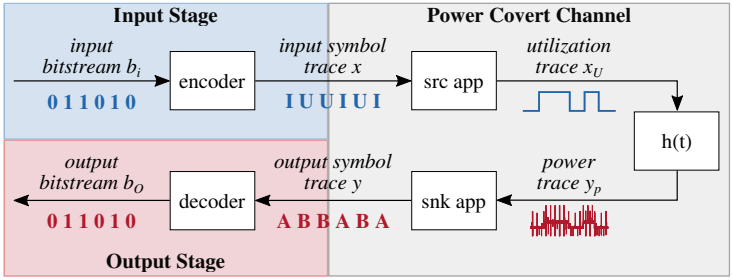


Figure 4.3: The proposed system abstraction model for a power covert channel.

a result of the transformation h. In our model, h depends on the platform configuration (i. e., the power management) as well as power characteristics (hardware specific parameters), and transforms the current system utilisation to the corresponding power value. The transformation h cannot be determined easily, and in section 4.6, we show that h can be time-invariant or time-variant, depending on the platform configuration. The sink application snk observes the power changes and generates the corresponding output symbol trace y with symbols $y[k]$. We assume that the power covert channel has the following three characteristics:

1. It is time-discrete; we represent every channel use as a single sample k, as the Model Specific Registers (MSRs) used for the power measurements are just updated with a period of T_{msr}.

2. It is value-discrete; we show in section 4.6 that there is only a limited set of output symbols.

3. It is noise-free; any measurements artefacts in the power trace are not visible in the output symbol trace y, due to the conversion from power values to symbols by the sink application.

Output stage. A decoder converts the output symbol trace y to a bitstream b_o. In an error-free information transfer, the output bitstream b_o equals the input bitstream b_i.

Our model enables us to understand the power covert channel better and to determine a capacity bound (section 4.6). We also refer to this model when we design our test environment and the experiments to evaluate the channel (section 4.7).

4.4 Threat model and target setup

Our threat model is based on the scenario outlined in Figure 4.2. Furthermore, we assume that the sink application records a power trace, which it forwards via the Ethernet interface to a third-party server.

As presented in section 4.2, the current system utilisation is a main factor in the control loop that is the power management system. We assume that the device is idle during the time of the attack; for example, a Laptop in an office during a weekend or during the night. Hence, the utilisation of the device is mainly controlled by the source application. Furthermore, the source and the sink application cannot control any of the system's parameters, i. e., the `cpufreq` or `cpuidle` governors, as this would require elevated permission levels. Therefore, the source and sink applications are tailored to the attacked platform, which requires detailed knowledge of the system. We present an assessment of the implications of changing platform parameters on such a naïve implementation and propose improvements to the applications to make the power covert channel more robust, in section 4.7.

We demonstrate the power covert channel on the example of Intel-based platforms. There are two reasons for our choice: (i) they allow power measurements through energy estimations

provided by the system via MSRs, and (ii) x86_64 is still the still most used architecture in server and laptop systems, where Intel holds a market share of around 70% in Q1 2020[2]. In this chapter, we consider the following platforms:

1. A Lenovo ThinkPad T440p laptop based on a 4$^{\text{th}}$ generation Intel Core i7-4710MQ quad-core processor. The CPU supports two hyper-threads per core and allows operating frequencies between 800 MHz and 2.4 GHz as well as turbo boost of up to 3.5 GHz.

2. A server rack based on a 3$^{\text{rd}}$ generation Intel Xeon E5-2690 octa-core processor. This processor also features two hyper-threads per core and allows operating frequencies from 1.2 GHz up to 2.9 GHz and a turbo boost of up to 3.8 GHz.

For the rest of this chapter, we will refer to platform 1 as Haswell-i7, and platform 2 as Sandy-Xeon.

In favour of reproducibility, we define a controlled environment that is used throughout all experiments, unless otherwise stated. Both platforms are situated in a server room with an average ambient temperature of $\approx$ 23 C° and run Ubuntu, version 18.04.3 on Haswell-i7 and version 16.04.5 on Sandy-Xeon. Furthermore, both platforms use the `cpuidle` menu governor and the `cpufreq` userspace governor, such that the operating frequency is locked to the maximum. The source and the sink application are pinned to specific cores during the experiments using `pthread_setaffinity_np()`. Moreover, both applications are run with the highest priority with `SCHED_FIFO` scheduling class to minimise the scheduling artefacts using `pthread_setschedparam()`. Initial experiments showed that any further differentiation between C-states that are deeper than

[2] `https://www.cpubenchmark.net/mobile/market_share.html`

C1E-HSW on Haswell-i7, respectively C1E-SNE for Sandy-Xeon is not possible. Therefore, we set the maximum wakeup latency to 10 μs; limiting the maximum C-state to C1E-HSW for Haswell-i7 and C1E-SNE for Sandy-Xeon[3].

4.5 Channel implementation

Similar to the previous chapters, the applications used in this evaluation are also based on Experiment Orchestration Toolkit (ExOT). In this section, we give a brief overview of the two applications used for implementing the power covert channel, the source and the sink application. For more detailed information regarding ExOT and how applications are build, please refer to chapter 2.

Source application. The source application requires a core list and an execution trace as input. It creates as many threads as cores specified in the core list, pins them to the defined cores and replays the execution trace by activating as many threads as specified and idling the rest of the threads using `usleep()`. Active threads will execute a tight loop similar to the cpuburn benchmark[4] to ensure that the cores are not sent to a C-state by the `cpuidle` subsystem, while cores with idle threads will enter the C-state. An example of a tight loop is illustrated in Listing 4.1. All timing checks are done using the Time Stamp Counter (TSC), which proves sufficiently lightweight and accurate for our task. In addition, the application checks the overall timing and adapts the execution trace to avoid timing drifts due to jitters in the execution of the tight loop and `usleep()`.

[3]The wakeup latencies can be read via the sysfs interface at `/sys/devices/system/cpu/cpu$i/cpuidle/state/$n/latency`.

[4]`patrickmn.com/projects/cpuburn`

```
double a = 2.0e0;
double b = 2.0e0;

for (int count = 0; count < 1000; count++) {
  a *= b;
  b -= a;
}
```

Listing 4.1: C++ example of a tight loop with floating point operations and 1000 iterations.

Sink application. The sink application samples the MSR for the PP0 power plane (MSR_PP0_ENERGY_STATUS) with a sampling rate T and immediately converts the samples to power values. These power values are then kept in an in-memory log, which is dumped to a file as soon as the execution is stopped. Similar to the source application, the sink application uses TSC to monitor the overall timing to adjust T to avoid a long term timing skew.

4.6 Channel capacity bound

As outlined in chapter 2, we need to determine the channel capacity bound of the covert channel as a comparable metric. Following the definitions made by MacKay [Mac03, Chapter 17], we determine an upper bound C for the capacity per channel use of a noise-free channel, as shown in Equation (4.1). Here, N denotes the number of sent symbols, i. e., the number

$$C = \lim_{N \to \infty} \frac{1}{N} \log M_N [bit] \tag{4.1}$$

of channel uses, and M_N the number of distinct and feasible

symbol series of length N.

We construct a state diagram, where the states S represent the states of the channel. Every valid path in the state diagram corresponds to a sequence of transitions. Consequently, every state transition in the diagram represents a symbol that is forwarded to the channel. Starting from the initial state of the channel, M_N is equal to the number of distinct paths of length N in the state diagram.

To determine M_N, the number of possible distinct paths of length N, we derive the $S \times S$ connection matrix $\mathbf{A}$ based on the state diagram. An example of such a derivation is illustrated in Figure 4.7. An element $A_{s,s'}$ of the connection matrix $\mathbf{A}$ is 1 if there is a transition from state s to s' and 0 otherwise. To this end, we count the transitions to a state by means of Equation (4.2) and Equation (4.3). $\mathbf{c}^{(0)}$ is the initial state vector consisting of one 1, representing the initial state, and 0 otherwise. $\mathbf{c}^{(n)}$ holds the number of paths that

$$\mathbf{c}^{(n+1)} = \mathbf{A}\mathbf{c}^{(n)} \tag{4.2}$$

$$\mathbf{c}^{(N)} = \mathbf{A}^N\mathbf{c}^{(0)} \tag{4.3}$$

$$\lim_{N\to\infty} \mathbf{c}^{(N)} = \text{constant} \cdot \lambda_1^N \cdot \mathbf{e}_1 \tag{4.4}$$

$$M_N = \sum_s c_s^{(N)} \tag{4.5}$$

lead to a certain state after n uses of the channel. In the limit, the principal eigenvalue of $\mathbf{A}$, i. e., the eigenvalue with the largest absolute value, starts dominating the iteration in Equation (4.3). As a result, we obtain Equation (4.4), which shows that the dominating term of $\mathbf{c}^{(N)}$ is λ_1^N. Here, λ_1 is the principal eigenvalue of $\mathbf{A}$ and $\mathbf{e}_1$ is the corresponding eigenvector.

The number of possible paths is calculated, as shown

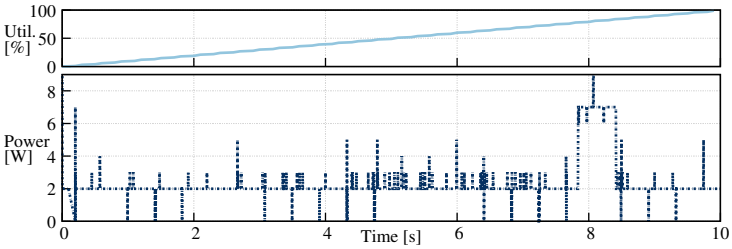


Figure 4.4: The power dissipation does not correlate to increasing utilisation from 0% to 100% on a single core.

in Equation (4.5), where $c_s^{(N)}$ is element s of the vector $\mathbf{c}^{(N)}$. We now use Equation (4.4) and Equation (4.5) in Equation (4.1) to obtain the channel capacity bound as shown in Equation (4.6).

$$C = \log_2 \lambda_1 \tag{4.6}$$

In the remainder of this section, we outline how to determine the state diagram model for the power covert channel and present the capacity calculations for the two platforms Haswell-i7 and Sandy-Xeon.

4.6.1 Determining the state diagram

Based on the threat and channel model presented in section 4.3, the power covert channel only allows a finite set of discrete symbols. These symbols represent the so-called *power covert channel states*, which we define in this subsection based on simple experiments.

First, we need an initial experiment to determine if the utilisation of a single active core influences the power measurements. Therefore, we ramp up the utilisation of one

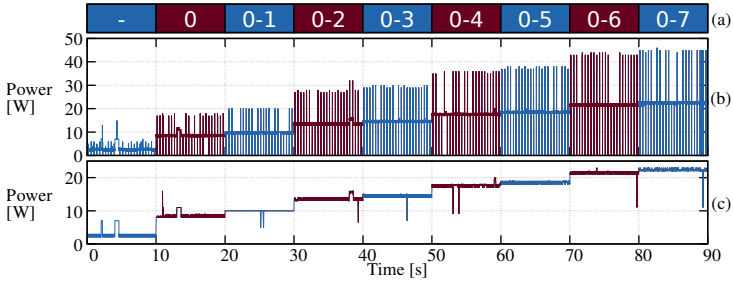


Figure 4.5: When setting a fixed operating frequency, we are able to identify how many physical cores are active. Due to the high amount of measurement artefacts, we apply median filtering to the raw power trace (b) to obtain the power trace (c). We observe a stepwise increase of the power dissipation depending on the number of fully utilised cores (a).

logical core from 0% to 100%, to check whether the resolution of the power trace is fine enough to detect different utilisation levels. The results for Haswell-i7 are depicted in Figure 4.4 and show some fluctuations for different utilisations. However, it is not possible to draw a direct connection from the utilisation trace to the power trace, as long as the core is not in any C-state. We conclude that utilisation changes may cause measurement artefacts in the power trace, but do not have an influence on the average measured power.

Next, we conduct an experiment to determine the power dissipation depending on the number of active cores for a *fixed frequency* case by using the maximum operating frequency. This allows us to determine the channel behaviour in a static scenario, a time-invariant transformation from utilisation to power h (see section 4.3). In Figure 4.5, the top plot (a) shows the set of fully-utilised logical cores (utilisation is 100%) in a 10 s time interval, (b) the power trace with a sampling rate of 1 ms, and (c) the power trace median filtered with a window

size of 8 samples.

The experiment reveals that there is a high amount of measurement artefacts, which may cause errors at higher symbol rates and can be explained by following MSR characteristics:

1. Reading is destructive, meaning that the value is set to 0 after reading.

2. According to the Intel® 64 and IA-32 Architectures Software Developer's Manual [Int16] the MSR is updated "*approximately*" every 1 ms.

Due to these two characteristics, it could happen that the sink application reads the MSR twice between two updates, causing a 0 in the power trace. Moreover, two logical cores are mapped to one physical core. For example, logical cores 0 and 1 are mapped to physical core 0.

Based on the observations of the experiments outlined in Figure 4.4 and 4.5, we define the states of the power covert channels as follows: A power covert channel state is defined by an observable and distinguishable power dissipation level of the system. Therefore, we identify 5 power covert channel states in our power trace for the fixed frequency case:

1. from time 0 to 10 s when no physical cores are utilised,

2. from 10 to 30 s when one physical core is utilised,

3. from 30 to 50 s when two physical cores are utilised,

4. from 50 to 70 s when three physical cores are utilised, and

5. from 70 to 90 s where all four physical cores are utilised.

This equals $N_C + 1 = 5$ states for Haswell-i7, where N_C is the number of physical cores.

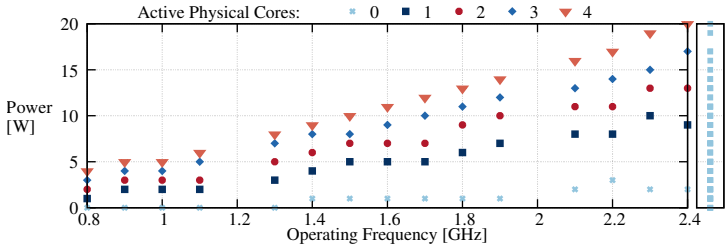
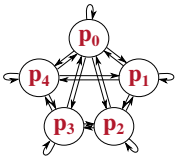


Figure 4.6: By varying the operating frequency, more power covert channel states can be exposed. Due to the lack of knowledge on the operating frequency, the sink application can only distinguish 20 power covert channel states, as illustrated in the power plane to the right.

The usage of a different `cpufreq` governor makes the channel more complex, as the power dissipation, and, respectively, the power covert channel states, also depend on the used processor frequency. Considering our channel model from section 4.3, this means that the transformation from utilisation to power h is time-variant. Therefore, we repeated the experiment illustrated in Figure 4.5 for every operating frequency of Haswell-i7. This experiment exploits all possible utilisation to power transformations h. Figure 4.6 illustrates the identified power covert channel states for different numbers of active physical cores and different operating frequencies. One point in the plot equals the integer-rounded mean of the power trace values within one power covert channel state, i. e., the integer-rounded power mean of interval 0 to 10 s from Figure 4.5 is represented as a point for 0 active physical cores at frequency 2.4 GHz in Figure 4.6.

The right scale presents the projection of all power covert channel states onto the power plane. This projection is necessary, as the sink application cannot determine the operating frequency of the cores but only the power trace. The



$$\mathbf{A} = \begin{Bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{Bmatrix} \tag{4.7}$$

Figure 4.7: State diagram (left) and connection matrix (right) for the fixed frequency case for Haswell-i7 with the power covert channel states p_0 to p_4.

power plane shows that there are 20 distinguishable power covert channel states in a *variable frequency* case, where the operating frequency changes can be fully controlled. We do not include the operating frequencies reachable through the Intel turbo boost in our analysis because these frequencies can only be reached under certain conditions and cannot be forced through the userspace governor.

As we now know the number of states $|S|$, we can determine the transition matrices $\mathbf{A}$. Under the conditions that every core can be switched on and off, and there are no restrictions to setting the operating frequency, for our two cases fixed and variable frequency, all elements of $\mathbf{A}$ are one. The state diagram and the corresponding transition matrix for the fixed frequency case for Haswell-i7 are outlined in Figure 4.7.

We also performed the same experiments and evaluation on our second platform Sandy-Xeon. Based on this evaluation, we identify $N_C + 1 = 9$ states for the fixed frequency case and 13 states for the variable frequency case.

Accessibility of power covert channel states. Elements of a transition matrix $\mathbf{A}$ of a power covert channel setup might be 0 if, for example, a governor would not allow all frequency transitions or some cores were prohibited from entering sleep

states. Therefore, it is essential to have a detailed knowledge of the setup to be able to determine all possible state transitions in the channel model.

4.6.2 Capacity calculation

Now, we derive the principal eigenvalue λ_1 of $\mathbf{A}$ and use Equation (4.6) to determine the channel capacity bound per channel use. For Haswell-i7, we derive upper channel capacity bounds of 2.32 and 4.32 bits per channel use; for Sandy-Xeon 3.17 and 3.70 bits per channel use, for the fixed and variable frequency case.

Knowing the capacity bound and the update period of the MSR $T_{\text{msr}} = 1$ ms, we calculate the theoretical bandwidth of the channel, as shown in Equation (4.8). This yields a maximum

$$B_{\text{max}} = \frac{C}{T_{\text{msr}}} \tag{4.8}$$

bandwidth of $B_{\text{max}} = 2322$ bits per second (bps) for Haswell-i7 and 3170 bps for Sandy-Xeon, considering a fixed operating frequency (time-invariant utilisation to power transformation h).

The variable frequency case yields higher maximum bandwidths of 4322 bps and 3700 bps for Haswell-i7 and Sandy-Xeon, respectively. However, implementing the variable frequency case version of the power covert channel would require the source application to have elevated privilege levels to be able to set the operating frequency governor to userspace and control the current operating frequency. This contradicts our threat model, outlined in section 4.4, which defines that neither the source nor the sink application has elevated privilege levels. Therefore, for the remainder of the

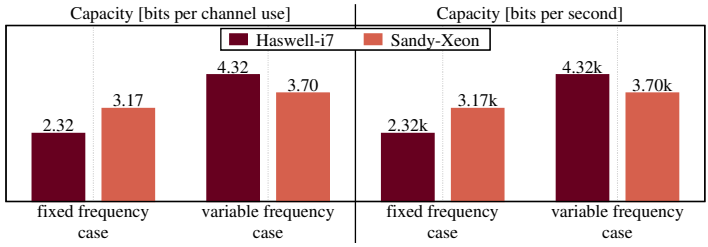


Figure 4.8: Determined capacity bound for the two analysed platforms. The capacity bounds indicate that the power covert channel might pose a high risk when the operating conditions are favourable for the attacker, i. e., the platforms are idle, and there is little interference.

chapter, we will only focus on the fixed frequency case, as an implementation does not require elevated privilege levels of the source application.

Capacity based threat potential assessment. According to the US department of defence 1985 *Orange Book* [US 85], "a covert channel bandwidth that exceeds a rate of one hundred (100) bits per second is considered high". Based on this definition, we state that, in terms of the capacity outlined in Figure 4.8, the power covert channel is a high-risk data leak.

4.7 Experimental analysis

In this section, we deploy our channel setup (see section 4.5) on the two platforms Haswell-i7 and Sandy-Xeon (see section 4.4) to provide experimental evidence that supports the theoretical analysis, consisting of the model defined in section 4.3 and the capacity bound we derived in section 4.6.

The sink application uses a sampling rate T of 1 ms, as oversampling of the MSR does not improve signal quality

Power Covert Channel State	Codeword		
	Binary	Huffman 5	Huffman 9
p_0	0	00	010
p_1		01	011
p_2		100	000
p_3		101	001
p_4	1	11	100
p_5			101
p_6			110
p_7			1110
p_8			1111

Table 4.1: Power covert channel state, or symbol, to bit-codeword mapping for the different encoding used in the evaluation.

without implementing a sophisticated zero replacement and filtering scheme. ExOT ensures that the applications are synchronised at the start of the transmission. Furthermore, we evaluate different source codes for transmission, as defined in Table 4.1. Each symbol of a code is equal to a power covert channel state:

- the Binary code defines a zero as no active core and a one as 4 cores active,
- a Huffman code with 5 states called Huffman 5, and
- a Huffman code with 9 states called Huffman 9 for Sandy-Xeon only.

Using Huffman 5 and Huffman 9, we exploit all available power covert channel states (see section 4.6) for the two platforms Haswell-i7 and Sandy-Xeon, respectively.

The decoding is done according to the block diagram in Figure 4.9. We first filtered the signal with a moving

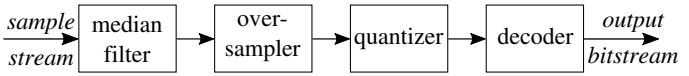


Figure 4.9: The signal decoding is performed in multiple stages.

median filter to remove measurement artefacts. Equation (4.9) ensures that the window length of the median filter decreases in proportion to the measurement samples per symbol. The window length of the median filter w_i at bitrate i is calculated according to Equation (4.9), where N_i is the number of measurement samples per symbol. The minimum and

$$w_i = \min\left\{\max\left\{2 \cdot \left\lfloor \frac{N_i}{3} + 0.5 \right\rfloor + 1, 1\right\}, 9\right\} \tag{4.9}$$

maximum filter length, 1 and 9, as well as the proportionality factor of 3 were evaluated experimentally towards a minimal bit error rate.

After filtering, the signal is oversampled so that there are at least 100 samples per symbols, whereas nearest-neighbour interpolation is used. The filtered signal is then quantised according to the platform-specific power levels using a majority voter. Last, the median power value for the symbol duration is calculated and provided to a random forest classifier with 100 estimators, and standard parameters otherwise, for symbol decoding[5].

As not all symbols in a Huffman code translate into the same number of bits, this may lead to different lengths of the received bitstream and the ground truth bitstream. In addition, a symbol error at the beginning of the message might result in

[5] `sklearn.ensemble.RandomForestClassifier`

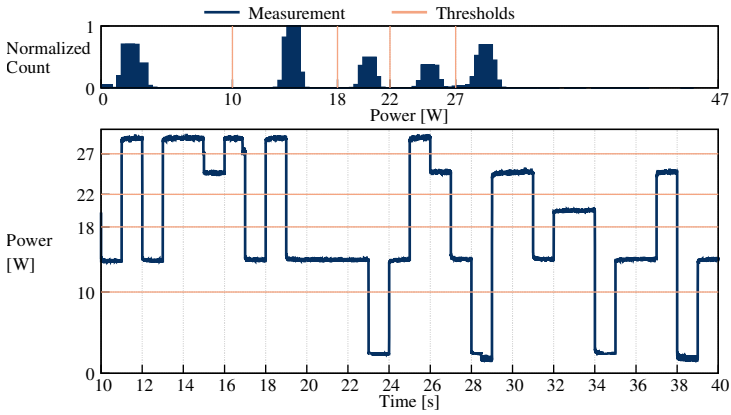


Figure 4.10: Data to determine the quantisation thresholds for Haswell-i7. The histogram and the time trace show that the power stages are well separated.

a shift by one bit in the remainder of the transmission, which would lead to a high bit error count. Therefore, instead of a bit-wise comparison of the received bitstream and the ground truth to calculate the number bit errors, we employ the Levenshtein distance as an error metric. The Levenshtein distance describes how many modifications have to be applied to a sequence so that it is identical to a reference sequence, whereas valid modifications are “insert”, “delete” and “replace”. We calculate the error rate as Levenshtein distance divided by the length of the ground truth.

4.7.1 Determining the quantisation thresholds

We empirically determine the quantisation thresholds by analysing the power trace of a 5000 bit transmission. These thresholds are used to map the power measurements to the corresponding power covert channel states. The quantisation levels could also be determined using supervised or unsu-

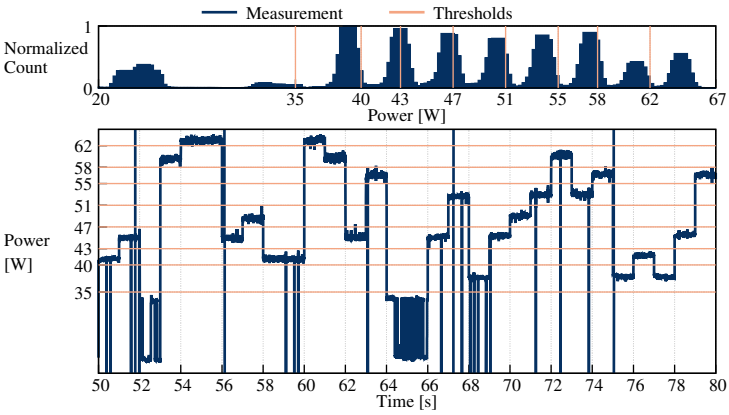


Figure 4.11: Data to determine the quantisation thresholds for Sandy-Xeon. There is more interference in the data, and the power levels are not clearly separated.

pervised machine learning methods, like random trees or clustering algorithms for 1D sequence data, but we leave an evaluation of such methods to future work. Figure 4.10 and 4.11 illustrate the maximum-normalised histogram and a snippet from the time trace for each platform.

To determine the quantisation thresholds for Haswell-i7, we simply separate the peaks in the histogram by trying to centre peaks in between the thresholds. The time trace in Figure 4.10 also indicates that this naïve approach is sufficient to provide a fitting signal quantisation.

Figure 4.11 indicates that determining the thresholds for Sandy-Xeon is more challenging. This is due to the increased amount of measurement artefacts, as well as the fact that the different power levels are less clearly distinct for Sandy-Xeon than for Haswell-i7. Therefore, empirical evaluation indicated that, rather than separating the peaks in the histogram, thresholds found using visual inspection of the trace data

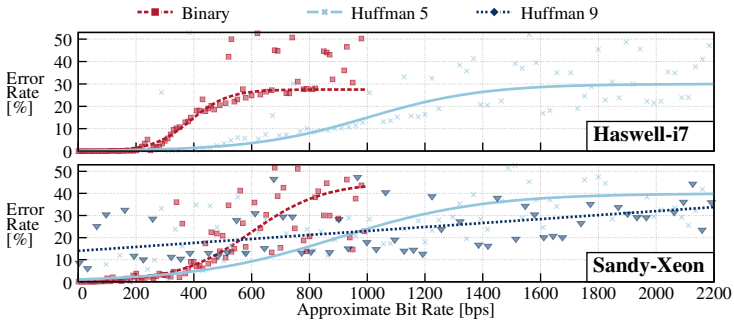


Figure 4.12: Average error rate for three runs for each bit rate and the corresponding trend line for the different coding schemes for Haswell-i7 (upper plot) and Sandy-Xeon (lower plot). The increased amount of outliers on Sandy-Xeon indicate that the power covert channel is more error-prone on this platform.

show better performance. For example, consider the trace in Figure 4.11. We identify the power levels in the power measurement trace, for instance, the highest power level from 54 s to 56 s. Next, we choose the quantisation thresholds to separate the power levels visible in the time trace data.

4.7.2 Controlled environment performance

To evaluate the performance of the power covert channel, we perform a symbol rate sweep from 1 to 1000 symbols per second, which is the maximum possible symbol rate due to the maximum sampling rate of 1ms (see section 4.6). Furthermore, we use a symbol rate spacing of 10 symbols per second. For each symbol rate, we perform three runs where we transmit a 1500 bit random message to train the random forest classifier and a 5000 bit random message to perform the performance evaluation, for each run. Figure 4.12 shows the

average error rate for three evaluation runs for both platforms Haswell-i7 and Sandy-Xeon and the corresponding trend lines. The symbol rate is converted to the approximate bit rate using the compression factor c, as defined in Equation (4.10) and Equation (4.11). For example, the Binary code has two

$$c = \frac{\sum_i^N (l_i)}{N} \tag{4.10}$$

$$\tilde{f_b} = f_s \cdot c \tag{4.11}$$

l_i ... length in bits of the i^{th} codeword in a code

N ... Number of codewords in a code

c ... Compression factor

$\tilde{f_b}$ ... Approximate bit rate

f_r ... Symbol rate

codewords of length 1, which results in a compression factor c of 1. The compression factor c for the Huffman encodings are 2.40 for Huffman 5 and 3.22 for Huffman 9.

The analysis results presented in Figure 4.12 show that the performance of the power covert channel is highly system dependent. While the Binary encoding seems to work better on Sandy-Xeon, allowing almost 450 bps with less than 10% error rate compared to only 350 bps on Haswell-i7, the contrary is true for Huffman 5 encoding. Here, on Sandy-Xeon, according to the trend line, we can only achieve bitrates of less than 600 bps for error rates lower than 10%, while on Haswell-i7 we achieve up to 900 bps. Transmissions with the Huffman 9 encoding result in error rates of at least 15%.

These observations and the fact that there are more outliers on Sandy-Xeon indicate that the power covert channel is more error-prone on Sandy-Xeon than onHaswell-i7 when using more than two power covert channel states. Based on a detailed

analysis of the experimental data, we trace back a majority of these errors on Sandy-Xeon to architectural properties. Despite the fact that we limit the wakeup latency for cores to return from C-states, the power trace does not follow our input utilisation trace fast enough. This might be caused by higher latencies when waking up from and sending cores to C-states. Moreover, the power traces collected from Sandy-Xeon show that there is a slow, long-term increase in the power measurements. We assume that this is caused by the rising temperature of the core, which also causes higher power dissipation. Therefore, the differentiation of the power covert channel states, i. e., the power measurement quantisation, is not working reliably on Sandy-Xeon. While the latencies cannot be compensated in the symbol decoding, de-trending methods can be used to nullify the long term power measurement increase.

Another problem of the power covert channel, which applies to both platforms Haswell-i7 and Sandy-Xeon, is the synchronisation between source and sink application. Inaccurate synchronisation leads to higher error probabilities at higher bitrates, as there are fewer samples per symbol. This could be handled with a more sophisticated signal processing and synchronisation scheme.

Finally, experiments have shown that the power measurements representing the power covert channel states may vary slightly for repeated executions of the same utilisation trace. To compensate varying power measurements, an improved implementation of the power covert channel needs to be capable of adapting the power threshold for power covert channel state detection according to the current trace. For example, adaptive power covert channel state detection can be implemented using a calibration header in the data transmission and threshold detection during decoding.

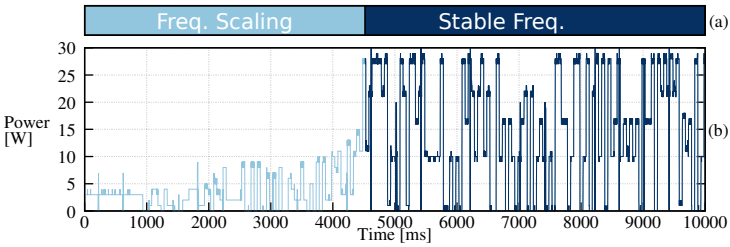


Figure 4.13: If the operating frequency is not stable, the data transmission is disturbed, as the frequency scaling leads to a time-variant utilisation to power transformation h.

4.7.3 Robustness

To briefly evaluate the robustness of the power covert channel, we analyse three scenarios on Haswell-i7:

1. the *variable frequency* case where the conservative governor is used to set the operating frequency of the cores,
2. the *light interference* case where a second source application is used to keep all but one physical core active permanently, and
3. the *heavy interference* case where we employ the ffmpeg application to utilise the device.

4.7.3.1 Variable frequency

The use of frequency governors adds more disturbance to the channel, as the transformation from utilisation to power h (see section 4.3) is time-variant. Figure 4.13 illustrates the start of a data transmission in a system using the conservative governor and Huffman 5 encoding at 100 symbols per second.

As the system is idle before starting the transmission, the conservative governor will scale to the lowest possible operating frequency to minimise the power dissipation of the device. Therefore, after starting the transmission, the power measurements increase as the system is scaling up the frequency due to the high utilisation generated by the source application. In the example illustrated in Figure 4.13, after approximately 4500 ms, the system reaches the maximum operating frequency. Due to the core utilisation during the transmission, the governor keeps the operating frequency at the maximum. Therefore, the system is now in a stable state, during which h is time-invariant, and the power measurements are properly mapped to the symbols, i. e., the power covert channel states, and decoded.

To be able to compensate for changing operating frequencies, i. e., a time-variant channel transfer function h, advanced signal coding schemes need to be applied. For example, for the conservative governor used in this evaluation, the implementation could use the following coding scheme to ensure that the system is at the highest operating frequency for the whole transmission. Every message uses a preamble that fully utilises the cores and forces the system to scale to the highest frequency. In addition, the messages would need to be encoded in a way that ensures that at least one core has a utilisation high enough to prevent the conservative governor from scaling down the frequency. Which encoding fulfils this utilisation requirement depends on the coding scheme, the transmission rate and the setting of the governor.

4.7.3.2 Light interference

In this scenario, a second source application is occupying all but one physical core. This prohibits the occupied cores from entering C-states for power saving and, therefore, some power

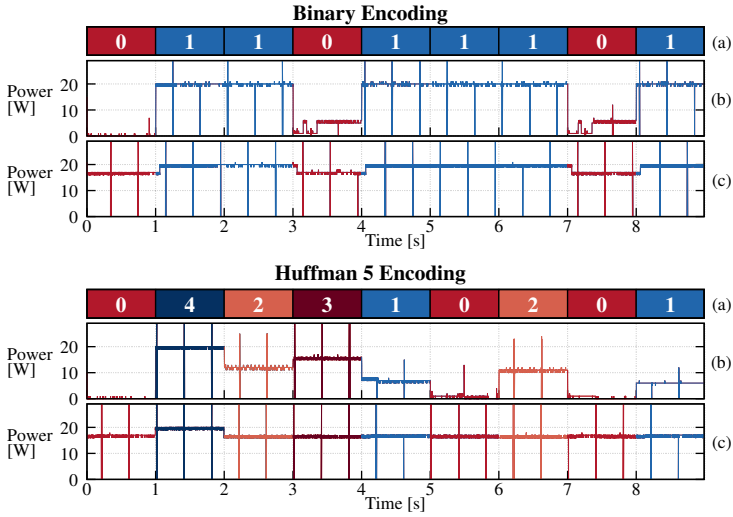


Figure 4.14: Binary encoding is more robust against interference than Huffman 5 encoding, as the symbols (a) can be distinguished without (b) and with interference (c).

covert channel states cannot be reached. Figure 4.14 illustrates power traces for the controlled environment and the inference scenario for both encoding schemes Binary and Huffman 5. The figures show that only a differentiation between two states is possible; therefore, the transmission with Binary encoding is still feasible while the Huffman 5 transmission is disturbed.

This experiment shows that a transmission is viable whenever the number of states needed to transmit N_T is less or equal than $(N_C - N_I + 1)$. Here, N_C is the number of physical cores available on the system and N_I is the number of physical cores occupied by interfering applications. This means that a coding scheme that requires less power covert channel state is more robust against such partial utilisation than a scheme that requires many power covert channels states. Moreover, an

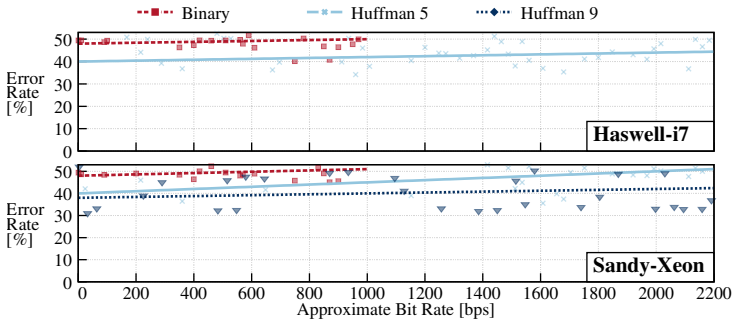


Figure 4.15: The power covert channel on both platforms, Haswell-i7 (upper plot) and Sandy-Xeon (lower plot), does not allow data transmission if the platforms are heavily utilised. We used ffmpeg video conversion to utilise the platforms.

attacker may use the power measurements to detect when the utilisation of the platform is low enough that sufficient power covert channel states are accessible to start a transmission.

4.7.3.3 Heavy interference

In a second test, we employ the ffmpeg application as a source of interference. Similar to chapter 2 and 3, we configure ffmpeg to run an infinite loop to convert a 9.56 minutes long animated mp4 video[6] and pipe the output to `/dev/null`. The results illustrated in Figure 4.15 show that a transmission using the power covert channel is no longer possible, as ffmpeg utilises as many cores as possible.

Our experiments show that, compared similar cover channels, like the thermal covert channel we presented in chapter 3, the power covert channel channel allows a higher throughput but is less robust to disturbances by other applications on the platform.

[6] `peach.blender.org`

4.8 Summary

In this chapter, we presented the power covert channel: a covert channel based on processor power measurements. For our evaluation, we chose broadly used Intel-based platforms, which provide power measurements via MSRs.

We presented a detailed theoretical analysis illustrating how to model the power covert channel and derived a capacity bound. Our methodology to derive the channel capacity bound is generally applicable to other covert channels with similar characteristics as the power covert channel. We considered a fixed operation frequency of the platform during the transmission, which resulted in a capacity bound of 2322 bps for a platform with 4 physical cores and 3170 bps with 8 physical cores.

Moreover, we conducted a thorough experimental analysis to exploit achievable throughputs under controlled conditions and evaluated the robustness of the power covert channel against external influences. Our experiments showed that, under controlled conditions, we could achieve throughputs of up to 900 bps with an error probability of less than 10% using a very simplistic channel implementation.

The power covert channel has a high channel capacity bound and allows high throughputs, which shows the channel's potential to leak information. However, the channel is more prone to disturbance than comparable covert channels, and a successful attack requires detailed knowledge of the attacked platform utilisation pattern, architecture and power management system. Therefore, we classify the power covert channel as a threat that only needs to be considered when designing highly secure systems.

5

Machine learning for covert channel symbol decoding

In the previous chapters, we outlined how to analyse data leaks exhaustively in current computing systems. We also presented two data leaks based on CPU sensor readings. However, such data leaks can be mitigated by restricting the access to CPU sensor readings.

In this chapter, we present an exhaustive analysis of a data leak which cannot be mitigated easily: the frequency covert channel. The frequency covert channel is a discrete covert channel based on indirect measurements of the operating frequency of the CPU. To analyse this discrete covert channel, we will apply the methods presented in chapters 2 and 4. Furthermore, we will show that a frequency covert channel cannot be established using naïve signal processing strategies. To remedy this, we present a method to build high-performance

This chapter is based on work presented in [Mie+18].

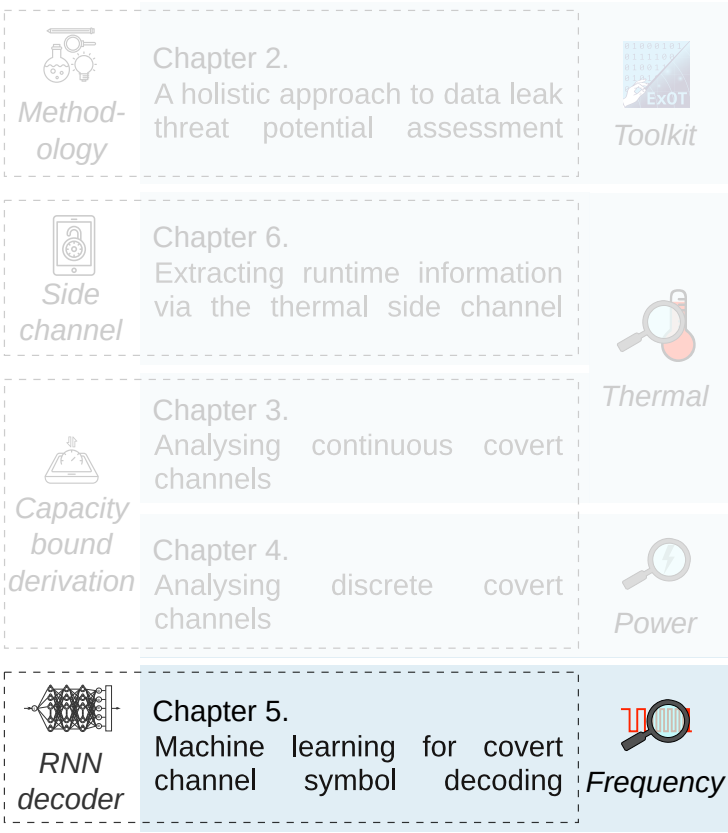


Figure 5.1: In chapter 5 we present a communication model and evaluation of the threat potential of the frequency covert channel on two distinct platforms. In addition, we propose a robust signal decoder based on a Recurrent Neural Network (RNN).

signal decoders based on machine learning techniques.

5.1 Introduction

Current mobile computing systems are expected to optimise their power consumption for various reasons, e. g., for prolonging the battery life or for thermal protection. One of the most commonly used techniques for power optimisation is Dynamic Voltage and Frequency Scaling (DVFS). DVFS allows the optimisation of the energy consumption by changing the operating frequency and the supply voltage according to performance needs. Typically, a software component in the operating system, denoted as governor, is responsible for selecting the frequency based on a specific predefined policy, the associated parameters and run-time information.

Another important design challenge for current computing systems is to provide security and confidentiality of sensitive information, despite concurrent application execution. To ensure the security and confidentiality of data, system designers often apply the security paradigm of permission separation and application isolation. This paradigm dictates that applications are executed in isolation and have specific sets of permissions regarding shared resources and memory. The corresponding mechanisms are typically implemented by the Operating System (OS), possibly supported by hardware. Examples for such mechanisms are sandboxing, virtualisation and fine-grained permission control like in Android, or moving applications into hardware enclaves, like when using Intel SGX[1] or the Arm TrustZone[2]. The secure operation of systems using these mechanisms is only guaranteed if no data transfer between the isolated elements is possible. Therefore, data leaks like covert channels are one of the biggest threats for systems relying on permission separation and application isolation.

In this chapter, we investigate how to build such a data

[1] `https://software.intel.com/en-us/sgx`

[2] `https://developer.arm.com/ip-products/security-ip/trustzone`

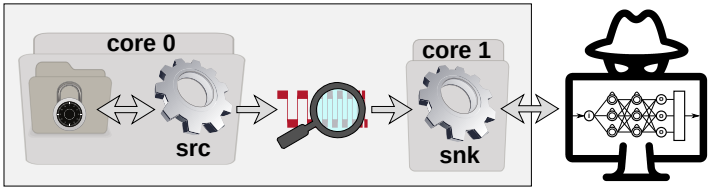


Figure 5.2: The two applications source (src) and sink (snk) are placed on different cores and are isolated from each other. While src has access to restricted data, snk can use the systems communication interfaces. If the two applications establish a covert communication channel to transfer data, the restricted data is leaked to an attacker. This covert channel compromises the security paradigm of permission separation and application isolation.

leak based on frequency scaling of mobile multicore systems. Figure 5.2 illustrates the example scenario we use to analyse the data leak. As in the previous chapters, we consider two colluding applications trying to establish a covert channel on a multicore system. The *source application* (src) running on core 0 has access to restricted data, i. e., a cryptographic key, but no access to the communication interfaces. The *sink application* (snk) runs on core 1 and has full access to a communication interface, but no access to the restricted data. To comply with the security paradigm of permission separation and application isolation, source and sink are isolated from each other and are not allowed to communicate. In addition, we consider a common setting in current multicore architectures, namely a processor with multiple cores in the same voltage and frequency domain, i. e., both cores share the same voltage and frequency level at any point in time. Furthermore, we assume that the system is idle, hence only the OS and its vital processes are active at attack time. As the system is idle, the source application is able to influence the behaviour of DVFS

by generating utilisation patterns. If the sink application is able to detect the frequency changes induced by the source application, the two applications have established a covert channel, which can be used to extract restricted information. This covert channel is referred to as *frequency covert channel*, as the key shared resource is the frequency of the cores.

We will present an implementation of the frequency covert channel without the need for elevated privileges by the attacker. Furthermore, we will show that the frequency covert channel has a sufficient bandwidth and low error rate to break the security paradigm of permission separation and application isolation. Considering that, with emerging trends like edge computing, embedded systems are also often equipped with powerful processors using DVFS, this illustrates that the frequency covert channel poses a substantial threat for a broad range of devices.

In this chapter, we will show that baseline signal decoding methods fail for data sent via the frequency covert channel. This is due to two main reasons, (i) the governor behaviour is heavily platform-dependent, and (ii) the operating frequency of a system depends on the current utilisation and past operating frequencies. Therefore, a signal decoder which is capable of compensating for these characteristics of the data stream is needed. We show how to build such a signal decoder based on a Recurrent Neural Network (RNN) and apply it on data sent through the frequency covert channel.

Contributions. Our main contributions in this chapter are

1. We are the first to apply a formal communication model and derive an upper bound on the capacity of the frequency covert channel. Furthermore, we derive a robust communication scheme based on the formal communication model of the frequency covert channel.

2. In contrast to previous work, our implementation of the frequency covert channel does not require elevated privileges for system file access. It allows the source and the sink application to communicate if they run on different cores.

3. To the best of our knowledge, we are the first to present the usage of a Recurrent Neural Network (RNN) for signal decoding in a covert channel scenario.

4. We present extensive experimental evidence to evaluate the feasibility of the frequency covert channel under realistic conditions for two platforms representative for laptops and hand-held devices. Furthermore, we use our extensive experimental evaluation to provide hints for possible mitigation strategies.

5.2 Frequency scaling in Linux

The operating frequency of current CPUs can be controlled by different modules in the system, for example, Linux Kernel DVFS, CPU throttling through Dynamic Thermal Management (DTM), device manufacturer-specific power and performance optimisation or CPU hot-plug techniques. In this chapter, we focus on the effect of the Linux Kernel DVFS. As our threat scenario considers the devices to be idle, we assume CPU throttling caused by DTM will not occur. Moreover, other power optimisation techniques for commercial devices are modelled as a (systematic) interference on the channel. This restriction is valid as long as there is a change of the frequency depending on the utilisation pattern of the CPU.

Linux is used among a diverse range of devices, for example, server or desktop systems, laptops, Android on

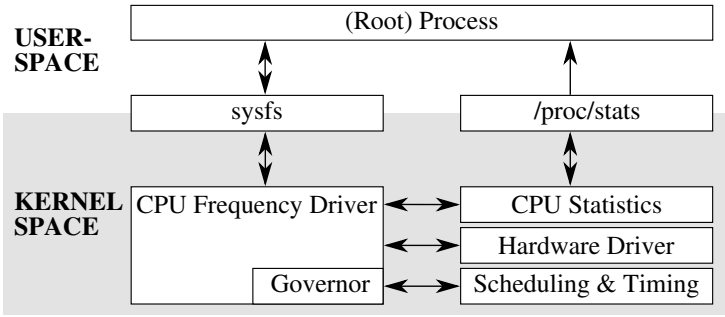


Figure 5.3: A simplified illustration of the CPU frequency control in the Linux Kernel, depicting which modules interact.

smartphones or tablets and various embedded systems like the Raspberry Pi. In addition, the open source nature of the Linux Kernel and its components allows us to review the code that handles frequency scaling. Examples of CPU frequency driver implementations in Linux are the *intel p_state* or the *acpi-cpufreq*[3] driver. In this chapter, we consider the acpi-cpufreq driver, which is used in Android systems as well as in Ubuntu and similar OSs. Figure 5.3 gives a simplified overview of the main components of the acpi-cpufreq driver, which we discuss in detail in this section.

5.2.1 The CPU frequency driver

The CPU frequency driver operates as an interface to all the other Kernel components, most importantly the `sysfs` nodes and the hardware driver. One responsibility of the CPU frequency driver is to maintain the `sysfs` nodes used to control frequency scaling from userspace. Furthermore, the CPU frequency driver instructs the hardware driver to set the frequency for each

[3] `https://uefi.org/specifications`

frequency domain, interacts with the scheduling unit and passes information, like the utilisation statistics, from the Kernel to the governor.

5.2.2 The governor

The governor should be called periodically with the timer period T_s, also called *sampling period*. When called, the governor determines the new frequency f_{set} for every frequency domain. Due to differences in hardware and user needs, many customised frequency governors are available for devices based on open source platforms like the Linux Kernel. The multitude of governor options allows the user to set the trade-off between performance and energy consumption. Different governors can vary in terms of static characteristics, like minimum and maximum frequency, but also in their frequency dynamics. In this chapter, we will focus on the *conservative governor*, which is one of the most commonly used governors in mobile, battery-powered systems.

The rules which the conservative governor applies to determine the next ideal target frequency f_{new} are summarised in Equation (5.1). Roughly speaking, the conservative governor

$$f_{\text{new}} = \begin{cases} f_{\max} & \forall \quad \frac{t_i}{t_m} < I_{\text{low}} \wedge f_{\max} < f_{\text{cur}} + \Delta f \\ f_{\text{cur}} + \Delta f & \forall \quad \frac{t_i}{t_m} < I_{\text{low}} \wedge f_{\max} \geq f_{\text{cur}} + \Delta f \\ f_{\text{cur}} & \forall \quad I_{\text{low}} \leq \frac{t_i}{T_s} \leq I_{\text{high}} \\ f_{\text{cur}} - \Delta f & \forall \quad \frac{t_i}{t_m} > I_{\text{high}} \wedge f_{\min} \leq f_{\text{cur}} - \Delta f \\ f_{\min} & \forall \quad \frac{t_i}{t_m} > I_{\text{high}} \wedge f_{\min} > f_{\text{cur}} - \Delta f \end{cases} \tag{5.1}$$

uses the average core utilisation in the past time interval to determine the new frequency f_{set}. If the utilisation is below or above a certain threshold, the governor reduces or increases

the frequency, respectively. Here, f_{cur} is the current frequency target and Δf is the frequency scaling step. To scale the frequency, the governor uses the relationship between idle time t_i and the total measurement time t_m, which is equal to the time elapsed between the last and the current call time of the governor. For simplicity, let us call the term t_i/t_m *idleness*. The governor adapts the frequency depending on the lower idleness threshold I_{low} and the higher threshold I_{high}. There are five scaling cases:

1. The idleness is lower than I_{low} and the sum of the frequency f_{new} and Δf is higher than f_{max}:
   f_{new} is set to f_{max}.
2. The idleness is lower than I_{low}:
   f_{new} is increased by Δf.
3. The idleness is between I_{low} and I_{high}:
   f_{new} is not changed.
4. The idleness is higher than I_{high}:
   f_{new} is decreased by Δf.
5. The idleness is higher than I_{high} and difference of the frequency f_{new} and Δf is lower than f_{min}:
   f_{new} is set to f_{min}.

After calculating f_{new}, the CPU frequency driver selects f_{set} from the discrete set of frequency levels that are available on the system. To select the frequencies, the CPU frequency driver applies one of two rules:

(A) If the frequency is scaled up, $f_{\text{new}} > f_{\text{cur}}$, f_{set} is set to the highest available frequency below or at the target.

(B) If the frequency is scaled down, $f_{\text{new}} < f_{\text{cur}}$, f_{set} is set to the lowest available frequency at or above the target.

After the frequency has been changed to f_{set}, f_{cur} is set to f_{new}.

5.2.3 Userspace interfaces

The `sysfs` and the `/proc/stats` pseudo file system nodes allow processes to access frequency scaling relevant information from userspace. On the one hand, detailed information on the cpu usage statistics can be read via the `/proc/stats` node. On the other, the `sysfs` nodes offer not only information about the governor, but also give processes with root permissions the opportunity to change the governor behaviour during runtime.

The `sysfs` holds a so-called *policy*[4] for every frequency domain. Among others, the policies contain the following parameter nodes:

- All cores affected by the policy in `affected_cpus`.
- The current frequency f_{cur} in `scaling_cur_freq`.
- The minimum frequency f_{min} in `scaling_min_freq`.
- The maximum frequency f_{max} in `scaling_max_freq`.
- `scaling_available_frequencies` reports all possible frequency steps.
- `scaling_available_governors` shows all available governors.
- Currently used driver and governor in `scaling_driver` and `scaling_governor`, respectively.

Furthermore, the `sysfs` contains nodes for the global governor and frequency driver settings, which apply to all frequency domains[5]. The global parameters contain, e. g., the sampling period T_s, the utilisation thresholds as $\left(1 - I_{\text{high}}\right) \cdot 100$ and $(1 - I_{\text{low}}) \cdot 100$, as well as the relative frequency step-size $\Delta f_{rel} \cdot 100$.

[4] `/sys/devices/system/cpu/cpufreq/policy$i/` in Ubuntu OSs, where i is the frequency domain number.

[5] Ubuntu: `/sys/devices/system/cpu/cpufreq/[governorname]/`

5.3 Threat Model

We consider the scenario presented in Figure 5.2, where we assume that the source and sink application are already deployed. This could either be done through code injection or by user installation. For example, an attacker could put two separate applications on the application store that seem unrelated but contain the necessary functionality to establish a joint covert channel. These applications are not suspicious, because each on its own does not violate any security restrictions and might be marked as coming from a different source.

The sink application `snk` measures and records the current frequency. The gathered information is then forwarded via a communication interface to an attacker device. Further, we assume that the attacked device is idle or only lightly utilised during the attack, e. g., a hand-held device like a smartphone during the night or a laptop powered on in an empty office during a weekend. Therefore, the source application `src` and the sink application `snk` wait until the average system utilisation is low and will presumably stay low for some time.

The target platforms for this chapter are mobile devices with shared frequency domains among multiple cores. This architectural feature is present in almost all state-of-the-art processor architectures that are used in computing devices such as mobile phones, tablet computers, laptops or servers. For instance, commonly used big.LITTLE architectures in mobile processors typically feature one frequency domain for all LITTLE and one frequency domain for all big cores, e. g., the Samsung Exynos 5422 Octa. Moreover, in systems with hyper-threading, where multiple logical cores reside on a single physical core, these logical cores share the same frequency domain.

The frequency of cores can be inspected with two methods:

```
1  double a = 2.0e0;
2  double b = 2.0e0;
3
4  for (int count = 0; count < 1000; count++) {
5    a *= b;
6    b -= a;
7  }
```

Listing 5.1: C++ example of a tight loop with floating point operations and 1000 iterations.

(a) reading system files, or (b) timing measurements. On Linux, method (a), can easily be blocked by requiring elevated privilege levels to read the system file[6] . The main restriction of method (b) is that it only works if the source and the sink application are placed on two cores within the same frequency domain.

Timing Measurements Method. To inspect the frequency via timing measurements with method (b), the sink application `snk` executes a tight loop with a fixed number of instructions and measures the computation time. An example of a tight loop is illustrated in Listing 5.1. This measurement is then used to determine the *empirical frequency* of the core. The sink application `snk` only needs access to a reliable timer in order to determine the time between starting and stopping the measurement load. For instance, the sink application may use the `gettimeofday()` system call a standard interface, which is a POSIX-conform and does not require elevated privileges. While the method for timing measurements (b) does not need any elevated privileges, it comes with two major challenges: (i) its accuracy suffers from interference from other tasks, and (ii) the measurement load increases the total utilisation of

[6] i. e., with the acpi-cpufreq drivers
`/sys/devices/system/cpu/cpu$i/cpufreq/scaling_cur_freq`

the cores which influences the frequency via the governor, i. e., the measurements indirectly influence the quantity to be determined. In section 5.8 we show how to cope with the challenges and implement the frequency covert channel without the need for elevated privileges.

5.4 Channel capacity bound

In this section, we present the derivation of the capacity bound of the frequency covert channel. As mentioned in chapter 2, capacity bounds are useful to assess the threat potential of data leaks, as they are independent from the data leak implementation artefacts. We determine the channel capacity bound using the method presented in section 4.6.

We consider the frequency covert channel to be value and time-discrete for the following two reasons: (i) it has a discrete number of different frequency levels, and (ii) only updates the operating frequency with a period of T_s. Therefore, to calculate the channel capacity bound, we need to determine the state diagram of the channel and the number of states of the channel. The number of states in the state diagram $|S|$ does not only depend on the possible frequency levels of the system, but also on the characteristics of the governor. As $|S|$ influences the complexity and capacity bound of the channel, let us derive a bound for $|S|$.

Based on the detailed description of the conservative governor in section 5.2, we characterise a state by the pair $(f_{\text{new}}, f_{\text{set}})$. Here, f_{new} denotes the target frequency and f_{set} the actual frequency, which is selected by the CPU frequency driver. The value of f_{new} only changes with a step size of Δf. Furthermore, f_{new} is clipped so that it cannot exceed f_{min} and f_{max}. f_{min} and f_{max} are the minimum and maximum

operating frequency the hardware platform supports. Hence, we have at most $2\lceil(f_{\max} - f_{\min})/\Delta f\rceil$ different possible values of f_{new}. As there are two possible rules to determine f_{set} from f_{new}, see rules (A) and (B) in subsection 5.2.2, we find $4\lceil(f_{\max} - f_{\min})/\Delta f\rceil$ as an upper bound on the total number of states, see Equation (5.2).

$$|S| \leq 4 \cdot \left\lceil \frac{f_{\max} - f_{\min}}{\Delta f} \right\rceil \tag{5.2}$$

The detailed system setup is described later on in section 5.5, but for the purpose of the example, we only need the governor parameters $f_{\min}$, $f_{\max}$, Δf_{rel} and the frequency levels from Table 5.1 for the two analysed platforms Haswell-i7 and ARMv7-Mobile. Here, Δf_{rel} is the frequency step-size given as the percentage of the maximum frequency $f_{\max}$, see Equation (5.3).

$$\Delta f_{\text{rel}} = \frac{\Delta f}{f_{\max}} \Rightarrow \Delta f_{=} \Delta f_{\text{rel}} \cdot f_{\max} \tag{5.3}$$

We now use Equation (5.1), the rules to choose f_{set} (see subsection 5.2.2) and the parameters of the governor for Haswell-i7 to derive the state diagram of the frequency covert channel on this device. Furthermore, we constrain the covert channel such that a symbol represents either an increase or decrease of the operating frequency. Although staying at the same frequency is possible, we do not consider it a valid symbol. This restriction is necessary due to implementation artefacts of the governor, which we present in subsection 5.6.1. The state diagram is depicted in Figure 5.4.

Using the state diagram of the frequency covert channel, we

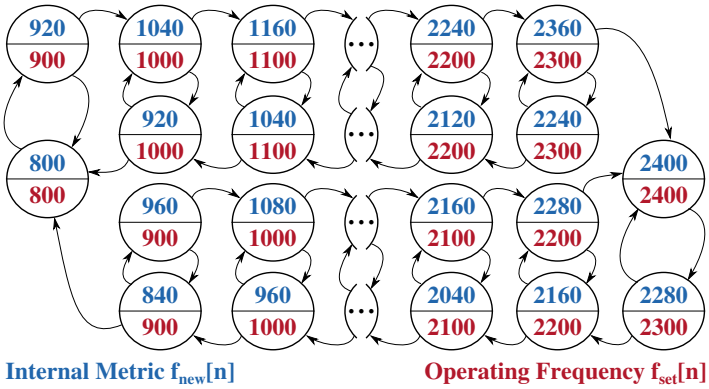
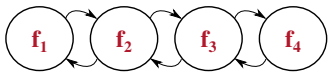


Figure 5.4: The state diagram of the frequency covert channel for Haswell-i7 and the conservative governor. Every state is defined by the tuple $(f_{\text{new}}, f_{\text{set}})$, e. g., $(920, 900)$.



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 5.5: Example of a state diagram (left) to connection matrix (right) conversion.

derive the connection matrix $\mathbf{A}$ of size $|S|\times|S|$. An example of the derivation of the connection matrix $\mathbf{A}$ from the state diagram is illustrated in Figure 5.5. Now we calculate the channel capacity bound, as explained in section 4.6. We use Equation (5.4),

$$C = \log_2 \lambda_1 \tag{5.4}$$

where λ_1 is the principal eigenvalue of $\mathbf{A}$. For Haswell-i7, our analysis yields an upper bound on the channel capacity of

$C = 0.972$ bits per channel use. If we apply this scheme to our second platform, ARMv7-Mobile, we determine a capacity of $C = 0.982$ bits per channel use.

As outlined in Table 5.1, Haswell-i7 has a governor sampling period $T_s = 80\,\text{ms}$ by default and ARMv7-Mobile $T_s = 100\,\text{ms}$. Using Equation (5.5), this yields a maximum bandwidth $B_{\text{max}} = 12.15\,\text{bits per second (bps)}$ and $9.82\,\text{bps}$, respectively. As we are considering the frequency covert

$$B_{\text{max}} = \frac{C}{T_s} \tag{5.5}$$

channel to be noiseless in this abstraction, this also equals the maximum throughput.

Finally, we note that these calculated upper capacity bounds cannot be achieved in a practical setting, as they are based on idealised conditions. The bounds assume a transmission scheme that maximises the information content per channel use, no errors due to interferences of other processes, perfect synchronisation between the source and sink applications and no implementation artefacts of the governor. In addition, the bound calculation is based on the assumption that every state transition is observable. We expect a further degradation of the achievable capacity for two main reasons:

1. The sink application can only observe f_{set} and it is not possible to determine f_{new}.
2. Some state transitions cannot be detected, for example the transition from state $(1040, 1000)$ to $(920, 1000)$ on Haswell-i7.

Therefore, we present an implementation of the channel in section 5.8 using RNNs, to validate the capacity bounds and determine their tightness.

5.5 Experimental setup and initial tests

Our experiments are carried out on two diverse hardware platforms representative for two kinds of mobile devices:

1. A Lenovo ThinkPad T440p laptop, based on a 4^{th} generation Intel Core i7-4710MQ quad-core processor. It can be clocked at frequencies in the range from 800 MHz to 2.4 GHz in 15 frequency levels, excluding the Intel Turbo Boost;

2. An Odroid-XU3 board, featuring a Samsung Exynos 5422 System-on-Chip (SoC) with an ARM big.LITTLE processor with two quad-core clusters of Cortex-A7 and Cortex-A15 cores. The LITTLE cluster is clocked at frequencies in the range of 200 MHz to 1.4 GHz in 13 frequency levels; the big cluster in a range of 200 MHz to 2.0 GHz in 19 levels.

In the rest of the chapter, we refer to platform 1 as Haswell-i7 and to platform 2 as ARMv7-Mobile. While Haswell-i7 is representative for business laptops, ARMv7-Mobile is representative for hand-held devices (ie tablets or smartphones). Haswell-i7 is running Ubuntu 16.04.1 LTS and ARMv7-Mobile is operating on Ubuntu 14.04.4 LTS.

To support reproducibility of all of the experiments, we defined a strict experimental environment for our two chosen platforms based on the Experiment Orchestration Toolkit (ExOT) data processing and experiment control design (see chapter 2 and [MKT20b]). On both platforms we use the `/dev/cpu_dma_latency` interface of the Linux kernel to set the maximum wakeup latency to 0 μs. As the deepest allowed sleep state is POLL (C0 active), the system cannot go into sleep mode, which could cause changes in the timing behaviour of the governor. We place both devices in an air-conditioned server

Param.	Value	Param.	Haswell-i7	ARMv7-Mobile
Δf_{set}	5%	T_s	80 ms	100 ms
I_{low}	20%	f_{min}	0.8 GHz	0.2 GHz
I_{high}	80%	f_{max}	2.4 GHz	2.0 GHz
f-levels in 0.1 GHz steps			w/o {1.2, 2.0}GHz	all

Table 5.1: Parameters of the conservative governor and the characteristics of the platforms Haswell-i7 and ARMv7-Mobile.

room with an ambient temperature of $\approx$ 23 C°. Furthermore, we fix the fan speed of both platforms Haswell-i7 and ARMv7-Mobile to the maximum level[7]. With these measures, we minimise any thermal side effects. To avoid scheduling artefacts, we run the source and sink application in the `SCHED_FIFO` scheduling class at the highest priority on both platforms, using the `pthread_setschedparam()` interface.

Source and sink application are executed on two separate cores that share a frequency domain. The source application is placed on core 4 (physical core 2) and the sink application on core 0 (physical core 0) of Haswell-i7. On ARMv7-Mobile, we run the two apps on the cores 6 and 7, two big cores, although the channel would still work if the two applications were executed on two of the LITTLE cores. The applications are pinned to the respective cores using the `pthread_setaffinity_np()` interface. Moreover, we do not alter the standard settings of the governor, presented in Table 5.1. Finally, we note that during all the experiments, the systems are only running the source, the sink and the default system services of the OS.

[7]Haswell-i7:
```
echo 'level 7' > /proc/acpi/ibm/fan
```
ARMv7-Mobile:
```
echo 255 > /sys/devices/odroid_fan.14/pwm_duty
```

5.5.1 Platform-dependent governor behaviour

We conduct initial experiments on Haswell-i7 using the 4.4.0-112-generic Linux Kernel. The initial experiments revealed that, in practice, the governor behaviour substantially deviates from the ideal behaviour described in section 5.2. These deviations cause unexpected frequency scalings, which lead to problematic transmission behaviour.

5.5.1.1 Timing issues

The first problem arises due to the fact that the governor is not called with a period of T_s. Therefore, the total measurement time of the governor (t_m) can vary (see subsection 5.2.2). According to a previous communication exchange with one of the acpi developers, this behaviour may be caused by one of two reasons, (i) the core is not active, or (ii) in kernel versions older than 4.7 calls to the governor are managed by a *work queue* which can cause delays if the queue is congested. After version 4.7, the developers decided to change the implementation and call the governor using the scheduler to increase the governor call periodicity. Therefore, this behaviour might not be observable using newer versions of the Linux kernel.

As the system cannot detect whether a governor call has been delayed because of the work queue, it always applies the policy implemented for non-active cores. This policy defines that the system should give rescheduled threads a chance to start on a reasonably high frequency after a core has been idle, implemented with Listing 5.2 in the governor. This piece of code causes a further deviation from the ideal governor behaviour described in subsection 5.2.2. The developers assume that the core was idle if the time between two consecutive governor calls (`wall_time`) is longer than twice the governors sampling period T_s (`sampling_rate`). In this case,

```
if (unlikely(wall_time>(2*sampling_rate)
        && j_cdbs->prev_load)) {
    load = j_cdbs->prev_load;
    j_cdbs->prev_load = 0;
} else {
    load = wall_time-idle_time) / wall_time
            * 100;
    j_cdbs->prev_load = load;
}
```

Listing 5.2: Snippet from the `cpufreq_governor.c` file in kernel version 4.4.0. If the time between two consecutive governor calls exceeds twice the sampling period, the current utilisation measurement is discarded, and the last one is used, if it is not zero.

the governor discards the current utilisation measurement (`wall_time-idle_time/wall_time*100`) and uses the last measurement (`j_cdbs->prev_load`), if it is not zero. As the replacement of the measured utilisation must not happen multiple times in a row, the governor performs a destructive read on the last measurement (`j_cdbs->prev_load=0`).

By setting up a customised Kernel, we were able to insert additional debug outputs to observe the timing behaviour of the governor. A simple experiment that illustrates the complex behaviour is illustrated in Figure 5.6. The intention of the experiment was to scale up once and then scale down. This frequency manipulation should only need two channel uses. However, the governor is not called periodically, as indicated by the tics on the x-axis. The governor is called the second time at 105.4 ms and the third time at 285.9 ms. Therefore, the elapsed time between the two calls is 180.5 ms, which is higher than twice the sampling period T_s. Hence, the governor assumes that the core was idle and does not update the utilisation, but keeps the previously-measured utilisation of 80%. As a result, the

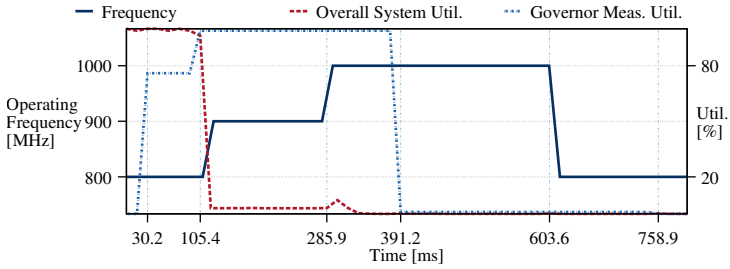


Figure 5.6: The plot shows the unexpected governor behaviour due to non-periodical governor calls, indicated by the vertical dashed lines, and architecture-dependent available operating frequencies. At the third call (285.9 ms) the governor assumes that the core was idle and does not update the utilisation value, which causes a frequency upscaling while the utilisation was below 20%. Furthermore, rather than observing two steps from 1000 MHz to 900 MHz and then to 800 MHz, due to limited visibility of governor states only one frequency scaling can be observed at the fifth governor call (603.6 ms).

governor scales the frequency up instead of down, despite the fact that the utilisation is below the lower utilisation threshold of 20%.

5.5.1.2 State transition issues

We already mentioned the second problem that may arise during a transmission: some internal state transitions are not visible as frequency changes of the core. In such a case, a state $(f_{\text{new}}, f_{\text{set}})$ is not completely observable, as it is not possible to determine f_{new} with a single measurement. Therefore, contrary to our assumption, when deriving the capacity bounds, we cannot use all states to encode symbols. This problem is also illustrated in our simple example in Figure 5.6. Considering the governor state diagram in Figure 5.4 and the channel input, we

conclude that the governor is in the state (1040, 1000) after the third governor call (285.9 ms). As the utilisation is lower than 20% after the third governor call and the observed frequency is 1000 MHz, the sending source application would expect the frequency to decrease to 900 MHz. However, with the fourth governor call, the governor moves from state (1040, 1000) to (920, 1000), which cannot be observed as the frequency stays at 1000 MHz. At 603.6 ms the governor is called the fifth time and the state changes from (920, 1000) to (800, 800), resulting in a visible state transition from 1000 MHz to 800 MHz.

5.5.1.3 Lessons learned

This simple experiment highlighted the main issues that arise when manipulating the operating frequency by changing the CPU utilisation. Therefore, we expect a degradation of the bandwidth of the frequency covert channel, caused by three problems:

1. The governor is not called periodically with the sampling period T_s. This will require additional overhead, as the transmission scheme cannot assume constant symbol durations.

2. Due to the variations of the governor call times, the governor may use old utilisation measurements instead of the actual one (see Listing 5.2). This can lead to unexpected operating frequency scalings during the data transmission.

3. Not all states can be used to transmit information, as some of them cannot be identified by just observing the operating frequency.

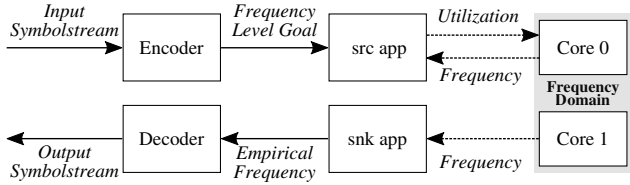


Figure 5.7: Block diagram of the transceiver system with signal flow indicated by solid arrows and indirect influences of one component on another with dashed arrows.

5.6 Channel implementation

Figure 5.7 illustrates the structure of the frequency covert channel implementation. In this section, we describe the transmission scheme used and the two applications source (`src`) and sink (`snk`).

5.6.1 Transmission scheme

Considering the conservative governor functionality outlined in section 5.2 and its practical behaviour analysed in subsection 5.5.1, as well as the standard parameters outlined in Table 5.1, we derive a robust transmission scheme, based on the following constraints:

1. As the timing of the frequency scaling is not foreseeable by our applications, a transmission scheme relying on fixed symbol length is not possible. Therefore, valid symbols can only be encoded into frequency changes, rather than using absolute frequency values.

2. Due to the functionality of the governor (see section 5.2), symbols can only be based on an incremental change of the frequency level.

3. It is not possible to decrease the frequency if the core is at

f_{min}, nor is it possible to increase it if the core is at f_{max}. In addition, we assume that the core frequency is f_{min} when the transmission starts. This is a valid restriction as our threat model (see section 5.3) defines that the attack will only start after the system was idle for some time. Thus, we have to make sure that the first channel use increases the frequency of the core. In addition, we have to restrict the number of consecutive symbols that would de- or increase the frequency.

4. We have to take into account the non-ideal governor behaviour shown in subsection 5.5.1, i. e., unexpected frequency scalings during a transmission. As our source application cannot forecast but only detect and react to frequency scalings, we have to tolerate a frequency uncertainty of one level.

5. Due to the characteristics of the frequency covert channel, we know that an error that is not recognised and corrected by the source application will corrupt all following symbols.

Considering these constraints, we define the following transmission scheme. To avoid that one non-correctable error corrupts all following symbols, the symbol-stream is divided into packets of fixed length. Each packet has a pre- and a postamble. The preamble consists of scaling up the frequency to the 10^{th} frequency level and then scaling down by 2 levels to reach the centre frequency. Similarly, the postamble consists of scaling up 2 levels and then scale down to the lowest frequency level. By going back to the lowest frequency level, we guarantee that the channel is reset after every packet and ensure that no error is dragged on from one packet to another. In addition, there are two data symbols, 0 and 1. A 0 is transmitted by scaling up 2 levels and then back to the centre frequency, a 1

is transmitted by scaling down 2 levels and then back to the centre frequency, respectively.

5.6.2 Source application

As shown in Figure 5.7, the main task of the source application is to utilise the core such that the frequency is scaled depending on the input frequency level gaol `flg`. Moreover, the source application compensates for non-observable state transitions, governor miss-scalings and non-periodical execution of the governor (see subsection 5.5.1) by detecting and tracking the current frequency level.

The core functionality of the source application is implemented sub-function `force_scale()`, illustrated in Figure 5.8. The inputs of the function `force_scale()` are (i) the current frequency level threshold `flg` describing the desired frequency level, and (ii) the current measured empirical frequency `ef_cur`, which is at the same time also an output. `ef_cur` is first set in the initialisation phase of the source application and updated every time `force_scale()` is called.

`force_scale()` is called whenever a new symbol is ready to be transmitted. First, the sub-function `get_ef()` is called to determine the low and high threshold for the empirical frequency. As measurement can vary slightly, the source application uses these thresholds to define an interval within the empirical frequency has to be for a certain frequency level, rather than using fixed values.

Next, the source application checks whether the current empirical frequency measurement `ef_cur` lies within or outside of the low and high threshold. Here, we distinguish three cases:

(I) If `ef_cur` is smaller than the lower threshold `ef_low`, the frequency needs to be scaled up (`scaling=UP`).

(II) If `ef_cur` is bigger than the higher threshold `ef_high`, the

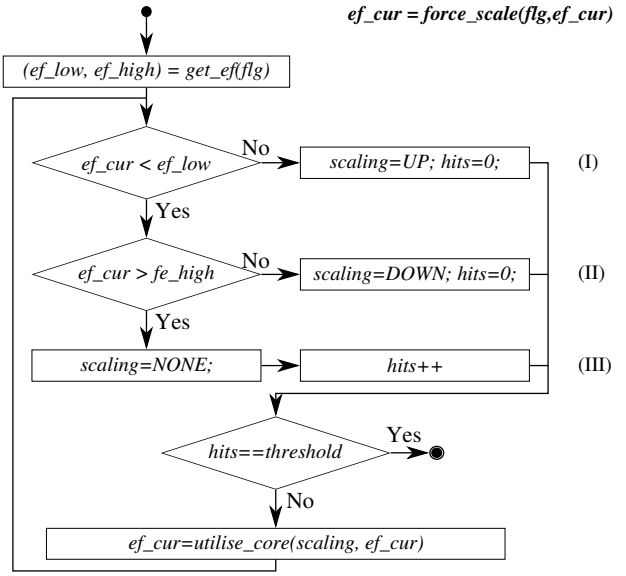


Figure 5.8: A simplified flowchart of the source application function `force_scale()`, using pseudocode with C-like notation. The inputs are the current frequency level threshold `flg` and the current empirical frequency value `ef_cur`, with `ef_cur` also being the output of the function.

frequency needs to be scaled down (`scaling=DOWN`).

(III) If `ef_cur` is between the two thresholds, the desired frequency level has been reached and no more scaling is required (`scaling=NONE`).

Case (III) is called a *hit*. In case of a hit, counter variable `hits` is incremented, otherwise `hits` is set to zero. As faulty measurements may occur, a symbol transmission (frequency scaling) is only considered successful if `threshold` consecutive hits occur.

If a symbol transmission was successful, the `force_scale()` function returns. In case the transmission was not completed successfully, the function `utilise_core()` is called to force the desired frequency scaling. The inputs to `utilise_core()` are the desired scaling direction `scaling` and the current empirical frequency `ef_cur`, which are used to determine the parameters needed to generate the appropriate core utilisation. The function `utilise_core()` also implements a so-called *backchannel*, which allows the source application to update the current empirical frequency based on the timing measurements, as described in section 5.3. The timing measurements are done using a tight loop similar to the one in the `cpuburn` stress-test[8] (see Listing 5.1) and `gettimeofday()`. One timing measurement is performed during the initialisation of the application to get a reference value, which is used to normalise all timing measurements to determine the empirical frequency.

The main function of the source application implements a timeout, which aborts sending of a packet after a pre-defined time and restarts the transmission process.

5.6.3 Sink application

The frequency is indirectly measured by the sink application, with a sampling period T; by default $T = 20\,\text{ms}$. The sink application inspects the frequency using the same timing measurement method as the source application (also see section 5.3). In order to minimise measurement uncertainty, the sink application performs multiple timing measurements per data point. These measurements are then averaged to determine the empirical frequency. To increase the accuracy of the empirical frequency measurements, the length of the tight loop used for the timing measurements can be increased.

[8] `patrickmn.com/projects/cpuburn`

However, a long, tight loop causes more utilisation, which leads to a higher channel interference. Therefore, we need to tune the length of the tight loop depending on the attacked device to achieve a good trade-off between accuracy and interference. The tuning is done manually before the experiments are started. However, this tuning could be automated using a more sophisticated design of the sink applications, but we leave this for future work. All time measurements are performed using `gettimeofday()`, which proves precise and lightweight enough for our purposes.

All samples are stored in a pre-allocated in-memory `log` and dumped at the end of the execution to a log-file. After the experiment has terminated, the log-file are analysed offline.

5.7 A Recurrent Neural Network as signal decoder

In this section, we give a brief overview of the underlying techniques used to build a Recurrent Neural Network (RNN) based signal decoder. We base the decoder design on Connectionist Temporal Classification (CTC) introduced by Graves [Gra12, Chapter 7], as we categorise the decoding as time-sequence classification with variable length symbols. CTC is the state-of-the-art technique for classification of (time-) sequences. It allows a RNN to make soft labelling decisions at every timestamp and calculate the probability of the correct sequence. By using this probability for the loss function, the RNN is able to learn to predict the label sequence without the pre-knowledge of the occurring locations and quantity of the labels. CTC allows us to implement the decoder with a single RNN and does not require the RNN to be combined with Hidden Markov Model (HMM). Furthermore, Graves [Gra12]

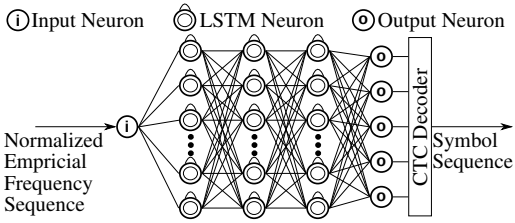


Figure 5.9: Network architecture example of a signal decoder based on a Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) neurons using Connectionist Temporal Classification (CTC).

already stated that CTC often even outperforms pure HMM or RNN-HMM hybrid models.

We employ a RNN based on Long Short-Term Memory (LSTM) units to make the decoding adaptive, and thus less error-prone, compared to static decoders. In contrast to simple feed-forward Neural Networks (NNs), RNNs have a feedback loop that allows us to capture temporal information of the data. An RNN example architecture is illustrated in Figure 5.9. We give more detailed information on RNN and the advantage of LSTM units in section 6.4.

The RNN we use in this chapter consists of one input neuron, bi-directional LSTM hidden layers and one layer that is fully connected to the last hidden layer. A bi-directional LSTM layer basically consists of two sub-layers, where one layer traverses the data from past to future and the second one from future to the past. Graves [Gra12] has empirically evaluated that bi-directional networks show better value and temporal accuracy than simple uni-directional networks. As we run our model on dedicated hardware, the complexity is not a limiting factor and, therefore, we favour more complex bi-directional networks over simpler uni-directional ones.

The output layer of our model consists of $Y + 1$ neurons, where Y is the number of used symbols. One output neuron is dedicated to each symbol used in the transmission, and one additional one is used for the blank symbol. The blank symbol is used as a separator for the other symbols and is placed by the RNN whenever no other symbol seems probable. It is also used to separate two sequential occurrences of the same symbol. The output of the output layer is then fed to the so-called *CTC-decoder*, which converts this output into the final symbol sequence.

The downside of a RNN based decoder is that it needs a lot of training data and high computational overhead for training. However, the training of the network can be done offline using dedicated hardware. Moreover, generating data for commercially available devices like the ones analysed in this chapter is easily possible, as the platform setup can be replicated. The training of the RNN is crucial for the performance of the decoder. If the RNN is not designed fitting the complexity of the decoding task, the training is badly parametrised or the training data is of bad quality, the decoder might not work at all.

5.8 Experimental analysis

We first analyse a static decoding strategy. The static decoding strategy is based on an empirical frequency-to-frequency level mapping, which we determined experimentally. An example of a message transmission with a packet length of 5 bits is illustrated in Figure 5.10. Starting from the top, plot (a) shows the input symbols, where S indicates the preamble and E the postamble. Plot (b) depicts the goal frequency level input to the source application. According to this frequency goal and

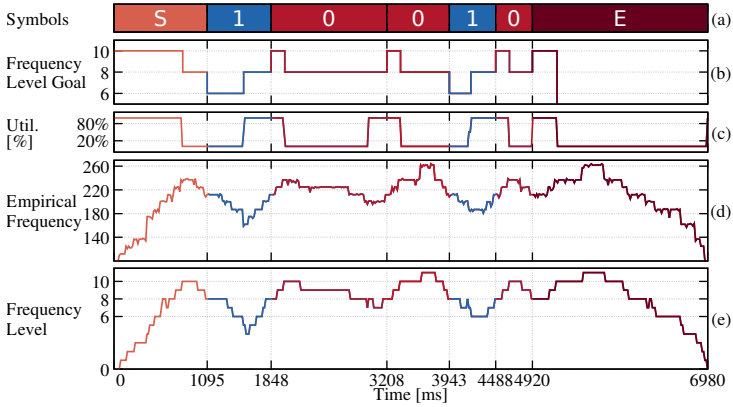


Figure 5.10: The input symbol stream (a), is converted to the goal frequency level (b). Using this input, the source generates the utilisation trace (c). This utilisation causes frequency scalings visible in the empirical frequency measurements trace (d). By filtering and discretise the frequency levels trace (e) is obtained, to reconstruct the symbol stream.

input through the backchannel, the source application tries to generate the utilisation presented in plot (c). Plot (d) shows the empirical frequency measurements of the sink application, including the measurement artefacts. Lastly, plot (e) illustrates the processed empirical frequency trace.

Most of the measurement artefacts shown in (d) are eliminated by the offline post-processing. In the offline post-processing, we apply an average filter with a window size of 9 samples and discretise the empirical frequency measurements. The discretisation is based on the platform-specific empirical frequency-to-frequency level mapping. Although the frequency scaling is not exactly as intended by the source application, the example shows that it is still possible to correctly decode the signal. This is thanks to the source application and coding measures which compensate for

most of the governor implementation artefacts mentioned in subsection 5.5.1. For example, at the end of the second bit (shortly before 3208 ms), the frequency level drops below the centre frequency. To compensate for this drop, the source application increases the utilisation again to force the system back to the centre frequency.

While the static decoding implementation worked fine for short and few packets, detailed experiments reveal its limitations. Using the static decoding scheme, we were not

$F(f(t))$	...	ideal transformation of operating frequency to empirical frequency
$f_e'(t)$	...	observed empirical frequency
M	...	multiplicative random interference factor
A	...	additive random interference factor

$$f_e'(t) = M \cdot F(f(t)) + A \tag{5.6}$$

able to reproduce the transmission error rates reliably when using multiple datasets consisting of many packets each. Packet error rates varied between 20% and up to 60% for repeated transmissions of the same 200 packets with a length of 8 bit each. Unlike in our initial tests, the measurement artefacts could not be fully compensated by the offline post-processing and generated a substantial amount of symbol errors. The static decoding strategy failed because the system showed a higher sensitivity to interferences than we originally anticipated. The caused by the traces containing (i) multiplicative, and (ii) additive random interference, as shown in Equation (5.6). These issues are induced by system interferences in the initialisation process and the measurements of the source or the

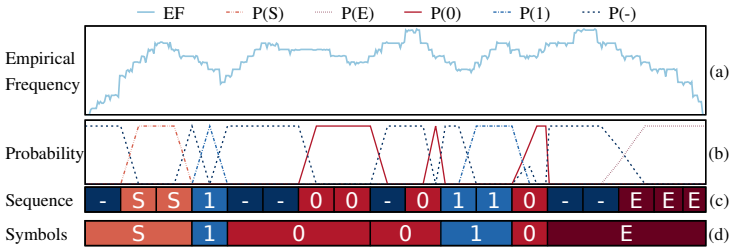


Figure 5.11: Example of a packet decoding using Connectionist Temporal Classification (CTC). Using the empirical frequency measurements (a) as input, the Recurrent Neural Network (RNN) determines the symbol probabilities (b). The Connectionist Temporal Classification (CTC) decoder uses the probabilities to generate the label sequence (c) and then determine the final output symbol stream (d), equal to the input shown in Figure 5.10.

sink application. They cause the static mapping from empirical frequency-to-frequency level to be incorrect.

5.8.1 Recurrent Neural Network based decoding

To address the limitations of the static decoding scheme, we employ a RNN as a signal decoder that takes a normalised empirical frequency measurement as an input. The LSTM layers contain 72 neurons each using *tanh* as the activation function for the output of the neurons and *sigmoid* for the gates. The output layer consists of 5 neurons using the *softmax* activation function. We need 5 output neurons for the 5 possible labels *S* (preamble), *E* (postamble), 0, 1 and - (blank). An example of a packet decoding is illustrated in Figure 5.11 for the same packet as used in Figure 5.10. Plot (b) depicts the probabilities for the different symbols, determined by the RNN. The plots (c) and (d) illustrate the two-step task the CTC-

decoder performs. First, the one-hot encoded label vectors are converted to a label sequence and further to the final output symbol sequence.

For the training, we generated packets with a random length between 8 and 32 bits, summing up to a total of 32000 bits. By using random length packets, we prevented the RNN from just learning the length of the packets, instead of correctly classifying the preamble and postamble. Furthermore, we did not need to re-train it for each different packet length again. We concatenated the recorded traces such that the resulting training sequences consist of 5000 empirical frequency readings. Such a sequence is called a sample. In the training, we used 200 of those samples, grouped into 10 mini-batches, where 20 samples (1 mini-batch) is used for validation. The training itself consisted of two phases (i) 200 epochs for the initial training and no regulative measures, and (ii) 1000 epochs for the fine-tuning, using a clip-norm of 70.0 to prevent gradient explosion. We used a Nesterov accelerated stochastic gradient descent optimiser with a learning rate of 0.001, a momentum of 0.9 and a decay of 0. Furthermore, if the validation loss decreased, we saved the RNN model after each epoch, to make sure we used the best model in the final decoder.

5.8.2 Threat level assessment

We evaluate the robustness of our transmission scheme for different payload length of the packets, i. e., 8, 16, 32 and 64 bit. For every packet length, we send 5 traces of 200 packets each, with random payload bits generated using the python random packet. The payload of each of our traces is big enough to contain multiple 512 bit elliptic-curve cryptography keys [Bar16], see Table 5.2.

To evaluate the throughput degradation of the frequency covert channel caused by the governor implementation

artefacts, we compare the results of our experiments with two theoretical baseline platforms. These baseline platforms have the same parameters as the respective real platform Haswell-i7 and ARMv7-Mobile (see Table 5.1), but we assume that none of the frequency covert channel and the governor implementation artefacts occur (see subsection 5.5.1 and section 5.6). We

$$\mathrm{TP}_{base} = \frac{payload}{T_s \cdot (CU_{PRE} + CU_{BIT} \cdot payload + CU_{POST})} \tag{5.7}$$

calculate the throughput TP_{base} of the baseline platforms using Equation (5.7). Here, *payload* is the number of bits per packet. CU_{PRE} is the number of channel uses for the preamble and CU_{POST} for the postamble, which are both 12. The number of channel uses per bit, denoted as CU_{BIT}, is 4. The respective throughput for each packet length is given in Table 5.2 for both baseline platforms.

5.8.2.1 Achievable rates and error probability

The upper diagram in Figure 5.12 shows the achieved throughput in bps, calculated as an average of each single packet throughput for all packets that have been transmitted without error. The packet throughput is calculated by dividing the number of payload bits in a packet by the time needed to send the whole packet, including preamble and postamble. The middle diagram presents the degradation of the throughput of the real platforms in comparison to the baseline platform, i. e., the percentage of throughput loss. The Packet Error Rate (PER) in % is illustrated in the bottom diagram.

Our experimental analysis shows that the achievable throughput is substantially lower than the capacity bound determined in section 5.4. The difference between capacity

bound and maximum throughput of our baseline platforms is caused by the transmission scheme. Our transmission scheme needs 4 channel uses per bit in an error-free environment without governor implementation artefacts. Therefore, we only achieve a throughput of 0.25 bits per channel use for packets of infinite length. This is significantly lower than the upper channel capacity bound of 0.972 bits per channel use for Haswell-i7 and 0.982 for ARMv7-Mobile, respectively. Yet, the high complexity of the transmission scheme is necessary due to the platform-dependent behaviour and implementation artefacts of the governor (see subsection 5.5.1).

As plot (b) in Figure 5.12 outlines, the real platforms Haswell-i7 and ARMv7-Mobile show a further throughput degradation when comparing them to the respective baseline platform, i. e., relative decrease of the throughput on the real platforms compared to the baseline platforms. This degradation is caused by corrections the source application has to apply due to unexpected frequency scalings. While we expect an increasing trend for the throughput with rising packet length, both platform Haswell-i7 and ARMv7-Mobile show a plunge for packets with a length of 32 bits. This shows that the platform dependencies and implementation artefacts of the governor cause substantial disturbance in the channel, independent of the duration of the packets.

We observe increasing packet error rates (PERs) for increasing packet lengths for both platforms, which correlates with the higher likelihood for bit errors, i. e., non-recoverable unexpected governor behaviour for increasing packet length. However, on ARMv7-Mobile the PER is lower by almost a factor of 4, than on Haswell-i7. We chalk the lower PER up to a combination of two reasons: (i) different Kernel versions of ARMv7-Mobile and Haswell-i7, and (ii) the architecture of the processor of ARMv7-Mobile. ARMv7-Mobile is operated on

Packet Payload	Payload Bits per Trace	Haswell-i7 TP_{base}	ARMv7-Mobile TP_{base}
8 bit	1600 bit	1.79 bps	1.43 bps
16 bit	3200 bit	2.27 bps	1.82 bps
32 bit	6400 bit	2.63 bps	2.11 bps
64 bit	12800 bit	2.86 bps	2.29 bps

Table 5.2: Packet payload, the corresponding number of data bits per trace and the throughput on the baseline platforms.

Kernel version 3.10.96, while Haswell-i7 uses 4.4.0-112-generic. As we presented in subsection 5.5.1, different Kernel versions can cause differences in the behaviour of the governor due to implementation artefacts. Furthermore, on Haswell-i7 all cores share one frequency domain, whereas the LITTLE and the big cores in ARMv7-Mobile have separate frequency domains. Due to the fact that the ARMv7-Mobile will try to utilise the LITTLE cores as much as possible and only migrate processes into the big cluster if necessary, we experience fewer interferences for our transmission.

Our analysis suggests that shorter packets show lower PERs but also yield less throughput. However, we cannot make a definite statement that this is true for every platform, as our analysis also shows platform variations. In order to find the best configuration for a specific platform, further exploration considering the packet length, as well as possible error detection or error correction codes and protocol overhead, has to be done.

5.8.2.2 Interference by other processes

In our attack scenario, we assume that the attacked platform is idle. This is a valid expectation, considering the usage pattern of mobile and embedded platforms. Nonetheless, other processes could still increase the core utilisation and interfere with the channel. While short utilisation bursts caused by background processes and idle applications can be handled, a high utilisation floor would cause more problems.

Short utilisation bursts have the same effect on the governor behaviour, as the issues introduced by Listing 5.2, namely, a single unexpected frequency scaling. We show with our implementation that single unexpected frequency scalings can already be handled with measures like a backchannel, an appropriate transmission scheme (see section 5.6) and a smart

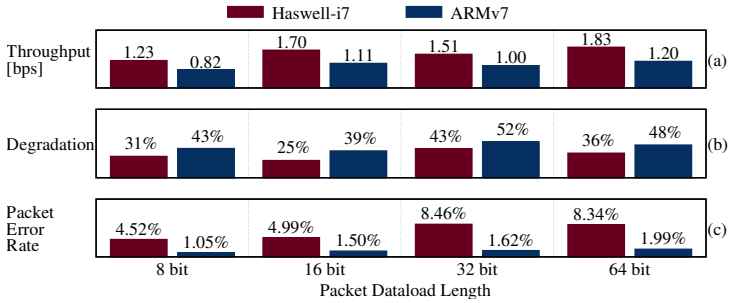


Figure 5.12: Increasing packet length versus (a) throughputs, (b) throughput degradation in relation to baseline platforms and (c) Packet Error Rate (PER). As we expect that increasing packet lengths lead to growing throughputs, the throughput plunge at 32 bit packets indicates high channel disturbance.

decoder (see subsection 5.8.1). In contrast, if the interfering utilisation is constant and high enough to force the governor to scale to a frequency higher than the lowest possible frequency, the number of reachable frequency levels is reduced. The reduction of frequency levels can only be compensated in the design of the transmission scheme or by a smart source application.

We conclude that burst interference by other processes can be compensated during an attack, while permanent interference might make an attack impossible. Thus, launching an attack while the platform is idle maximises the chance of success.

5.8.2.3 Threat Classification

To assess the threat potential of the frequency covert channel, we consider the small message criterion by Moskowitz and Kang [MK94]. Therefore, we assume the following scenario: The highly sensitive data is a cryptographic key that could be

used to enter a company network or servers. The cryptographic key is saved on the hand-held platform of a company, such that the employee can access the company network and servers while on a business trip. Today, the National Institute of Standards and Technology considers elliptic-curve cryptography with a key size of 512 bit to be highly secure [Bar16].

We assume that an attacker manages to deploy a setup of the slowest implementation of our frequency covert channel on the hand-held platform. Furthermore, the attacker uses one parity bit per packet for error detection and a handshake scheme. The handshake scheme involves sending an acknowledge from the sink to the source application after 8 data packets to acknowledge whether the packets were received correctly. An acknowledge packet does not contain a

T ... Total number of packets
D ... Number of data packets
k ... Transmission round
A ... Number of acknowledgement packets
e ... Error probability

$$T = \left\lceil \sum_{k=1}^{\infty} \left(D \cdot e^{(k-1)} \right) \right\rceil + \left\lceil \sum_{k=1}^{\infty} \left(A \cdot e^{(k-1)} \right) \right\rceil \tag{5.8}$$

$$= \left\lceil D \cdot \frac{1}{1-e} \right\rceil + \left\lceil A \cdot \frac{1}{1-e} \right\rceil \tag{5.9}$$

$$= 86 \text{ packets} \tag{5.10}$$

parity bit and the 8 data bits correspond to the 8 data packets sent before. This results in $D = \lceil 512/7 \rceil = 74$ data packets and $A = \lceil D/8 \rceil = 10$ acknowledgement packets that need to be transmitted correctly.

In our scenario, the acknowledge packet is always trans-

mitted correctly, and only faulty packets are retransmitted. Therefore, in the transmission round $k = 1$, D data and A acknowledgement packets have to be transmitted; in transmission round $k = 2$, $\lceil D \cdot e\% \rceil$ data and $\lceil A \cdot e \rceil$ acknowledgement packets; in transmission round $k = 3$, $\lceil D \cdot e \cdot e \rceil$ data and $\lceil A \cdot e \cdot e \rceil$ acknowledgement etc.. As illustrated in Equation (5.8), the expected total number of packets that need to be transmitted can be expressed as an infinite row. This results in an expected total of $T = 86$ packets for an error-free transmission of the cryptographic key, as illustrated in the Equations (5.9) to (5.10). As each packet contains $b = 8$ bits and we assume an average throughput of $TP = 0.82$ bps, we determine a transmission duration of 13.98 min, as shown in Equation (5.11).

b ... Bits per packet $\qquad$ TP ... Average throughput

$$\frac{T \cdot b}{TP} = \frac{86 \text{ packets} \cdot 8 \text{ bits per packet}}{0.82 \text{ bits per second}} = 839.02 \text{ s} = 13.98 \text{ min} \tag{5.11}$$

Our analysis shows that we can expect the attack to be carried out successfully in approximately 14 minutes. Therefore, the attack could be executed, for example, within the lunch break of the employee. Furthermore, as our implementation and transmission scheme are rather naïve, this attack can be executed more efficiently. For instance, an attacker could find a more efficient symbol encoding and achieve throughputs closer to the channel capacity bound. Moreover, devices ranging from embedded systems used in mobile devices to server systems rely on DVFS, necessary to establish the frequency covert channel. Using this reasoning, we state that the frequency

covert channel poses a significant threat to a wide range of platforms.

5.9 Mitigation Strategies

Applications commonly require access to either a timer or timestamps during execution. These timers and timestamps enable applications to indirectly measure processing speed without the need for special OS permissions, as described in section 5.3. As timers cannot be easily restricted, mitigation strategies for the frequency covert channel require advanced mechanisms. We provide a brief, high-level outlook on possible mitigation strategies and list some examples, without claiming a complete overview of past work.

Similar to Intel SGX or the Arm TrustZone, an effective mitigation strategy could be based on hardware enclaves, i. e., not to share the frequency domain among different cores. Furthermore, the concurrent execution of applications with different security clearances on the same core has to be restricted.

Another possible solution is a detector using different characteristics of a process execution or the system architecture, i. e., memory access or instruction stream, and apply classifiers to identify malicious processes. Such a detector can either be realised in software using middleware or system calls [PZ13; Can+12; ADY13], or as dedicated hardware [Hoe+13; Ana+13; Ozs+15; YST16; CV14]. Using training frameworks, some existing malware detectors could be trained to detect the frequency covert channel [KRT16].

A more direct approach is altering the part of the system which is responsible for the information leakage [Sha+15]. As presented in section 5.4, the choice of governor is vital for

the performance of the frequency covert channel. Therefore, a smart governor design can reduce the available bandwidth of the frequency covert channel, such that the threat is negligible. For example, deviating from the expected governor behaviour by introducing randomness in terms of timing and frequency scaling will reduce the achievable channel bandwidth. Such behaviour has, by accident, already been implemented in the Linux Kernel 4.4.0 and led to a substantial throughput degradation. However, this strategy needs to be evaluated carefully, as it can also have a negative impact on the effectiveness of the governor in terms of energy saving.

Finally, attacks only work well on idle devices, i. e., when there is no additional load that influences the core frequency (see section 5.3). Detecting and protecting the idle-mode can be used correspondingly. For example, in the case of smartphones or tablets, the governor design could take device characteristics into account, i. e., controlling the idle mode depending on whether the screen is on or off[9].

5.10 Summary

In this chapter, we analysed a covert channel based on the frequency of the core, called frequency covert channel. We showed that the main threat posed by the frequency covert channel is the possibility to leak data to compromise the widely used security paradigm of permission separation and application isolation.

To this end, a source application utilises the core such that its frequency scalings encode the transmitted data. As a counterpart, a sink application detects the changes of the

[9] e. g., *InteractiveX* governor for Android, see `http://androidforums.com/threads/android-cpu-governors-explained.513426/`

frequency by repeatedly measuring the duration of a fixed set of operations. The measurement method used by the sink application does not require any elevated privilege levels or special function, and can therefore easily be deployed on any platform. We modelled the channel as noise-free time and value discrete to derive the corresponding channel capacity bounds. The capacity deviation method is applicable for every platform-governor combination that allows us to derive a channel state diagram. The channel capacity bound was used to compare different kinds of covert channels or help to estimate the associated security risk and develop mitigation strategies. For the conservative governor (see section 5.4) we determined the upper channel capacity bound of about 1 bit per channel use.

We developed transmission schemes for two distinct and representative platforms, Haswell-i7 and ARMv7-Mobile, which we experimentally evaluated. Our findings showed that it is possible to achieve throughputs of 1 to 2 bps with packet error rates between 1% to 8%. Furthermore, our experimental evaluation illustrated that the capacity and achievable throughputs with the frequency covert channel have a high dependency on the used hardware, frequency governor and OS version (see section 5.8). Last, we gave a brief outline of possible mitigation strategies to omit the threat presented by the frequency covert channel.

Despite the low capacity of the frequency covert channel, considering that (i) the attacker does not need any special permissions to establish the frequency covert channel, (ii) almost every current mobile multicore system and many embedded systems are affected, (iii) systems can often be compromised by leaking a relatively small amount of information, i. e., a cryptographic key or a password, and (iv) these systems are often idle, which makes the execution of the attack easier, we stated that this covert channel needs special attention.

6

Extracting runtime information via the thermal side channel

As Lampson [Lam73] outlined, there are two large groups of data leaks: (i) covert channels, where two colluding applications actively share information, and (ii) side channels, where information is extracted by observing an unaware system. While in chapters 2 to 5, we focused on covert channels to assess the threat potential of data leaks, in this chapter, we present a side channel attack.

Based on the finding that thermal sensor readings can be used to leak data, shown in chapter 3, we present the novel thermal side channel attack. This attack relies on openly available thermal sensor readings on current devices to extract runtime information of applications. We illustrate how to

This chapter is based on work presented in [MAT20].

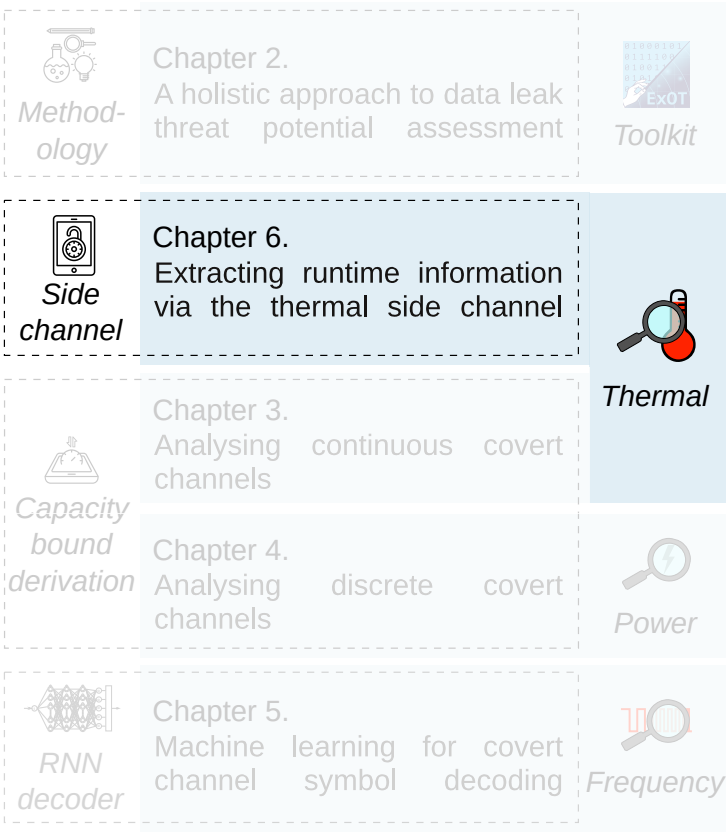


Figure 6.1: In chapter 6, we present the novel thermal side channel attack. This attack leverages openly available thermal sensor readings on smartphones to extract application runtime information.

employ Neural Networks and subsequent label processing to determine which application is running at which time on a commercial smartphone.

6.1 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. In this chapter, we show how to bypass this security paradigm 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 Store that allow the thermal information of an Android device

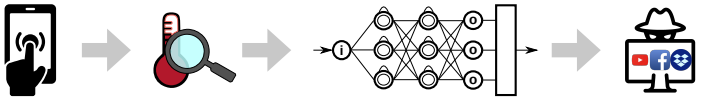


Figure 6.2: (a) Different applications are executed on a mobile device depending on the user input. (b) Thermal information provided by the Operating System is collected by a third-party application. (c) Analysis of the thermal data to determine the application sequence. (d) The application sequence is used to create a usage profile or detect other applications, causing security and privacy violations.

to be read, without the need for elevated privileges[1]. It has already been shown that these thermal readings may lead to security issues [DS05]. In particular, the thermal covert channel presented in chapter 3 shows the possibility of a data leak, as Ristenpart et al. [Ris+09] 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 chapter, we present a side channel attack based on thermal sensor readings of a mobile device, as depicted in Figure 6.2. 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

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

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 chapter are:

1. We present a novel 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.

6.2 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.

6.2.1 Attack scenario

We base our threat model on the scenario presented in Figure 6.2 and the information flow in Figure 6.3. A user runs a sequence of applications **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

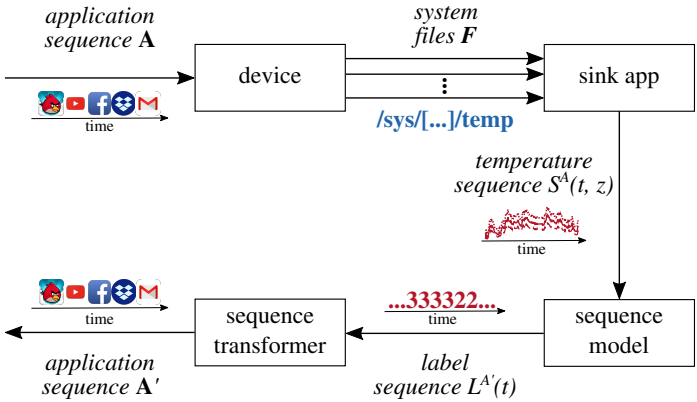


Figure 6.3: The information flow during a thermal side channel attack. A user executes a sequence of different applications **A** on the device. The sink application monitors the resulting heat generation from the device by reading the respective system files **F** for the different thermal zones $z \in \mathbf{Z}$. 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 $\mathbf{A'}$.

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* application as a benign game. Therefore, the adversary is able to monitor the temperature sensors of the mobile device.

The sink in Figure 6.3 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.

6.2.2 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 Experiment Orchestration Toolkit (ExOT), presented in chapter 2, 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. 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 6.4. The sequence transformer is based on classical

filtering algorithms and rules of condensing labels, as outlined in section 6.5.

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.

6.3 Data augmentation

The overall process to generate a large set of diverse thermal sequences is depicted in Figure 6.4. 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 shown in Equation (6.1), where β denotes the thermal coefficient, $T(t)$ denotes the temperature at time t, we have $t_2 \geq t_1$, and T^{idle} denotes the steady-state temperature. For convenience, we introduce the temperature function $C(\cdot)$ in Equation (6.2). It returns the temperature difference to the steady-state temperature $\Delta T(t_0 + \Delta t) = T(t_0 + \Delta t) - T^{\text{idle}}$ after time t and uses the initial

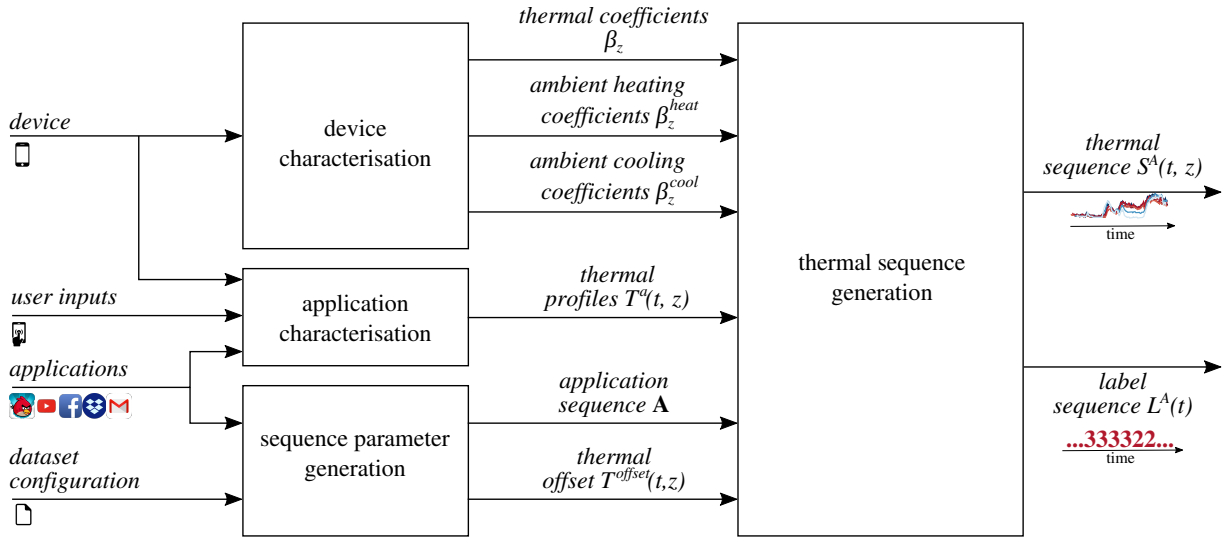


Figure 6.4: Overall data augmentation scheme structure. We generate a thermal sequence and its labels based on (i) the particular device and its measured thermal characterisation, (ii) the set of identifiable applications and their thermal characterisations, and (iii) the highly-configurable data set configuration. Using diverse configurations, we generate representative training data.

temperature difference $\Delta T(t_0) = T(t_0) - T^{\text{idle}}$, the thermal coefficient β, and the time difference t. In order to be able

$$T(t_2) = T^{\text{idle}} + (T(t_1) - T^{\text{idle}}) \cdot e^{-\beta(t_2 - t_1)} \tag{6.1}$$

$$\Delta T(t_0 + t) = C(\Delta T(t_0), \beta, t) = \Delta T(t_0) \cdot e^{-\beta \cdot t} \tag{6.2}$$

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 $C(\cdot)$.

6.3.1 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 β_z,
2. the ambient heating coefficient β_z^{heat}, and
3. the ambient cooling coefficient β_z^{cool}.

The internal temperature coefficient β_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 β_z for each thermal zone z using regression analysis of the temperature traces, i. e., fitting the temperature function $C(\cdot)$ in Equation (6.2) to the temperature measurements using non-linear least squares.

The two ambient coefficients for heating and cooling, β_z^{heat} and β_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 β_z, we use regression analysis to determine β_z^{heat} and β_z^{cool}.

6.3.2 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, Δt 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 β_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.

6.3.3 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^{th} application which is started at time t_i^{start} and closed at time t_i^{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.

Depending on the information in the dataset configuration, we generate the application sequence $\mathbf{A}$ 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 $\mathbf{T^{ambient}}$ 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^{ambient} at time t_i^{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 in Equation (6.3), where $t_{n+1}^{\text{enter}} = \infty$. Note that when the ambient temperature increases, we use β_z^{heat} as the thermal parameter for the thermal behaviour model of the zone, while for decreasing ambient temperatures, we employ β_z^{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.

6.3.4 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 con-

catenation 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 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 $C(\cdot)$ in Equation (6.4). 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 $1 \leq i \leq n_A$, 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

$$T^{\text{offset}}(t,z) = \begin{cases} \mathcal{U}(-1,1) + \Delta T_0^{\text{ambient}} \\ \qquad \forall\ 0 \le t < t_1^{\text{enter}} \\ \mathcal{U}(-1,1) + \Delta T_i^{\text{ambient}} + C\left(T^{\text{offset}}(t_i^{\text{enter}},z) - T_i^{\text{ambient}}, \beta_z^{\text{heat}}, t - t_i^{\text{enter}}\right) \\ \qquad \forall\ 1 \le i \le n\ \wedge\ t_i^{\text{enter}} < t \le t_{i+1}^{\text{enter}}\ \wedge\ T_{i-1}^{\text{ambient}} \le T_i^{\text{ambient}} \\ \mathcal{U}(-1,1) + \Delta T_i^{\text{ambient}} + C\left(T^{\text{offset}}(t_i^{\text{enter}},z) - T_i^{\text{ambient}}, \beta_z^{\text{cool}}, t - t_i^{\text{enter}}\right) \\ \qquad \forall\ 1 \le i \le n\ \wedge\ t_i^{\text{enter}} < t \le t_{i+1}^{\text{enter}}\ \wedge\ T_{i-1}^{\text{ambient}} > T_i^{\text{ambient}} \end{cases} \tag{6.3}$$

$$\overline{T}^a(t,z) = \begin{cases} T^a(t,z) \\ \qquad \forall\ t \le t_{crop} - t_{close} \\ T^a(t + t_{exec} - t_{crop}, z) + C\left(T^a(t_{crop} - t_{close}, z) - T^a(t_{exec} - t_{close}), \beta_z, t + t_{close} - t_{crop}\right) \\ \qquad \forall\ t_{crop} - t_{close} < t \le t_{crop} \end{cases} \tag{6.4}$$

$$S^A(t,z) = \begin{cases} T^{\text{offset}}(t,z) + \overline{T}_1^a(t,z) \\ \qquad \forall\ 0 = t_1^{\text{start}} \le t \le t_1^{\text{end}} \\ T^{\text{offset}}(t,z) + \overline{T}_i^a(t,z) + C\left(S^A(t_{i-1}^{\text{end}}, z) - \overline{T}_i^a(0,z), \beta_z, t - t_i^{\text{start}}\right) \cdots \\ \qquad \forall\ 2 \le i \le n_A\ \wedge\ t_i^{\text{start}} < t \le t_i^{\text{end}} \end{cases} \tag{6.5}$$

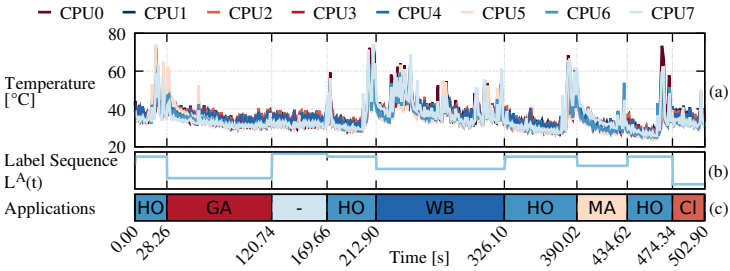


Figure 6.5: Thermal sequence build from different thermal profiles collected from a Sony Xperia Z5, see Equation (6.6). (a) illustrates the thermal traces, (b) the label sequence $L^{\mathrm{A}}(t)$ and (c) the application trace $\mathbf{A}$. Our data augmentation removes temperature discontinuities at application pre-emption points without distorting the thermal profiles.

thermal sequence. We obtain the thermal sequence $S^A(t, z)$ for each temperature zone z in Equation (6.5) Here, we have n_A applications with indices $1 \leq i \leq n_A$, where the first application starts at time $t_1^{\text{start}} = 0$. Note that we use the cropped temperature profiles, as determined in Equation (6.4). 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 6.5 illustrates the thermal sequence generated from the example application sequence $\mathbf{A}$ defined in Equation (6.6). The application labels are defined in Table 6.1, and the thermal profiles were collected from a Sony Xperia Z5 smartphone and

the settings outlined in section 6.6. The example illustrates

$$\begin{aligned}\mathbf{A} = (&<\text{HO},\ 0.00\,\text{s},\ 1.13\,\text{s}>,<\text{GA},\ 1.13\,\text{s},\ 4.83\,\text{s}>,\\ &<\ \text{-}\ ,\ 4.83\,\text{s},\ 6.79\,\text{s}>,<\text{HO},\ 6.79\,\text{s},\ 8.52\,\text{s}>,\\ &<\text{WB},\ 8.52\,\text{s}, 13.04\,\text{s}>,<\text{HO},13.04\,\text{s},15.60\,\text{s}>,\\ &<\text{MA},15.50\,\text{s}, 17.38\,\text{s}>,<\text{HO},17.38\,\text{s},18.97\,\text{s}>,\\ &<\text{CL},18.97\,\text{s},20.22\,\text{s}>)\end{aligned} \tag{6.6}$$

that our data augmentation scheme is able to generate traces that look realistic and do not show any discontinuities.

6.4 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 6.3. 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.

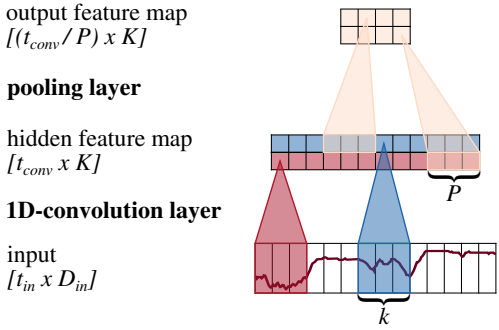
6.4.1 Neural Network architecture

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.



t_{in} ... *number of input time steps*
t_{conv} ... *number of time steps after convolution*
D_{in} ... *number of input dimensions*
K ... *number of kernels*
P ... *pooling size*
k ... *kernel size*

Figure 6.6: 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.

6.4.1.1 The Convolutional Neural Network

A Convolutional-Neural-Network (CNN) is one of the techniques used in time-series processing, for example, seismic data [Mey+19]. As stated by Goodfellow, Bengio and Courville [GBC16, Chapter 9], a CNN implements the convolution operation to extract feature maps from the input using kernels, defined in Equation (6.7), 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

$$y(t) = (x * k)(t) = \sum_{i=-\infty}^{\infty} (x(i) \cdot k(t-i)) \tag{6.7}$$

of the convolutional layer output. Roughly speaking, a CNN

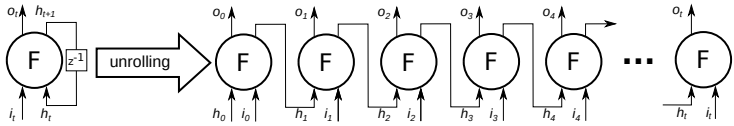


Figure 6.7: A Recurrent Neural Network (RNN) cell, in simple representation (left), and unrolled over time. i_t represents the input, o_t the output and h_t the internal state at time t.

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 6.6 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 6.6. 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 $[(t_{conv}/P) \times K]$.

6.4.1.2 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 6.7 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 6.7), 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.

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 [Gra12, 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 [HS97] 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 [Gra12, 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 [Gra12, 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.

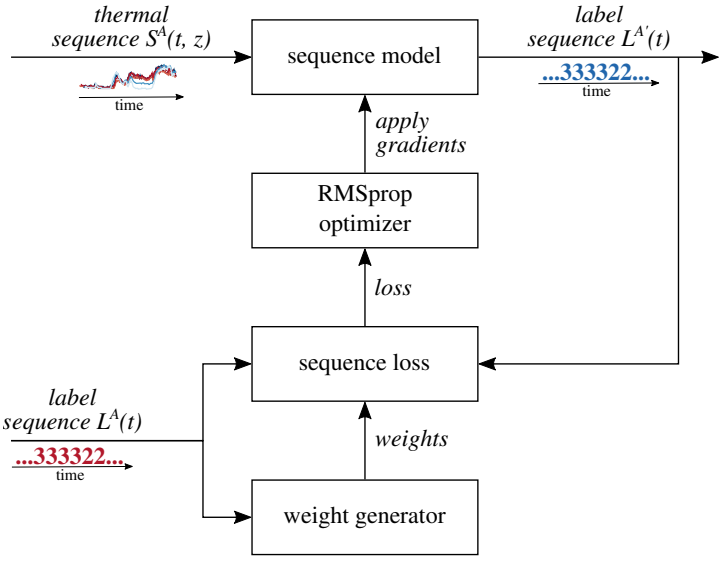


Figure 6.8: Setup during the training phase of the sequence model.

6.4.2 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 6.8 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 6.3.

During training, we use the *sequence loss* from the TensorFlow 2.0 Addons `seq2seq` package[2]. This loss function allows us to weight the individual samples of the trace for two purposes: (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 [TH12]. 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[3]. However, we acknowledge criticism towards stochastic optimisation techniques that has been raised in the past [RKK19], 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.

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

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

6.5 Sequence transformation and performance metrics

Following the information flow, as depicted in Figure 6.3, 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.

6.5.1 Sequence transformation

After training the sequence model, we feed an example thermal sequence to obtain the label trace, as illustrated in Figure 6.9. The example is generated using thermal profiles collected from a Sony Xperia Z5 smartphone, using the experimental setup outlined in section 6.6. 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 6.9 illustrates a thermal sequence using the applications outlined in Table 6.1 and thermal profiles from a Sony Xperia Z5 (see section 6.6 and section 6.7). (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 [Gra12]. CTC based models provide an output distribution over all possible application

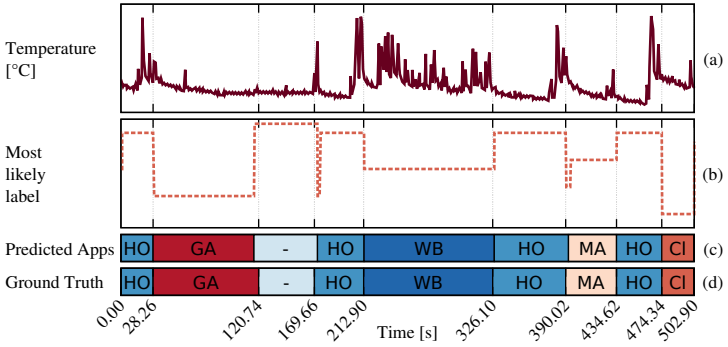


Figure 6.9: A thermal sequence processing example taken from the evaluation in section 6.7 using thermal profiles collected from a Sony Xperia Z5. shows (a) the thermal sequence $S^A(t, z)$ for one of the zones, (b) the labelling sequence $L^{A'}(t)$, (c) the predicted application sequence $\mathbf{A'}$ and (d) the actual application sequence $\mathbf{A}$. The 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.

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 an application has been executed multiple times in a row or once for a longer period of time.

In Figure 6.9, 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. 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 6.9 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 [Gra12] 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.

6.5.2 Performance metrics

In order to evaluate the performance of the whole approach as outlined in Figure 6.3, we need to determine 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 [SC07]. 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*.

6.6 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 ARMv7-S5 SM-900H based on a Samsung Exynos 5422 SoC, with Android 5.0 and 3 thermal zones; from now on referred to as ARMv7-S5.
- A Sony Xperia ARMv8-Z5 based on a Snapdragon 810 SoC with Android 7.0 and 36 thermal zones, referred to as ARMv8-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 chapter 2). 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

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

Table 6.1: Used applications and associated labels during the performance evaluation.

reads the corresponding sysfs files[4]. In addition, the sink application determines the current foreground application, i. e., the ground truth. The foreground application is determined by querying the usage stats or activity manager, depending on the Android build. 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 ARMv7-S5 and 46 ms on ARMv8-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

[4]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`

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 6.1 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.

6.6.1 Thermal parameters

For applying a thermal side channel attack according to Figure 6.3, we need to train the corresponding sequence model, see Figure 6.4. To be able to apply the data augmentation scheme as described in section 6.3, 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 ARMv8-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 ARMv7-S5, we exclude the battery sensor as it does not show any application-dependent thermal variations. All parameters are outlined in Figure 6.10.

To determine the thermal coefficients β_z, we place the target devices in a testbed [Sig20, 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

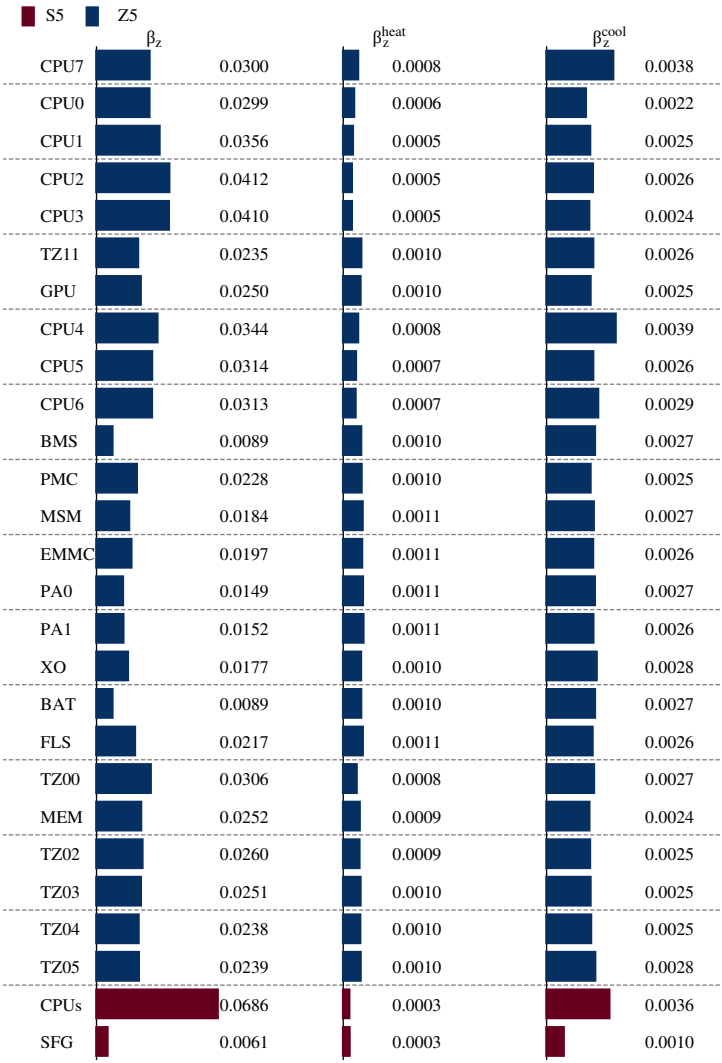


Figure 6.10: Thermal coefficients for internal (β_z), increasing (β_z^{heat}) and decreasing ambient temperatures.

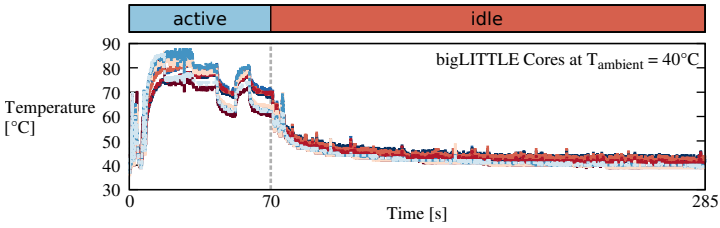


Figure 6.11: The device is heated up by an active application and we derive the thermal coefficients from the measurements taken when the device is idle and cools down.

then execute a CPU benchmark[5] 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 6.11.

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 β_z^{heat} and β_z^{cool}. The results in Figure 6.10 illustrate that the ambient thermal coefficients β_z^{heat} and β_z^{cool} are significantly lower than the respective β_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 6.10 show that the CPUs have the highest thermal dynamics on both smartphones ARMv7-S5 and ARMv8-Z5. Furthermore, while the internal thermal coefficients β_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 β_z^{heat} for different zones. Therefore, we will

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

only use the eight CPU thermal zone readings on ARMv7-S5 and the single CPU cluster reading on ARMv8-Z5 as input for the application detection.

6.6.2 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 6.4, we need user inputs to interact with the selected applications. We use the RepetiTouch Pro application[6] 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 6.1. 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.

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

6.7 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 6.6.2. 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.

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 6.6.2.

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[7] application and add it to the test data.

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

Instead of introducing new labels for applications not defined in Table 6.1, the unknown ("-") label is used.

6.7.1 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 6.12, and compare it to the performance metrics illustrated in Figure 6.13.

The per-time-step accuracy outlined in Figure 6.13 indicates that the models perform better with data from ARMv8-Z5 than ARMv7-S5, which is also supported by the trace illustrated in Figure 6.12. However, the relative Levenshtein distance is quite high for both platforms, as the experiments yield 0.83 for ARMv7-S5 and 0.70 for ARMv8-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 6.13 for the two platforms, ARMv7-S5 and ARMv8-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 6.12.

6.7.2 Effectiveness of the data augmentation scheme

The experimental results depicted in Figure 6.13 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 6.14 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 ARMv8-Z5, the performance for the reduced training data scenario decreases. For ARMv7-S5, however, training fails completely. We assume that these two effects are caused by the following factors:

- The performance for ARMv8-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 ARMv7-S5, training of the sequence model is not

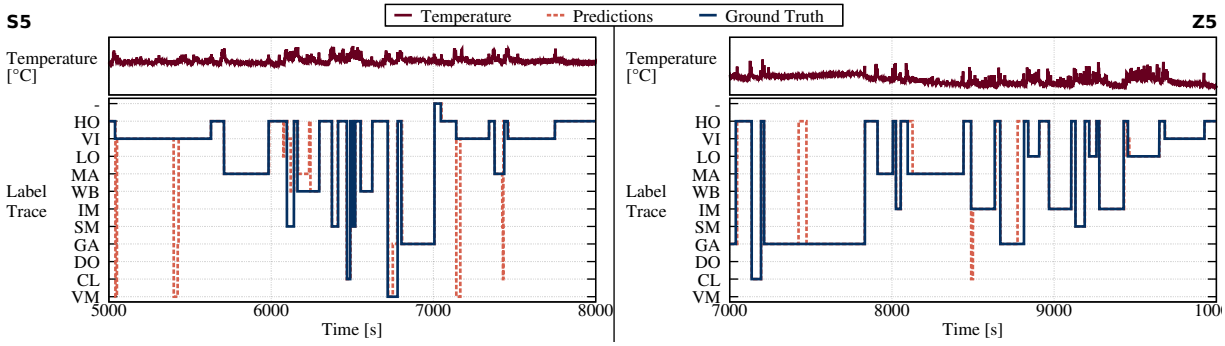


Figure 6.12: Traces indicate a higher amount of labelling errors on ARMv7-S5 than on ARMv8-Z5.

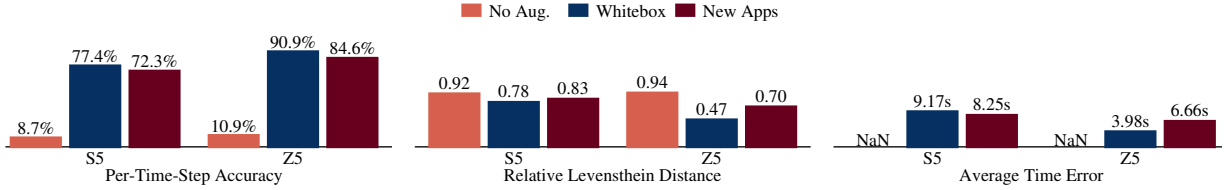


Figure 6.13: Training the model without augmented data is not possible. Models trained with augmented data perform well, but adding a new application cause performance degradation.

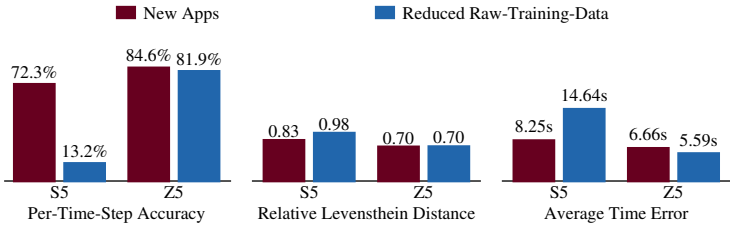


Figure 6.14: Reducing the base dataset of thermal profiles for the data augmentation causes a slight performance drop for ARMv8-Z5, but has a substantial impact on the performance for data from ARMv7-S5. This is caused by the dependency of our data augmentation scheme on the amount and quality of the thermal profiles available for data generation.

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.

6.7.3 Performance on different devices

The results illustrated in Figure 6.13 show that in general models trained for ARMv8-Z5 outperform the models for ARMv7-S5. While we mentioned in the previous subsection that the quality of the thermal profiles collected on ARMv7-S5 seem to be less representative, we also assume that the lower performance on ARMv7-S5 is caused by the lack of thermal information that we can collect. On ARMv8-Z5, we can use 8 sensor reading for 8 cores, while on ARMv7-S5 we can only get 1 reading for all 8 cores. Therefore, the amount of thermal information that we can extract is lower on ARMv7-S5, which lowers the performance of the sequence model.

6.7.4 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 6.13. 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.

6.7.5 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 6.15 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, 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,

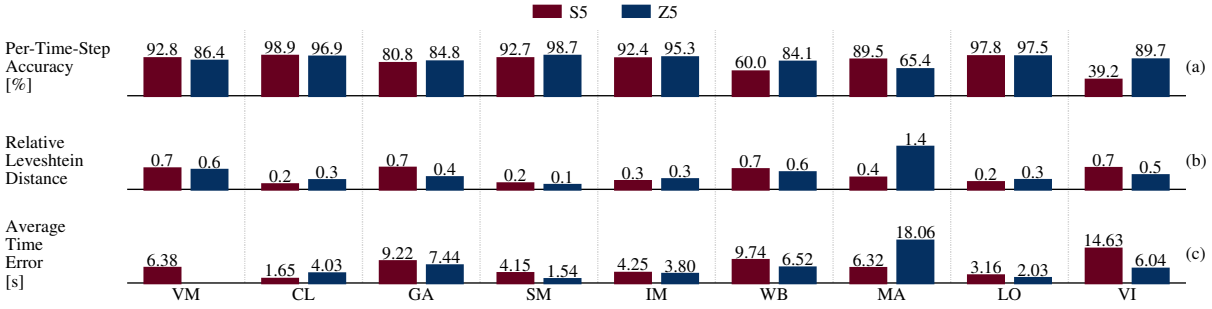


Figure 6.15: Performance metrics for the detection of a single application. While the per-time-step accuracy is very high for most applications, the relative Levenshtein distance varies between 0.1 to 0.6. This indicates that the applications with a high relative Levenshtein distance have a less unique 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.

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 6.7.4, 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.

6.7.6 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 ARMv7-S5, and (ii) 4.5h (16200 s) for ARMv8-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

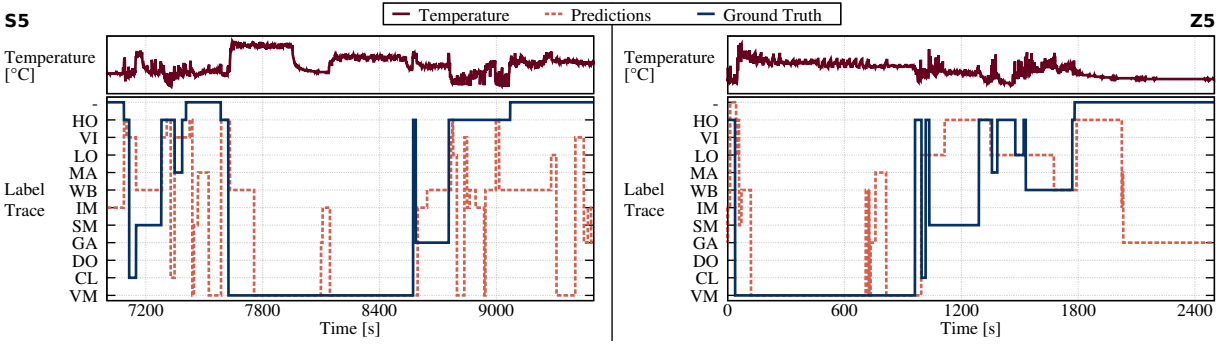


Figure 6.16: The real-world traces for both smartphones, ARMv7-S5 and ARMv8-Z5. The short temporal snippets validate the performance metrics and illustrate that the models are not able to correctly label the thermal trace.

example illustrated in Figure 6.16. Also, models for the binary use cases were not capable of detecting the respective target application in the real-world trace.

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 scheduling and core pinning variations into account, as all thermal profiles are collected from smartphones running the same set of background services.

6.8 Summary

In this chapter, 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.

7

Concluding remarks and outlook

The breakdown of the Dennard scaling, i. e., the effect that with decreasing size of transistors the power density stays constant, caused two developments: (i) the shift towards multicore architectures to further increase the computational power of processors, and (ii) the introduction of power management systems for a resourceful use of the available thermal budget. Here, the thermal budget defines the amount of thermal energy that can be dissipated without exceeding the physical limitations of the chip. In addition, the rise of battery-powered systems further fuelled the development of more sophisticated power management systems, increasing the battery lifetime while still offering a high quality of service to the user.

Due to the increased computational capabilities of our devices, they are now often shared between multiple users or application domains to increase the utilisation of the available resources. For example, cloud systems run multiple virtual

machines on the same physical server, or a single smartphone may be used for work and private applications. While programs from different user or application domains should be isolated and independent from each other, the behaviour of the power management system of the underlying hardware depends on the overall system utilisation and, therefore, establishes a link between those applications.

The possibility of influencing applications in different execution domains on the same hardware by manipulating the behaviour of the power management system has security implications. As Lampson [Lam73] stated, data can only be regarded as secure if it can be guaranteed that it cannot be transferred to a third party without the consent of the data owner. Therefore, the prospect of (mis-) using the power management system to send, or leak, data from one application or user domain to another poses a severe security threat.

In recent years, the focus of the research community on detecting and mitigating such data leaks has increased, but some challenges remain. For instance, due to the variability in the analysis methodology, comparability and reproducibility of the results were often not achieved. Furthermore, due to the complexity of data leaks and the lack of a suitable toolkit, the amount of engineering effort required to perform a data leak analysis remained high. In addition, many previously-presented analyses were solely based on empirical evaluation and did not provide a theoretical analysis.

As data leaks related to the power management system rely on the interplay of many different system aspects, mitigating them may not always be feasible without major changes to the system design. Therefore, we argue that it is important to assess the threat potential of data leaks to decide whether immediate actions need to be taken, or if it is sufficient for the security level of a device to monitor a potential data leak.

However, a confident assessment of the threat potential of data leaks requires an exhaustive analysis. Such an exhaustive analysis must consist of a theoretical analysis to determine the capabilities of the data leak and an extensive experimental study to provide evidence that a data leak could pose a threat under realistic conditions.

In this thesis, we addressed these open challenges and provided analyses of power management related data leaks in multicore systems. We illustrated how to perform exhaustive analyses of data leaks. Moreover, we provided theoretical analyses by establishing channel models and determining upper bounds on the channel capacity, i. e., the maximum amount of information that could be transferred through a data leak under ideal conditions. Furthermore, we presented extensive experimental studies of different data leaks to illustrate the feasibility of them being implemented and posing a security threat.

With our data leak analyses, we showed that the power management system has implications for the security framework of a system. We illustrated that the threat emerging from these power management related data leaks has to be taken into account when designing secure systems. In the remainder of this chapter, we give an overview of the contributions of each chapter and present possible future directions.

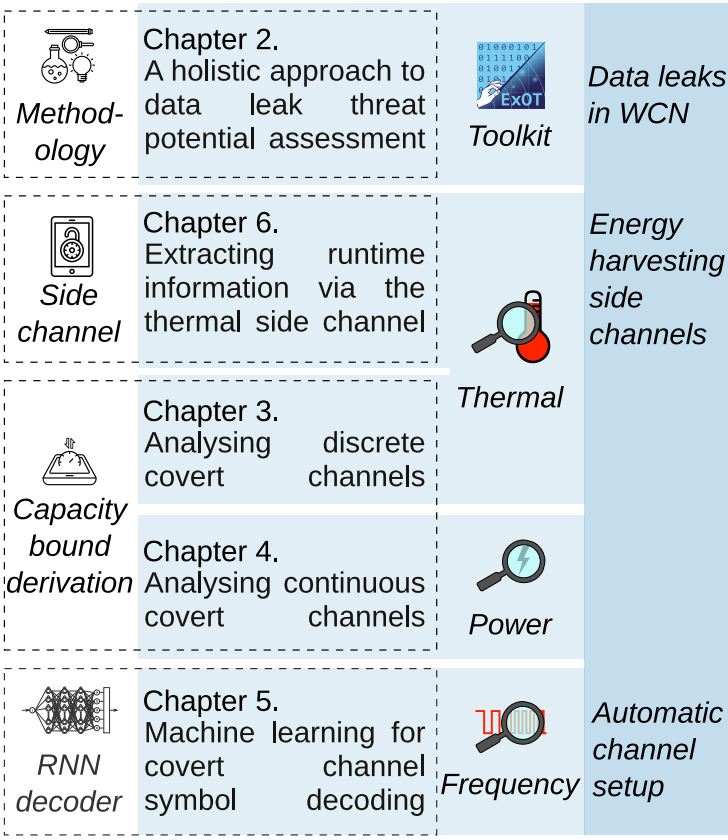


Figure 7.1: We presented a methodology for data leak threat potential evaluation and the supporting toolkit ExOT and how to apply it in chapter 2, 3 and 4. Based on these findings and using ExOT future research could evaluate data leaks in wireless sensor networks. Using the machine learning techniques applied in chapters 5 and 6, future research could develop a method for automatic channel setups and evaluate energy harvesting side channels.

7.1 Contributions

An overview of the chapters and the main contributions is illustrated in Figure 7.1. We discuss the contributions in detail in the remainder of this section.

7.1.1 Data leak evaluation methodology

In chapter 2, we presented a methodology to analyse data leaks and assess their threat potential. The methodology supports the reproducibility, comparability and expressiveness of the analysis results by relating general data leaks to cover channels and providing a four phase framework to execute an exhaustive covert channel analysis.

To analyse covert channels exhaustively, we first proposed to determine a channel model to establish a theoretical understanding of the data leak. Based on this model, we then derived a channel capacity bound, which allowed us to assess the threat potential, as the capacity bound is independent of implementation artefacts. In the third phase, we proposed to back the theoretical analysis with experimental data by executing a performance evaluation of a simple covert channel implementation in a controlled scenario. The fourth phase consisted of an implementation of the covert channel in a realistic environment to illustrate its threat outside the laboratory setup.

7.1.2 Experiment Orchestration Toolkit (ExOT)

We presented the Experiment Orchestration Toolkit (ExOT) in chapter 2, which implements the methodology and helps to reduce the engineering effort necessary for a data leak analysis.

ExOT is designed to be easy to extend and to reduce the effort needed to port experiments to a different setup.

Furthermore, ExOT simplifies the task of data generation, collection, analysis and sharing by providing python based data processing modules to simplify the handling of data. ExOT contains a C/C++ application library with Java extensions to enable fast development of experimental applications for a wide range of different devices. ExOT is published under the revised BSD 3-clause licence and freely available for download.

7.1.3 Methods for capacity bound derivation

We presented methods to derive capacity bounds for two different classes of covert channels – continuous and discrete.

In chapter 3, we illustrated how to derive the capacity of continuous covert channels by approximating the power spectral density function of the channel. By executing a series of experiments, it was possible to approximate the input power spectrum S_{xx} and the output power spectrum S_{yy} of the channel. Using these spectra, we showed that we could determine the channel power spectrum $S_{hh} = S_{yy}/S_{xx}$ and apply water-filling algorithms to determine an upper channel capacity bound.

In chapter 4, we presented a methodology to derive capacity bounds for discrete covert channels as well. To derive an upper bound on the channel capacity, a state model of the covert channel had to be established. Based on this model, it was possible to derive the transition matrix $\mathbf{A}$ and determine the channel capacity bound per channel use. In cases where experimental evaluation allowed us to determine the state transition time T, we showed that it was possible to determine an upper bound on the channel capacity.

7.1.4 Robust Recurrent Neural Network (RNN) based signal decoder design

Some covert channels experience high interference and are very dependent on the interplay of different components of a system, for instance, the frequency covert channel presented in chapter 5. Therefore, conventional signal decoding mechanisms may fail.

To tackle the issues of high signal interference and variability, we presented a signal decoder design based on a Recurrent Neural Network (RNN). In particular, we employed fully connected Long Short-Term Memory (LSTM) unit layers and a fully connected dense layer to implement the Connectionist Temporal Classification (CTC) sequence labelling algorithm [Gra12]. We showed that a decoder based on this design is able to decode data sent through the frequency covert channel reliably and can easily be ported to different platforms.

7.1.5 Thermal data augmentation scheme

Evaluating data leaks may require large amounts of data, and it may not be feasible to generate appropriate data using a measurement setup. Therefore, there is a need for data augmentation methods to provide appropriate data sets to evaluate data leaks.

In chapter 6, we presented such a data augmentation scheme for thermal data. The data augmentation scheme allowed us to generate large sets of thermal data to train complex machine learning models successfully. We have also presented the limitations of our data augmentation technique and outlined which improvements are necessary so that the augmented data resembles real-world data more closely.

7.1.6 Evaluation of power management related covert channels

As stated before, the power management system has security implications that have not yet been thoroughly studied. We presented exhaustive evaluations of three different power management related covert channels and assessed their threat potential:

- The *thermal covert channel*, a robust cross-core data leak established by encoding data into the thermal behaviour of a device, by deliberately heating it up or letting it cool down.
- The *power covert channel*, which allows high throughput cross-core communication between otherwise isolated applications, by modulating the power dissipation of the device.
- The *frequency covert channel*, a covert channel that is established by encoding information in the operating frequency of the device, which is hard to mitigate, as it relies only on timing measurements.

These covert channels illustrated that the power management system has to be taken into account when designing the security framework of a system.

7.1.7 Thermal side channel attack

In chapter 6, we presented the novel thermal side channel attack. This attack allows us to leak application runtime information in current smartphones. We showed that by employing a Convolutional-Neural-Network (CNN), a Recurrent Neural Network (RNN) and subsequent label processing, we were able to determine which application was running at what time on a

smartphone. The experimental evaluation was executed using real user interaction with two smartphones and data collected both from a controlled laboratory setup and from outside of the laboratory.

7.2 Possible future directions

The contributions of this thesis mark an important step towards an improved approach to data leak analysis, which yields comparable, reproducible and expressive results. Furthermore, in this thesis, we illustrated that the power management of current computing systems adds another attack surface, which must be considered when designing security frameworks. In this section, we highlight new research directions related to the security implications of power management systems in computing systems.

Automatic channel setup for discrete covert channels. The signal decoder design presented in chapter 5 already illustrated how to automate the receiver setup in a communication channel. This design could be extended to the whole communication chain for discrete covert channels. In particular, process mining techniques might be able to determine the state model of discrete covert channels by providing detailed characteristics of the channel. Based on this model, sender and receiver pairs based on adaptive techniques, like reinforcement learning or genetic algorithms, may be used to automate the setup of the covert channel fully and relieve researchers from the engineering-heavy task of covert channel implementations. An important aspect of this research is also the attack footprint. As many machine learning or adaptive algorithms are quite resource-intense, techniques must be developed to enable the application of such an automated channel setup under highly

constraint conditions.

Data leaks in wireless communication networks. Wireless communication protocols are a potential key technology in future Internet-of-Things (IoT) applications, for example, in the automation of industry production. Therefore, wireless communication protocols should receive more attention from a security perspective to identify possible attack vectors. For example, investigating the ability to detect and identify wireless communication protocols used in industrial IoT applications and further decode their packets with Commercial Off-the-Shelf (COTS) hardware is important for understanding the potential attacks and design countermeasures for wireless IoT networks. This is crucial, as wireless communication protocols commonly used in industrial IoT applications exhibit distinct temporal and spectral characteristics. COTS wireless radios and transceivers are capable of recognising and decoding packets from a different protocol in the same frequency band. A focus could be put on the sub-1 GHz band technologies, like Long Range Wide Area Network (LoRaWAN), which are extensively used in industrial context due to their energy efficiency, long battery life and extended range for remote sensors. Possible research questions are:

- Is it possible to build a device using COTS hardware with energy and form factor constraints?
- How can we detect, localise and identify wireless protocols?
- How can we decode packets of the identified protocols?
- How much information about the network (e. g., topology, structure, number of nodes, etc.) can be inferred if the set of observable nodes is limited?

- How can we harden defences of protocols against sniffing and attacks?

Possible challenges are, for instance, the fact that the presence of multiple devices operating heterogeneous protocols and dynamically changing their protocol parameters makes it hard to detect, localise and identify communication protocols. Furthermore, the temporal and spectral properties of protocols in the sub-1GHz-band might have an impact on the capability to gather data and identify a protocol based on that data.

Energy harvesting side channels. Another emerging topic in the IoT era is energy harvesting systems, i. e., systems that collect the energy necessary for their operation from their environment. Similar to the power management system in server or mobile systems, the energy harvesting strategies applied in IoT systems might offer a previously undiscovered attack surface. To analyse whether there are attack vectors for energy harvesting systems, a model of the interaction of the system sensors and actuators with the environment is necessary. In addition, it is necessary to establish threat mitigation techniques to minimise the attack surface. However, due to the nature of energy harvesting systems, mitigation strategies are restricted by resource constraints. Furthermore, it is important to distinguish between actual attacks and ageing effects of the system or changing environmental conditions. This research may lead to a formal verification and testing framework to ensure that energy harvesting systems conform to their security requirements. Such a verification must go beyond a single-use case demonstration and provide guarantees on the system security level, which still need to be defined.

7.3 Availability

All the data presented in this thesis, and the tools used to process it are available online.

Experiment Orchestration Toolkit (ExOT): [MKT20b]
Detailed information on Experiment Orchestration Toolkit (ExOT) can be found on the website `exot.ethz.ch`, as well as links to download it. Below, on the left, the Experiment Orchestration Toolkit (ExOT) logo illustrated, and on the right the QR-Code linking to the ExOT website.

To process the data of this thesis, download version 1.1.0 of Experiment Orchestration Toolkit (ExOT) either manually or by using the script below:

```
${GITURL}=https://gitlab.ethz.ch/tec/public/exot
for repo in eengine app_unx app_apk compilation; do
  git clone ${GITURL}/${repo}.git
  cd ${repo}
  git checkout pub_Mie20
  git submodule -init -recursive update
  cd ..
done
```

Data: [Mie20]
The data is available in multiple repositories in the ETH Research Collection, which are listed bellow.

Chapter 2:
`http://www.doi.org/10.3929/ethz-b-000418146`

http://www.doi.org/10.3929/ethz-b-000418157
http://www.doi.org/10.3929/ethz-b-000418163
http://www.doi.org/10.3929/ethz-b-000418166

Chapter 3:
http://www.doi.org/10.3929/ethz-b-000418168
http://www.doi.org/10.3929/ethz-b-000418171
http://www.doi.org/10.3929/ethz-b-000418173

Chapter 4:
http://www.doi.org/10.3929/ethz-b-000418174

Chapter 5:
http://www.doi.org/10.3929/ethz-b-000418180

Chapter 6:
http://www.doi.org/10.3929/ethz-b-000418183
http://www.doi.org/10.3929/ethz-b-000418184

The data needs to be extracted to the `datapro/data` directory of ExOT, for example using following commands:

```
for tar in $(ls -d ${DOWNLOAD_DIR}/*.tar.part00); do
  cat ${tar/00/\*} >
  ↪  ${EXOT_PATH}/eengine/data/tmp_${tar/\.part00/}
  tar -xf ${EXOT_PATH}/eengine/data/tmp_${tar/\.part00/}
  ↪  -C ${EXOT_PATH}/eengine/data/
  rm ${EXOT_PATH}/eengine/data/tmp_${tar/\.part00/}
done
```

Here, `DOWNLOAD_DIR` is the path to the directory which contains the downloaded zip files and `EXOT_PATH` is the path to the root directory in which the checkouts of all ExOT repositories are located. Experiments may contain a `README.md` with further instructions.

Bibliography

[20012] JCGM 200. *International vocabulary of metrology–Basic and general concepts and associated terms*. 2012. URL: https://www.bipm.org/utils/common/documents/jcgm/JCGM_200_2012.pdf.

[ADY13] Yousra Aafer, Wenliang Du and Heng Yin. 'Droidapiminer: Mining api-level features for robust malware detection in android'. In: *International Conference on Security and Privacy in Communication Systems*. Springer. 2013, pp. 86–103. DOI: 10.1007/978-3-319-04283-1_6. URL: https://doi.org/10.1007/978-3-319-04283-1_6.

[AEW09] Andrea Arcangeli, Izik Eidus and Chris Wright. 'Increasing memory density by using KSM'. In: *Proceedings of the linux symposium*. Citeseer. 2009, pp. 19–28. URL: https://www.kernel.org/doc/ols/2009/ols2009-pages-19-28.pdf.

[Ana+13] Ittai Anati et al. 'Innovative technology for CPU based attestation and sealing'. In: *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*. Vol. 13. 2013. URL: https://pdfs.semanticscholar.org/708a/3c03556b5bc20b5bd8e58ef2f47f6a9fc7d2.pdf.

[Arm19] Arm Limited. *ARM Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*. Tech. rep. Apr.

2019. URL: https://static.docs.arm.com/ddi0487/db/DDI0487D_b_armv8_arm.pdf.

[AST89] G. Andria, M. Savino and Amerigo Trotta. 'Windows and interpolation algorithms to improve electrical measurement accuracy'. In: *IEEE Transactions on Instrumentation and Measurement* 38.4 (Aug. 1989), pp. 856–863. DOI: 10.1109/19.31004. URL: https://doi.org/10.1109/19.31004.

[Bar15] Davide B. Bartolini. 'Techniques and Tools for Efficient, QoS-Driven Warehouse-Scale Computing'. Tesi di Dottorato (PhD Thesis). Politecnico di Milano, 2015. URL: https://www.politesi.polimi.it/handle/10589/100464.

[Bar16] Elaine Barker. 'Recommendation for key management Part 1: General (Revision 4)'. In: *NIST special publication* 800.57 (2016), pp. 1–147. URL: https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-4/final.

[Bar18] Carsten Barth. *Come again? Towards repeatable security experiments*. Tech. rep. 1. Semester Thesis; Supervisors: Philipp Miedl and Lothar Thiele. Gloriastrasse 35, 8092 Zürich, Switzerland: ETH Zürich, Aug. 2018.

[BG16] Anna L Buczak and Erhan Guven. 'A survey of data mining and machine learning methods for cyber security intrusion detection'. In: *IEEE Communications Surveys & Tutorials* 18.2 (2016), pp. 1153–1176. DOI: 10.1109/COMST.2015.2494502. URL: https://doi.org/10.1109/COMST.2015.2494502.

[BM01] David Brooks and Margaret Martonosi. 'Dynamic Thermal Management for High-Performance Microprocessors'. In: *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*. HPCA '01. USA: IEEE Computer Society, 2001, p. 171. ISBN: 0-76951-019-1. DOI: 10.5555/580550.876439. URL: http://csdl.computer.org/comp/proceedings/hpca/2001/1019/00/10190171abs.htm.

[BMT16] 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. London, United Kingdom: ACM, 2016, 24:1–24:16. ISBN: 978-1-45034-240-7. DOI: 10.1145/2901318.2901322. URL: http://doi.acm.org/10.1145/2901318.2901322.

[Bro+09a] J. Brouchier et al. 'Temperature Attacks'. In: *IEEE Security and Privacy* 7.2 (Mar. 2009), pp. 79–82. ISSN: 1540-7993. DOI: 10.1109/MSP.2009.54. URL: https://doi.org/10.1109/MSP.2009.54.

[Bro+09b] Julien Brouchier et al. *Thermocommunication*. Cryptology ePrint Archive, Report 2009/002. 2009. URL: https://eprint.iacr.org/2009/002.

[Can+12] Davide Canali et al. 'A Quantitative Study of Accuracy in System Call-Based Malware Detection'. In: *Proceedings of the 2012 International Symposium on Software Testing and Analysis*. ISSTA 2012. Minneapolis, MN, USA: Association for Computing Machinery, 2012, pp. 122–132. ISBN: 978-1-45031-454-1. DOI: 10.1145/2338965.2336768. URL: https://doi.org/10.1145/2338965.2336768.

[CLW69] James W. Cooley, Peter A. W. Lewis and Peter D. Welch. 'The Fast Fourier Transform and Its Applications'. In: *IEEE Trans. on Educ.* 12.1 (Mar. 1969), pp. 27–34. ISSN: 0018-9359. DOI: 10.1109/TE.1969.4320436. URL: https://doi.org/10.1109/TE.1969.4320436.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006. ISBN: 978-0-471-24195-9.

[CV14] Jie Chen and Guru Venkataramani. 'CC-Hunter: Uncovering Covert Timing Channels on Shared Processor Hardware'. In: *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-47. Cambridge, United Kingdom: IEEE Computer

Society, 2014, pp. 216–228. ISBN: 978-1-47996-998-2. DOI: 10.1109/MICRO.2014.42. URL: https://doi.org/10.1109/MICRO.2014.42.

[Den+74] Robert H Dennard et al. 'Design of ion-implanted MOSFET's with very small physical dimensions'. In: *IEEE Journal of Solid-State Circuits* 9.5 (1974), pp. 256–268. DOI: 10.1109/JSSC.1974.1050511. URL: https://doi.org/10.1109/JSSC.1974.1050511.

[DS05] P Dadvar and K Skadron. 'Potential thermal security risks'. In: *Semiconductor Thermal Measurement and Management Symposium, 2005 IEEE Twenty First Annual IEEE*. 2005, pp. 229–234. DOI: 10.1109/STHERM.2005.1412184. URL: https://doi.org/10.1109/STHERM.2005.1412184.

[EPA15] Dmitry Evtyushkin, Dmitry Ponomarev and Nael Abu-Ghazaleh. 'Covert Channels through Branch Predictors: A Feasibility Study'. In: *Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '15. Portland, Oregon: Association for Computing Machinery, 2015. ISBN: 978-1-45033-483-9. DOI: 10.1145/2768566.2768571. URL: https://doi.org/10.1145/2768566.2768571.

[EPA16] Dmitry Evtyushkin, Dmitry Ponomarev and Nael Abu-Ghazaleh. 'Understanding and Mitigating Covert Channels Through Branch Predictors'. In: *ACM Transactions on Architecture and Code Optimization (TACO)* 13.1 (Mar. 2016). ISSN: 1544-3566. DOI: 10.1145/2870636. URL: https://doi.org/10.1145/2870636.

[Esm+11] Hadi Esmaeilzadeh et al. 'Dark Silicon and the End of Multicore Scaling'. In: *SIGARCH Comput. Archit. News* 39.3 (June 2011), pp. 365–376. ISSN: 0163-5964. DOI: 10.1145/2024723.2000108. URL: https://doi.org/10.1145/2024723.2000108.

[Fit19] Athanasios Fitsios. *Towards Task Inference on Mobile Systems based on Thermal Traces*. Tech. rep. 1. Semester Thesis; Supervisors: Philipp Miedl, Rehan Ahmed

and Lothar Thiele. Gloriastrasse 35, 8092 Zürich, Switzerland: ETH Zürich, Mar. 2019.

[FPK17] Apostolos P Fournaris, Lidia Pocero Fraile and Odysseas Koufopavlou. 'Exploiting Hardware Vulnerabilities to Attack Embedded System Devices: a Survey of Potent Microarchitectural Attacks'. In: *Electronics* 6.3 (2017), p. 52. DOI: 10.3390/electronics6030052. URL: https://doi.org/10.3390/electronics6030052.

[GBC16] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[Göt+17] Johannes Götzfried et al. 'Cache Attacks on Intel SGX'. In: *Proceedings of the 10th European Workshop on Systems Security*. EuroSec'17. Belgrade, Serbia: Association for Computing Machinery, 2017. ISBN: 978-1-45034-935-2. DOI: 10.1145/3065913.3065915. URL: https://doi.org/10.1145/3065913.3065915.

[Gra12] Alex Graves. 'Supervised sequence labelling'. In: *Supervised Sequence Labelling with Recurrent Neural Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 5–13. ISBN: 978-3-64224-797-2. DOI: 10.1007/978-3-642-24797-2_2. URL: https://doi.org/10.1007/978-3-642-24797-2_2.

[Gru+16a] Daniel Gruss et al. 'Flush+Flush: A Fast and Stealthy Cache Attack'. In: *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721*. DIMVA 2016. San Sebastián, Spain: Springer-Verlag, 2016, pp. 279–299. ISBN: 978-3-31940-666-4. DOI: 10.1007/978-3-319-40667-1_14. URL: https://doi.org/10.1007/978-3-319-40667-1_14.

[Gru+16b] Daniel Gruss et al. 'Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR'. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Vienna, Austria: Association for Computing Machinery, 2016, pp. 368–379.

ISBN: 978-1-45034-139-4. DOI: 10.1145/2976749.2978356. URL: https://doi.org/10.1145/2976749.2978356.

[Gur+15] Mordechai Guri et al. 'BitWhisper: Covert Signaling Channel between Air-Gapped Computers Using Thermal Manipulations'. In: *Proceedings of the 2015 IEEE 28th Computer Security Foundations Symposium*. CSF '15. USA, 2015, pp. 276–289. ISBN: 978-1-46737-538-2. DOI: 10.1109/CSF.2015.26. URL: https://doi.org/10.1109/CSF.2015.26.

[Has+05] Jahangir Hasan et al. 'Heat Stroke: Power-Density-Based Denial of Service in SMT'. In: *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*. HPCA '05. IEEE Computer Society, 2005, pp. 166–177. ISBN: 0-76952-275-0. DOI: 10.1109/HPCA.2005.16. URL: https://doi.org/10.1109/HPCA.2005.16.

[He17] Xiaoxi He. 'A Smart Attack using the Frequency Covert Channel'. Supervisors: Philipp Miedl, Matthias Meyer and Lothar Thiele. MA thesis. Gloriastrasse 35, 8092 Zürich, Switzerland: ETH Zürich, Oct. 2017.

[Hel+04] Joseph L. Hellerstein et al. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004. ISBN: 978-0-47166-881-7.

[HO92] C. Heegard and L. Ozarow. 'Bounding the capacity of saturation recording: the Lorentz model and applications'. In: *Selected Areas in Communications, IEEE Journal on* 10.1 (Jan. 1992), pp. 145–156. ISSN: 0733-8716. DOI: 10.1109/49.124474.

[Hoe+13] Matthew Hoekstra et al. 'Using Innovative Instructions to Create Trustworthy Software Solutions'. In: *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '13. Tel-Aviv, Israel: Association for Computing Machinery, 2013. ISBN: 978-1-45032-118-1. DOI: 10.1145/2487726.2488370. URL: https://doi.org/10.1145/2487726.2488370.

[HS14] Michael Hutter and Jörn-Marc Schmidt. 'The Temperature Side Channel and Heating Fault Attacks'. In: (2014). Ed. by Aurélien Francillon and Pankaj Rohatgi, pp. 219–235. DOI: 10.1007/978-3-319-08302-5_15. URL: http://eprint.iacr.org/2014/190.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. 'Long Short-Term Memory'. In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[Hun+15] C. Hunger et al. 'Understanding contention-based channels and using them for defense'. In: *Proceedings of the 21st IEEE International Symposium on High Performance Computer Architecture*. 2015, pp. 639–650. DOI: 10.1109/HPCA.2015.7056069. URL: https://doi.org/10.1109/HPCA.2015.7056069.

[INK11] 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*. June 2011, pp. 1–7. DOI: 10.1109/ReCoSoC.2011.5981510. URL: https://doi.org/10.1109/ReCoSoC.2011.5981510.

[Int15] Intel Corporation. *Intel 64 and IA-32 architectures software developer's manuals volume 3: System programming guide*. 2015. URL: http://cse.iitkgp.ac.in/~goutam/compiler/readingMaterial/intelXeon/253665.pdf.

[Int16] Intel. 'Intel® 64 and IA-32 Architectures Software Developer's Manual'. In: *Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B and 3D* (2016). URL: https://software.intel.com/en-us/articles/intel-sdm.

[Int18] Intel Corporation. *Intel® 64 and IA-32 Architectures Developer's Manual*. Architectures Software Developer's Manual: Intel® 64 and IA-32 includes supporting processors programming environment and architecture. May 2018. URL: https://www.intel.com/content/

www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-manual-325462.html.

[IRW17] 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. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1079–1094. ISBN: 978-1-45034-946-8. DOI: 10.1145/3133956.3133994. URL: https://doi.org/10.1145/3133956.3133994.

[Klo18] Bruno Klopott. *You also want to explore other security leaks? Building an easily extendable application library for security leak research*. Tech. rep. 1. Semester Thesis; Supervisors: Philipp Miedl and Lothar Thiele. Gloriastrasse 35, 8092 Zürich, Switzerland: ETH Zürich, Aug. 2018.

[Klo19] Bruno Klopott. 'How bad are data leaks really?' Supervisors: Philipp Miedl and Lothar Thiele. MA thesis. Gloriastrasse 35, 8092 Zürich, Switzerland: ETH Zürich, June 2019.

[Koc+18] Paul Kocher et al. 'Spectre attacks: Exploiting speculative execution'. In: *arXiv preprint arXiv:1801.01203* (2018). URL: https://spectreattack.com/.

[KRT16] Mikhail Kazdagli, Vijay Janapa Reddi and Mohit Tiwari. 'Quantifying and Improving the Efficiency of Hardware-Based Mobile Malware Detectors'. In: *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-49. Taipei, Taiwan: IEEE Press, 2016. DOI: 10.5555/3195638.3195683. URL: https://doi.org/10.5555/3195638.3195683.

[Lam73] Butler W. Lampson. 'A Note on the Confinement Problem'. In: *Commun. ACM* 16.10 (Oct. 1973), pp. 613–615. ISSN: 0001-0782. DOI: 10.1145/362375.362389. URL: https://doi.org/10.1145/362375.362389.

[LDC02] Chee How Lim, W Robert Daasch and George Cai. 'A thermal-aware superscalar microprocessor'. In: *Quality Electronic Design, 2002. Proceedings. International Symposium on*. IEEE. 2002, pp. 517–522. DOI: 10.1109/ISQED.2002.996797. URL: https://doi.org/10.1109/ISQED.2002.996797.

[Lip+16] Moritz Lipp et al. 'ARMageddon: Cache Attacks on Mobile Devices'. In: SEC'16 (2016), pp. 549–564. DOI: 10.5555/3241094.3241138. URL: https://doi.org/10.5555/3241094.3241138.

[Lip+18] Moritz Lipp et al. 'Meltdown'. In: *arXiv preprint arXiv:1801.01207* (2018). URL: https://spectreattack.com/.

[Mac03] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003. URL: http://www.inference.org.uk/mackay/itila/book.html.

[Mar+12] Claudio Marforio et al. 'Analysis of the Communication between Colluding Applications on Modern Smartphones'. In: *Proceedings of the 28th Annual Computer Security Applications Conference*. ACSAC '12. Orlando, Florida, USA: Association for Computing Machinery, 2012, pp. 51–60. ISBN: 978-1-45031-312-4. DOI: 10.1145/2420950.2420958. URL: https://doi.org/10.1145/2420950.2420958.

[Mas+15] Ramya Jayaram Masti et al. 'Thermal Covert Channels on Multi-core Platforms'. In: *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 865–880. ISBN: 978-1-93197-123-2. DOI: 10.5555/2831143.2831198. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/masti.

[MAT20] Philipp Miedl, Rehan Ahmed and Lothar Thiele. 'We know what you're doing! Application detection using thermal data'. In: *Leibniz Transactions on Embedded*

Systems Special Issue on Embedded System Security.1 (2020). Under review.

[Mau+17] Clémentine Maurice et al. 'Hello from the other side: SSH over robust cache covert channels in the cloud'. In: *NDSS, San Diego, CA, US* (2017). URL: https://cmaurice.fr/pdf/ndss17_maurice.pdf.

[Mei18] Manuel Meier. *Feature Extraction from Thermal Traces for the Thermal Fingerprinting Attack*. Tech. rep. 1. Semester Thesis; Supervisors: Philipp Miedl, Rehan Ahmed and Lothar Thiele. Gloriastrasse 35, 8092 Zürich, Switzerland: ETH Zürich, May 2018.

[Mey+19] Matthias Meyer et al. 'Systematic identification of external influences in multi-year microseismic recordings using convolutional neural networks'. In: *Earth Surface Dynamics* 7.1 (2019), pp. 171–190. DOI: 10.5194/esurf-7-171-2019. URL: https://www.earth-surf-dynam.net/7/171/2019/.

[Mic+15] Yan Michalevsky et al. 'PowerSpy: Location Tracking Using Mobile Device Power Analysis'. In: (Aug. 2015), pp. 785–800. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/michalevsky.

[Mie+18] Philipp Miedl et al. 'Frequency Scaling as a Security Threat on Multicore Systems'. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (Nov. 2018), pp. 2497–2508. ISSN: 1937-4151. DOI: 10.1109/TCAD.2018.2857038. URL: https://doi.org/10.1109/TCAD.2018.2857038.

[Mie20] Philipp Miedl. *Data: Threat potential assessment of power management related data leaks*. June 2020. DOI: 10.3929/ethz-b-XXXXXXXXX. URL: http://hdl.handle.net/XX.XXX.XXXXX/XXXXXX.

[Mil18] Max Millen. 'Analysis and Optimization of Frequency Governors'. Supervisors: Rehan Ahmed, Philipp Miedl and Lothar Thiele. MA thesis. Gloriastrasse 35, 8092 Zürich, Switzerland: ETH Zürich, Apr. 2018.

[MK94] Ira S Moskowitz and Myong H Kang. 'Covert channels-here to stay?' In: *Computer Assurance, 1994. COMPASS'94 Safety, Reliability, Fault Tolerance, Concurrency and Real Time, Security. Proceedings of the Ninth Annual Conference on*. IEEE. 1994, pp. 235–243. DOI: 10.1109/CMPASS.1994.318449. URL: https://doi.org/10.1109/CMPASS.1994.318449.

[MKT20a] Philipp Miedl, Bruno Klopott and Lothar Thiele. *Data: Thermal and cache covert channel analysis with ExOT*. Mar. 2020. DOI: 10.3929/ethz-b-000378872. URL: http://hdl.handle.net/20.500.11850/378872.

[MKT20b] Philipp Miedl, Bruno Klopott and Lothar Thiele. *ExOT Website*. Mar. 2020. URL: https://www.exot.ethz.ch/.

[MKT20c] 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. URL: https://doi.org/10.3929/ethz-b-000377986.

[MM07] 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.

[Moo98] G. E. Moore. 'Cramming More Components Onto Integrated Circuits'. In: *Proceedings of the IEEE* 86.1 (Jan. 1998), pp. 82–85. ISSN: 1558-2256. DOI: 10.1109/JPROC.1998.658762. URL: https://doi.org/10.1109/JPROC.1998.658762.

[MP14] Aleksandra Mileva and Boris Panajotov. 'Covert channels in TCP/IP protocol stack - extended version-'. In: *Central European Journal of Computer Science* 4.2 (2014), pp. 45–66. ISSN: 1896-1533. DOI: 10.2478/s13537-014-0205-6. URL: https://doi.org/10.2478/s13537-014-0205-6.

[MT18] 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. Pau, France: Association for Computing Machinery, 2018, pp. 1585–1592. ISBN: 978-1-45035-191-1. DOI: 10.1145/3167132.3167301. URL: https://doi.org/10.1145/3167132.3167301.

[MT20] Philipp Miedl and Lothar Thiele. *Capacity calculations in "Increased reproducibility and comparability of data leak evaluations using ExOT"*. Mar. 2020. DOI: 10.3929/ethz-b-000378017. URL: http://hdl.handle.net/20.500.11850/378017.

[Mur06] 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. Alexandria, Virginia, USA: Association for Computing Machinery, 2006, pp. 27–36. ISBN: 1-59593-518-5. DOI: 10.1145/1180405.1180410. URL: https://doi.org/10.1145/1180405.1180410.

[Ozs+15] Meltem Ozsoy et al. 'Malware-aware processors: A framework for efficient online malware detection'. In: *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE. 2015, pp. 651–661. DOI: 10.1109/HPCA.2015.7056070. URL: https://doi.org/10.1109/HPCA.2015.7056070.

[PLB07] Venkatesh Pallipadi, Shaohua Li and Adam Belay. 'cpuidle: Do nothing, efficiently'. In: *Proceedings of the Linux Symposium*. Vol. 2. Citeseer. 2007, pp. 119–125. URL: https://www.kernel.org/doc/ols/2007/ols2007v2-pages-119-126.pdf.

[PS06] Venkatesh Pallipadi and Alexey Starikovskiy. 'The ondemand governor'. In: *Proceedings of the Linux Symposium*. Vol. 2. 00216. sn. 2006, pp. 215–230. URL: ftp://157.158.0.32/pub/linux/kernel/people/lenb/acpi/doc/OLS2006-ondemand-presentation.pdf.

[PZ13] 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. USA: IEEE Computer Society, 2013, pp. 300–305. ISBN: 978-1-47992-971-9. DOI: 10.1109/ICTAI.2013.53. URL: https://doi.org/10.1109/ICTAI.2013.53.

[PZ17] Danny Philippe-Jankovic and Tanveer A Zia. 'Breaking VM Isolation-An In-Depth Look into the Cross VM Flush Reload Cache Timing Attack'. In: *International Journal of Computer Science and Network Security (IJCSNS)* 17.2 (2017), p. 181. ISSN: 1738-7906. URL: https://researchoutput.csu.edu.au/en/publications/breaking-vm-isolation-an-in-depth-look-into-the-cross-flush-reloa-2.

[Rai+12] Devendra Rai et al. 'Power Agnostic Technique for Efficient Temperature Estimation of Multicore Embedded Systems'. In: *Proceedings of the 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*. CASES '12. Tampere, Finland: Association for Computing Machinery, 2012, pp. 61–70. ISBN: 978-1-45031-424-4. DOI: 10.1145/2380403.2380421. URL: https://doi.org/10.1145/2380403.2380421.

[RBP09] Charles Reis, Adam Barth and Carlos Pizano. 'Browser Security: Lessons from Google Chrome'. In: *ACM Queue* 7.5 (2009), p. 3. DOI: 10.1145/1551644.1556050. URL: https://dl.acm.org/doi/pdf/10.1145/1551644.1556050.

[Ris+09] Thomas Ristenpart et al. '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. Chicago, Illinois, USA: Association for Computing Machinery, 2009, pp. 199–212. ISBN: 978-1-60558-894-0. DOI: 10.1145/1653662.1653687. URL: https://doi.org/10.1145/1653662.1653687.

[RKK19] Sashank J. Reddi, Satyen Kale and Sanjiv Kumar. *On the Convergence of Adam and Beyond*. 2019. arXiv: 1904.09237 [cs.LG]. URL: https://arxiv.org/abs/1904.09237.

[Ron+15] Hong Rong et al. '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. USA: IEEE Computer Society, 2015, pp. 68–75. ISBN: 978-1-46737-183-4. DOI: 10.1109/BDCloud.2015.12. URL: https://doi.org/10.1109/BDCloud.2015.12.

[SAS02] Kevin Skadron, Tarek Abdelzaher and Mircea R. Stan. 'Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management'. In: *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*. HPCA '02. USA: IEEE Computer Society, 2002, p. 17. DOI: 10.5555/874076.876476.

[SC07] Stan Salvador and Philip Chan. 'Toward Accurate Dynamic Time Warping in Linear Time and Space'. In: *Intell. Data Anal.* 11.5 (Oct. 2007), pp. 561–580. ISSN: 1088-467X. DOI: 10.5555/1367985.1367993. URL: https://dl.acm.org/doi/10.5555/1367985.1367993.

[Sel16] Mirko Selber. *UnCovert: Operating Frequency, a Security Leak?* Tech. rep. 1. Semester Thesis; Supervisors: Philipp Miedl and Lothar Thiele. Gloriastrasse 35, 8092 Zürich, Switzerland: ETH Zürich, Feb. 2016.

[Sel17] Mirko Selber. 'UnCovert3: Covert Channel Attacks on Commerical Multicore Systems'. Supervisors: Philipp Miedl and Lothar Thiele. MA thesis. Gloriastrasse 35, 8092 Zürich, Switzerland: ETH Zürich, Apr. 2017.

[Sha+15] Ali Shafiee et al. 'Avoiding Information Leakage in the Memory Controller with Fixed Service Policies'. In: *Proceedings of the 48th International Symposium on Microarchitecture*. MICRO-48. Waikiki, Hawaii: Association for Computing Machinery, 2015, pp. 89–101. ISBN:

978-1-45034-034-2. DOI: 10.1145/2830772.2830795. URL: https://doi.org/10.1145/2830772.2830795.

[Sha01] C. E. Shannon. 'A Mathematical Theory of Communication'. In: *SIGMOBILE Mob. Comput. Commun. Rev.* 5.1 (Jan. 2001), pp. 3–55. ISSN: 1559-1662. DOI: 10.1145/584091.584093. URL: https://doi.org/10.1145/584091.584093.

[Sig20] Lukas Sigrist. 'Design and Instrumentation of Environment-Powered Systems'. PhD thesis. ETH Zurich, 2020.

[Spo+17] Riccardo Spolaor et al. 'No Free Charge Theorem: a Covert Channel via USB Charging Cable on Mobile Devices'. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2017, pp. 83–102. DOI: 10.1007/978-3-319-61204-1_5. URL: https://doi.org/10.1007/978-3-319-61204-1_5.

[Str17] Raphael Strebel. *What is my Thermal Fingerprint?* Tech. rep. 1. Semester Thesis; Supervisors: Philipp Miedl, Rehan Ahmed and Lothar Thiele. Gloriastrasse 35, 8092 Zürich, Switzerland: ETH Zürich, July 2017.

[Suz+11] Kuniyasu Suzaki et al. 'Memory Deduplication As a Threat to the Guest OS'. In: *Proceedings of the Fourth European Workshop on System Security*. EUROSEC '11. Salzburg, Austria: Association for Computing Machinery, 2011, 1:1–1:6. ISBN: 978-1-45030-613-3. DOI: 10.1145/1972551.1972552. URL: https://doi.org/10.1145/1972551.1972552.

[Tan02] Andrew Tanenbaum. *Computer Networks*. 4th. Prentice Hall Professional Technical Reference, 2002. ISBN: 978-1-29202-422-6.

[TH12] Tijmen Tieleman and Geoffrey Hinton. 'Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude'. In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[Tho17] Ólafur Jón Thoroddsen. *UnCovert 4: The Power Covert Channel*. Tech. rep. 1. Semester Thesis; Supervisors: Philipp Miedl and Lothar Thiele. Gloriastrasse 35, 8092 Zürich, Switzerland: ETH Zürich, June 2017.

[TS19] 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. Seaside, CA, USA: Association for Computing Machinery, 2019, pp. 298–303. ISBN: 978-1-45036-137-8. DOI: 10.1145/3289602.3293920. URL: https://doi.org/10.1145/3289602.3293920.

[US 85] U.S. Department of Defense. *DOD Trusted Computer System Evaluation Criteria "The Orange Book" [DOD 5200.28]*. National Computer Security Center, 1985. URL: https://csrc.nist.gov/csrc/media/publications/conference-paper/1998/10/08/proceedings-of-the-21st-nissc-1998/documents/early-cs-papers/dod85.pdf.

[VPL10] P. P. Vaidyanathan, See-May Phoong and Yuan-Pei Lin. *Signal Processing and Optimization for Transceiver Systems*. Cambridge Books Online. Cambridge University Press, 2010. ISBN: 978-1-13904-274-1. DOI: 10.1017/CBO9781139042741. URL: http://dx.doi.org/10.1017/CBO9781139042741.

[Wel67] Peter D. Welch. 'The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms'. In: *IEEE Transactions on Audio and Electroacoustics* 15.2 (June 1967), pp. 70–73. DOI: 10.1109/TAU.1967.1161901. URL: https://doi.org/10.1109/TAU.1967.1161901.

[Wil16] Pascal Wild. *UnCovert: Evaluating thermal covert channels on Android systems*. Tech. rep. 1. Semester Thesis; Supervisors: Philipp Miedl and Lothar Thiele. Gloriastrasse 35, 8092 Zürich, Switzerland: ETH Zürich, Aug. 2016.

[WL06a] Zhenghong Wang and R.B. Lee. 'Covert and Side Channels Due to Processor Architecture'. In: *Proceedings of the 22nd Annual Computer Security Applications Conference*. ACSAC '06. USA: IEEE Computer Society, 2006, pp. 473–482. ISBN: 0-76952-716-7. DOI: 10.1109/ACSAC.2006.20. URL: https://doi.org/10.1109/ACSAC.2006.20.

[WL06b] Zhenghong Wang and Ruby B Lee. 'Covert and side channels due to processor architecture'. In: *Proceedings of the 22nd Annual Computer Security Applications Conference*. ACSAC '06. USA: IEEE Computer Society, 2006, pp. 473–482. ISBN: 0-76952-716-7. DOI: 10.1109/ACSAC.2006.20. URL: https://doi.org/10.1109/ACSAC.2006.20.

[WXW15] Zhenyu Wu, Zhang Xu and Haining Wang. 'Whispers in the Hyper-Space: High-Bandwidth and Reliable Covert Channel Attacks inside the Cloud'. In: *IEEE/ACM Trans. Netw.* 23.2 (Apr. 2015), pp. 603–614. ISSN: 1063-6692. DOI: 10.1109/TNET.2014.2304439. URL: https://doi.org/10.1109/TNET.2014.2304439.

[Xu 11] 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. Chicago, Illinois, USA: Association for Computing Machinery, 2011, pp. 29–40. ISBN: 978-1-45031-004-8. DOI: 10.1145/2046660.2046670. URL: https://doi.org/10.1145/2046660.2046670.

[Yar16] Yuval Yarom. *Mastik: A Micro-Architectural Side-Channel Toolkit*. Accessed 21st of May 2019. 2016. URL: https://cs.adelaide.edu.au/~yval/Mastik/.

[YF14] Yuval Yarom and Katrina Falkner. 'FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack'. In: *Proceedings of the 23rd USENIX Conference on Security Symposium*. SEC'14. San Diego, CA: USENIX

Association, 2014, pp. 719–732. ISBN: 978-1-93197-115-7. DOI: 10.5555/2671225.2671271. URL: https://doi.org/10.5555/2671225.2671271.

[YST16] Mengjia Yan, Yasser Shalabi and Josep Torrellas. 'Replayconfusion: Detecting Cache-Based Covert Channel Attacks Using Record and Replay'. In: *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-49. Taipei, Taiwan: IEEE Press, 2016. DOI: 10.5555/3195638.3195685. URL: https://doi.org/10.5555/3195638.3195685.

[Yue+14] Mengchao Yue et al. 'Constructing Timing-Based Covert Channels in Mobile Networks by Adjusting CPU Frequency'. In: *Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '14. Minneapolis, Minnesota, USA: Association for Computing Machinery, 2014. ISBN: 978-1-45032-777-0. DOI: 10.1145/2611765.2611768. URL: https://doi.org/10.1145/2611765.2611768.

[ZBA11] S. Zander, P. Branch and G. Armitage. 'Capacity of Temperature-Based Covert Channels'. In: *Communications Letters, IEEE* 15.1 (2011), pp. 82–84. DOI: 10.1109/LCOMM.2010.110310.101334. URL: https://doi.org/10.1109/LCOMM.2010.110310.101334.

[ZM08] 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. San Jose, CA: USENIX Association, 2008, pp. 211–226. DOI: 10.5555/1496711.1496726,. URL: https://www.usenix.org/legacy/event/sec08/tech/full_papers/zander/zander_html/.

List of Publications

The following list includes publications that form the basis of this thesis. The corresponding chapters are indicated in parentheses.

Philipp Miedl, Bruno Kloptott and Lothar Thiele **Increased reproducibility and comparability of data leak evaluations using ExOT** *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)* IEEE (Chapter 1, Chapter 2, Chapter 3)

Davide B. Bartolini, Philipp Miedl and Lothar Thiele **On the Capacity of Thermal Covert Channels in Multicores** *Proceedings of the Eleventh European Conference on Computer Systems, EuroSys '16* ACM (Chapter 1, Chapter 3)

Philipp Miedl and Lothar Thiele **The Security Risks of Power Measurements in Multicores** *Proceedings of the 2018 ACM Symposium on Applied computing* ACM 2018 (Chapter 1, Chapter 4)

Philipp Miedl, Xiaoxi He, Matthias Meyer, Davide Basilio Bartolini and Lothar Thiele **Frequency Scaling as a Security Threat on Multicore Systems** *CASES'18, International Conference on Compilers, Architectures, and Synthesis for Embedded Systems* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, IEEE (Chapter 1, Chapter 5)

Philipp Miedl, Rehan Ahmed and Lothar Thiele **We know what you're doing! Application detection using thermal data** *Special Issue on Embedded System Security* Leibniz Transactions on Embedded Systems [**Under review**] (Chapter 1, Chapter 6)

Curriculum Vitæ

Personal Data

Name	Philipp Miedl.
Date of Birth	July 23, 1989.
Citizenship	Austria.

Education

2015–2020	ETH Zurich, Switzerland. Computer Engineering and Networks Laboratory. Ph.D. supervised by Prof. Dr. Lothar Thiele.
2011–2014	Graz University of Technology, Austria. Masters program Telematics with the main curricula "System on Chip Design" & "Mobile Computing". Diplom-Ingenieur (equivalent MSc).
2008–2011	Graz University of Technology, Austria. Bachelors program Telematics. Bachelor of Science (BSc)

Professional Experience

2015–2020	ETH Zurich, Switzerland. Computer Engineering and Networks Laboratory. Research and teaching assistant.
2012–2014	Maxim Integrated AG, Graz, Austria Contractor as SoC Verification Engineer (2012) & Firmware Development Engineer (2012–2014)
2005–2010	A1 Telekom Austria AG, Vienna, Austria Internships July/August 2005–2008 & 2010 Web Development, Technical Support Help-desk, Customer order processing