Establishing trusted Machine-to-Machine communications in the Internet of Things through the use of behavioural tests

Thesis submitted in accordance with the requirements of
the University of Liverpool for the degree of Doctor in Philosophy by

Valerio Selis

April 2018

"Be less curious about people and more curious about ideas."

Marie Curie

Abstract

Today, the Internet of Things (IoT) is one of the most important emerging technologies. Applicable to several fields, it has the potential to strongly influence people's lives. "Things" are mostly embedded machines, and Machine-to-Machine (M2M) communications are used to exchange information. The main aspect of this type of communication is that a "thing" needs a mechanism to uniquely identify other "things" without human intervention. For this purpose, trust plays a key role. Trust can be incorporated in the smartness of "things" by using mobile "agents".

From the study of the IoT ecosystem, a new threat against M2M communications has been identified. This relates to the opportunity for an attacker to employ several forged IoT-embedded machines that can be used to launch attacks. Two "things-aware" detection mechanisms have been proposed and evaluated in this work for incorporation into IoT mobile trust agents. These new mechanisms are based on observing specific thing-related behaviour obtained by using a characterisation algorithm.

The first mechanism uses a range of behaviours obtained from real embedded machines, such as threshold values, to detect whether a target machine is forged. This detection mechanism is called machine emulation detection algorithm (MEDA). MEDA takes around 3 minutes to achieve a detection accuracy of 79.21%, with 44.55% of real embedded machines labelled as belonging to forged embedded machines. These results indicated a need to develop a more accurate and faster detection method. Therefore, a second mechanism was created and evaluated. A dataset composed of behaviours from real, virtual and emulated embedded systems that can be part of the IoT was created. This was used for both training and testing classification methods. The results identified Random Forest (RF) as the most efficient method, recognising forged embedded machines in only 5 seconds with a detection rate of around 99.5%. It follows that this solution can be applied in real IoT scenarios with critical conditions.

In the final part of this thesis, an attack against these new mechanisms has been proposed. This consists of using a modified kernel of a powerful machine to mimic the behaviour of a real IoT-embedded machine, referred to as a fake timing attack (FTA). Two metrics, mode and median from ping response time, have been found to effectively detect this attack. The final detection method involves combining RF and k-Nearest Neighbour to successfully detect forged embedded machines and FTA in only 40 seconds, with an overall detection performance (ODP) of 99.9% and 93.70% respectively. This method also was evaluated using behaviours from embedded machines that were not present in the training set. The results from that evaluation demonstrate that the proposed solution can detect embedded machines unknown to the method, both real and virtual, with an ODP of 99.96% and 99.92% respectively.

In summary, a new algorithm able to detect forged embedded machines easily, quickly and with very high accuracy has been developed. The proposed method addresses the challenge of securing present and future M2M-embedded machines with power-constrained resources and can be applied to real IoT scenarios.

Contents

Illustrations

List of Figures

List of Tables

Abbreviations

k-NN	k-Nearest Neighbour
AES	Advanced Encryption Standard
AP	Access Point
API	Application Programming Interface
ARP	Address Resolution Protocol
BI	Business Insider
BLE	Bluetooth Low Energy
BMA	Bad-Mouthing Attack
BS	Base Station
BSA	Ballot-Stuffing Attack
CA	Certification Authority
CCTV	closed-circuit television
CM	Characterisation Metric
CoI	Community of Interest
CPU	Central Processing Unit
CQI	Channel Quality Indicator
CTS	Clear to Send
DCH	Dedicated CHannel
DDoS	Distributed Denial of Service
DNS	Domain Name System
DoS	Denial of Service
DT	Decision Tree
EAP	Extensible Authentication Protocol
EAPoL	EAP over LAN
ERT	Extremely Randomized Trees
ETSI	European Telecommunications Standards Institute

FC	Fog Computing
FES	Final Evaluation Score
FIPA	Foundation for Intelligent Physical Agents
FN	False Negatives
FP	False Positives
FPGA	Field-Programmable Gate Array
FTA	Fake Timing Attack
GMA	Good-Mouthing Attack
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GSMA	Global System for Mobile Communications Association
GTS	Guaranteed Time Slot
H2H	Human-to-Human
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
IBM	International Business Machines
IBS	Intelligent Building Systems
iCore	Internet Connected Objects for Reconfigurable Ecosystem
IDC	International Data Corporation
IDT	Interrupt Descriptor Table
IEEE	Institute of Electrical and Electronics Engineers
IHS	Intelligent Healthcare Systems
IMS	IP Multimedia Subsystem
IoE	Internet of Everything
IoT	Internet of Things
IoT MA	IoT Mobile Agent
IoT-A	Internet of Things - Architecture
IP	Internet Protocol
IPP	Internet Printing Protocol
ITS	Intelligent Transportation Systems
ITU	International Telecommunication Union
L-SVM	Linear kernel-based SVM
L1-FS	L1-based Feature Selection
LDA	Linear Discriminant Analysis

LTE	Long-Term Evolution
M2FH	Machine-to-Fake Human
M2FM	Machine-to-Fake Machine
M2H	Machine-to-Human
M2M	Machine-to-Machine
MAC	Media Access Control
MAP	Mobile Agents Platform
MEDA	Machine Emulation Detection Algorithm
MITM	Man-in-the-Middle
MMS	Multimedia Messaging Service
MQTT	Message Queue Telemetry Transport
MR	Machina Research
NB	Naïve Bayes
NCTA	National Cable & Telecommunications Association
NFC	Near Field Communication
NOP	No Operation
NTP	Network Time Protocol
ODP	Overall Detection Performance
ODS	Overall Detection Speed
OOA	On-Off Attack
OS	Operating System
OSA	Opportunistic Service Attack
P-SVM	Polynomial kernel-based SVM
PPUF	Public PUF
PTO	Pre-Trusted Object
PUF	Physical Unclonable Function
QDA	Quadratic Discriminant Analysis
QoS	Quality of Service
R-SVM	Radial kernel-based SVM
RAM	Random Access Memory
REM	Real Embedded Machine
RF	Random Forest
RFE	Recursive Feature Elimination
RFID	Radio-Frequency IDentification

RSA	Rivest-Shamir-Adleman
RTOS	Real-Time Operating System
RTS	Request to Send
SAMME	Stagewise Additive Modelling using a Multi-class Exponential Loss Function
SCADA	Supervisory Control and Data Acquisition
SHA	Secure Hash Algorithm
SIDT	Store Interrupt Descriptor Table
SIoT	Social Internet of Things
SKB	Select-K-Best
SKB-Chi2	Select-K-Best with Chi-squared stats of non-negative features
SKB-F	Select-K-Best with ANOVA F-value
SMS	Short Message Service
SPA	Self-Promoting Attack
SSH	Secure Shell
ST	Social Trust
SVC	Support Vector Machine Classifier
SVM	Support Vector Machine
T2T	Thing-to-Thing
TCP	Transmission Control Protocol
TKIP	Temporal Key Integrity Protocol
TLS/SSL	Transport Layer Security/Secure Sockets Layer
TMF	Trust Management Framework
TN	True Negatives
TP	True Positives
UES	Unknown Embedded System
UMTS	Universal Mobile Telecommunications System
V2H	Vehicle-to-Human
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
VC	Virtual Continuum
VES	Virtual and Emulated Embedded System
VM	Virtual Machine
WPA	Wi-Fi Protected Access

WPA2	Wi-Fi Protected Access 2
WPS	Wi-Fi Protected Setup
WSN	Wireless Sensor Network
XMPP	Extensible Messaging and Presence Protocol

Preface

This thesis is primarily my own work. The sources of other materials are identified.

Acknowledgements

First and most importantly, I would like to express my deep gratitude to my wife, Dr Alessandra Frau, for her unconditional support for my embarking on and most importantly completing this journey. Her love, patience and research experience have always been inestimable. Thanks a lot for your patience; I could not have done it without you. Secondly, I would like to thank my supervisor, Prof Alan Marshall, for his support and for believing in me during these four years. He gave me the opportunity to begin and finish this journey and his guidance and suggestions have been invaluable. Thirdly, I would like to thank Dr Jonny Milliken, who inspired me as a researcher. It is after working with him in Belfast that I decided to pursue this path.

I would also like to thank Dr Andrew Bolster, a journey mate with whom I shared a house and a lab for a while. Our conversations were a source of great ideas. I want also to thank the guys of the Advanced Networks Research Group for their support and companionship. A special thanks to all the EEE Department staff, including the administrative and technical team, for their advice and help during this journey.

Thanks to the guys of TOM Ltd, in particular to Julian Watts for his friendship. It was challenging but also great to have something else to think about during these years.

I would like to really thank my mother, Maria Consolata, my sister and brothers, Jessica, Maurizio and Marco, and their family members, Epifanio, Lorenzo, Chiara and Elisabetta. They were always present despite the physical distance between us to support, encourage and believe in me. I also thank Mario, Bonaria, Gianmarco, Ilaria, Thomas and Francesco for giving me the opportunity to be part of their family and for their kindness. Thanks to all my friends in Liverpool and back home, who are always there for me, even if we do not see each other so often.

Finally, I would like to dedicate this thesis to the memory of my beloved father, Salvatore, who is a constant source of inspiration.

Chapter 1

Introduction

The number of Internet-connected "things" is increasing rapidly. It is estimated that today there are already between 5 and 15 billion devices connected to the Internet. Around 35 billion "things" are expected to be connected by 2020, of which 13 billion are likely to be wirelessly connected. These "things" connected to the Internet are part of the Internet of Things (IoT) paradigm and their cooperation through Machine-to-Machine (M2M) communications will be fundamental to achieving specific and complex goals. This type of communication is characterised by the fact that there is no human intervention.

The rapidly growing number of "things" connected to the Internet is leading to security issues that have been highlighted by researchers and media news. As these communications increase over time, there is also the opportunity for attackers to exploit them and consequently attack the network or part of it. These attacks can lead to several issues, from the loss of end-user privacy to a large-scale economic impact. It is quite obvious that securing the communication among "things" must be a prioritised factor before continuing to deploy them. In this scenario, trust has an important role in assuring that the exchange of information is secure and reliable. Terms such as "thing", "object", "device", "node" and "machine" are used interchangeably throughout this thesis. Moreover, the term "object" means everything that is not directly related to human beings.

1.1 Internet-connected Things

Objects that will be part of the IoT must have access to the IoT infrastructure in order to communicate with other objects. To do this, objects must have at least one network interface. There are currently various networks that can be used to connect objects in the IoT [2–4]:

- *Wired networks*: a network type in which all the components are connected using cables;

- *Wireless networks*: a network type in which all the wireless components are connected using radio waves such as:
  - *Cellular networks*: a wireless network in which all the mobile components are connected through cells;
  - *Body networks*: a wireless network in which the components are body sensor units;
  - *Vehicular networks*: a wireless network specific for vehicles;
- *Sensor networks*: a wired or wireless network in which the components are sensor nodes.

The general idea is that M2M communications use a wireless data connection as a link between systems, remote devices or locations and individuals [5]. The standards for wireless networks with an important role for connecting machines in the IoT are [2, 4]:

- Wi-Fi networks: IEEE 802.11;
- Bluetooth: IEEE 802.15.1;
- Wireless Sensor Network (WSN): IEEE 802.15.4 or ZigBee;
- Radio-Frequency IDentification (RFID) networks: ISO 18000;
- Cellular networks: Global System for Mobile Communications (GSM), Universal Mobile Telecommunications System (UMTS), IEEE 802.16, Long-Term Evolution (LTE) or 5G.

Table 1.1 shows the objects that can be part of the IoT along with their corresponding features. When available, the main characteristics of these objects may be subdivided as follows [6]:

- *CPU*: from embedded to high performance processor, such as ARM, MIPS, PowerPC, AVR, x86, x86-64, etc.;
- *OS*: open source and proprietary, such as Linux-based, Windows-based, iOS-based, Contiki, etc.;
- *Data storage:* from a few KB to a TB;
- *Random Access Memory (RAM)*: from a few KB to a GB;
- *Network interface*: from one to multiple interfaces at the same time, such as wired, RFID, Bluetooth, ZigBee, Wi-Fi, GSM, etc.;
- *Power*: from no power to high power, battery and/or wired;
- *Type*: mobile or static.

TABLE 1.1: Examples of characteristics of Internet-connected "things".

"Things"	**Type**	**Power**	**Processor**	**Network Interface(s)**
Smart RFID devices (tags, readers, etc.)	M, S	NA/LP	NA/LCPU	Cellular, RFID, Wi-Fi, Wired
Smart body objects (sensors, watches, glasses, bracelets, etc.)	M	LP/MP	LCPU	Bluetooth, Wi-Fi, ZigBee
Smart location objects (Global Positioning System (GPS) based, gyroscope based, accelerometer based, etc.)	M, S	LP/MP	LCPU/MCPU	Bluetooth, Cellular, ZigBee, Wi-Fi, Wired
Robots	M, S	MP/HP	MCPU/HCPU	Bluetooth, Wi-Fi, Wired, ZigBee
Portable devices (smartphones, tablets, etc.)	M	MP/HP	MCPU/HCPU	Bluetooth, Cellular, RFID, Wi-Fi, Wired
Smart building objects (sensors, switches, dimmers, etc.)	S	LP	LCPU	Bluetooth, Wi-Fi, Wired, ZigBee
Network connectivity appliances (routers, gateways, switches, etc.)	S	MP	LCPU/MCPU	Cellular, Wi-Fi, Wired, ZigBee
PCs/servers/NASs	S	MP/HP	MCPU/HCPU	Bluetooth, Cellular, Wi-Fi, Wired, ZigBee
Smart vehicles (cars, bikes, etc.)	M	LP/MP	LCPU/MCPU	Bluetooth, Cellular, Wi-Fi
Smart appliances (fridges, televisions, etc.)	S	LP/MP/HP	LCPU/MCPU	Bluetooth, Wi-Fi, Wired, ZigBee
Smart city objects (street sensors, street lights, traffic lights, etc.)	S	LP/MP	LCPU/MCPU	Cellular, Wi-Fi, Wired, ZigBee

NA: Not Available; M: Mobility; S: Static; LP: Low Power (mW); MP: Medium Power (W); HP: High Power (>kW); LM: Low Memory (kB); MM: Medium Memory (MB); HM: High Memory (>GB); LCPU: Low Unit (MHz); MCPU: Medium Unit (GHz single-core); HCPU: High Unit ($\geq$GHz multi-core).

Most of these things are mainly embedded systems with different capabilities and complexities, starting from a simple smart sensor to a more sophisticated device. These have low-to-medium capabilities, and therefore could easily be compromised by attackers.

Almost every object in the world could be part of the IoT, but every object is not part of it a priori. An object can be part of the IoT if there are at least two main capabilities: connectivity and smart capabilities. For example, a simple temperature sensor cannot be part of the IoT, because there is a lack of intelligence. An object can be defined smart when it has the capability to collect information, in many cases elaborate it, and transfer it to other objects or to the IoT system, whereas connectivity capability allows objects to be connected to and interact with the IoT infrastructure and other IoT objects. In this example, a simple sensor becomes a smart sensor. The intelligent capability can be included in the object by using IoT mobile agents, from now on called "agents". An agent is a piece of software that takes in input information, mostly from the real-world environment, processes it and forwards it to the IoT system or other agents for further processing. An agent can also receive information from the IoT system or other agents with the aim of accomplishing a task. Agents are not like other software applications that can run in objects; these allow objects to intelligently communicate with each other for achieving specific goals. Agents aim to facilitate M2M and Machine-to-Human (M2H) interactions by understanding the environment in which objects are used and to give them the ability to speak the same language [7–9].

1.2 Real-Life Scenarios

At present, there are various applications of M2M communications that have started to emerge in several fields, such as Intelligent Healthcare Systems (IHS), Intelligent Transportation Systems (ITS), Intelligent Building Systems (IBS), smart robots, manufacturing systems and smart grids [3, 10, 11]. In this context, an unprecedented increase in M2M data will occur. It is possible to envision that things will manage and control part of human life in an autonomous way. We will interact with things and these will adapt our environment based on predefined requirements. This will happen primarily in a passive way and without a direct request from the user. The following subsections describe examples of real-life scenarios which highlight the importance of IoT and therefore M2M devices.

1.2.1 Intelligent Transportation Systems

In this first scenario the aim is to reduce fuel consumption, reduce CO_2 emissions, improve vehicle and driving safety, improve traffic efficiency, enable drivers to receive external information etc. In this, M2M devices can create three types of networks: Vehicle-to-Vehicle (V2V), Vehicle-to-Infrastructure (V2I) and Vehicle-to-Human (V2H). In V2V,

information is exchanged among devices; in V2I, information is exchanged between devices and the IoT infrastructure; and in V2H, information is exchanged between devices and humans. This IoT system is composed of smart objects installed and configured in "things" such as vehicles, roads, goods, service areas etc. [12, 13]

A possible ITS scenario involving cars is as follows:

- Improve navigation:
  - Drivers choose to drive or to use the autopilot.
  - The car suggests the fastest route by considering information obtained by other vehicles and the ITS, i.e. traffic, accidents, weather conditions etc.
  - The ITS uses roadside signs to alert vehicles of possible accidents, road congestion etc. If appropriate, it suggests alternative routes and adjusts traffic lights by considering the real-time traffic flow, choosing to change the timing of red and green lights.
- Improve road safety:
  - The car alerts the user to any problem with the vehicle, e.g. low fuel level, low battery power, motor problems etc. It is able to suggest in real-time the closest and cheapest petrol station, car repair garage etc.
  - If the user is driving, sensors help them stay in the correct lane, give information about distances, assist with parking etc.
  - Pedestrians crossing the road may be advised about nearby cars that would not be able to stop in time, suggest the closest cross-walk etc.
  - The ITS alerts cars of any pedestrian crossing the street, especially when there is poor street visibility caused by adverse weather conditions or obstacles such as walls, parked cars etc.
  - In case of an accident, cars can call emergency services automatically by providing the current location, driver and passenger health status, number of vehicles involved etc.
- Improve passenger entertainment:
  - Access the Internet, for example, to download music, videos etc.
  - Adjust audio and luminosity levels in the vehicle.

1.2.2 Intelligent Healthcare Systems

In this scenario, the aim is to improve patient-doctor relationships, increase human health conditions, increase patients' health monitoring, decrease spread of disease, elderly care etc. Here, M2M devices such as smart body sensors, local network devices

etc. would have an important role in reporting patients' conditions and sending alerts to the IoT system [14–16].

Possible future scenarios involving the IHS are as follows:

- Medical records of patients are available everywhere in case of emergency and historical records can be used by the IHS to prevent future health problems.
- In the event of an emergency, information from smart body sensors of a patient may be used to alert the nearest ambulance. Meanwhile, the IHS suggests the fastest route to the nearest hospital.
- In case of a new disease, the IHS predicts its spread before it becomes a national or global risk, alerting national competent authorities and the World Health Organization.
- By collecting information from body sensors, the IHS can advise the user what action to take to prevent a disease.

1.2.3 Intelligent Building Systems

In this scenario, the aim is to optimize building energy consumption, improve building safety and security, optimize building occupancy etc. In this, M2M devices such as smart meters, smart sensors, smart LED lights, smart closed-circuit television (CCTV) cameras, etc. provide specific information to the central system [17–19].

Possible IBS scenarios involving users of buildings are as follows:

- The IBS manages the lighting system to reduce energy consumption depending on user location, external and internal light levels etc.
- The number and location of users in big buildings such as shopping centres, airports, rail stations etc. are used to suggest the fastest way to reach a place of interest (shop, gate, platform, etc.) by considering the flow of people, in which floors the elevators are on, escalator status etc.
- In case of fire or an emergency, the IBS suggests to the user the nearest and safest exit, alerts the fire station and ambulances, turns off all unnecessary equipment in order to contain the fire, limit the damage caused, protect users etc.
- In domestic buildings, the IBS can automatically manage the entertainment system by adjusting the lighting and volume, playing the user's preferred music, suggesting movies etc. It can alert the user when guests are arriving by recognizing their facial features or by their location. It can also alert the user if there is a thief on the premises or inside the home and automatically call the police.

1.3 Threats Against Things and the Internet of Things

The increase in deployment of the IoT is leading to a corresponding increase in attacks against things and the IoT infrastructure. Figure 1.1 shows the trend in top news on attacks against the system based on news websites, technology-based magazines and companies including the BBC [20–33], CNN [34–36], *The Guardian* [37, 38], *The Inquirer* [39–41], *The Register* [42, 43], *The Washington Post* [44], *Technology Review* [45–48], *Scientific American* [49], *Wired* [50–54], *Forbes* [55–61], Computerworld [62–68], PC World [69–74], Symantec [75], Trend Micro [76–90] and Sucuri [91, 92].

Starting from 2010, the year in which IoT began to emerge, things also began to be attacked. The Stuxnet worm was used not only to attack nuclear centrifuges, but also to destroy them. In 2011, embedded things began to be compromised by using wireless connections. A car was attacked via Bluetooth and used to make unauthorised cell-phone calls. Later, in 2013, the core of the IoT infrastructure began to be attacked. Domain Name System (DNS) servers were attacked by using a Distributed Denial of Service (DDoS) attack and the Internet was greatly slowed down. The followed year, both the IoT infrastructure and things were attacked. Sony Pictures was attacked, causing gaming consoles, TVs and smartphones to be disconnected from its entertainment network [56]. In May 2015, the first worst case scenario happened: Internet-connected embedded things such as CCTV cameras were hacked and used to attack the IoT infrastructure by launching a DDoS attack [73]. In the same year, other attacks took place in which embedded things had a major role by being used to attack the IoT infrastructure.

As Figure 1.1 indicates, from 2012 on, embedded things were increasingly targeted by attackers and then used to attack the IoT infrastructure or part of it. These attacks happened because embedded systems are left mostly without any kind of protection. This exposure gives attackers an opportunity to compromise and remotely use them for malicious purposes. Recently, hackers compromised a fish tank in a North American casino to attack its network and gather private information. It was used to send data to another device in Finland [93]. Another news item showed how a security flaw, called "Devil's Ivy", can be used to compromise IoT devices from large 24 companies, including Bosch, Canon, Cisco, D-Link, Fortinet, Hitachi, Honeywell, Huawei, Mitsubishi, Netgear, Panasonic, Sharp, Siemens, Sony and Toshiba [94]. This flaw allows access to the IoT devices by giving attackers full remote control, which gives them the opportunity to use these devices for perpetrating other attacks. It should be noted that Figure 1.1 shows only the reported attacks, therefore, the total number of attacks perpetrated against the IoT may be greater. This could be because companies usually do not report attacks in order to avoid reputational damage, therefore there could be attacks that, although unreported, took place nevertheless.

Today, automated systems are present in the field for monitoring and scanning IoT devices and their vulnerabilities, such as Masscan [95], Nmap [96], Shodan [97] etc. In particular, Shodan gives the opportunity to search IoT devices by specifying their type, location etc. Results are shown in a graphical interface with details of the devices

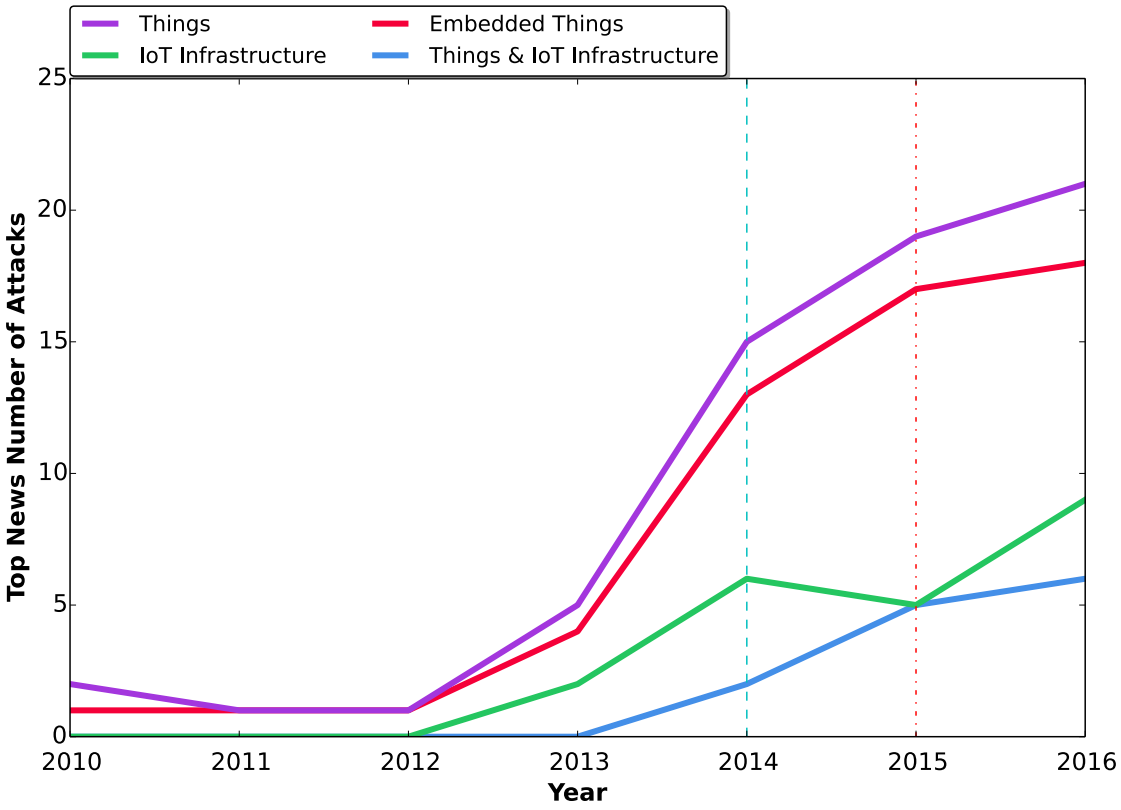


FIGURE 1.1: Trend of top attack news against things and the IoT infrastructure. Attacks against embedded things are highlighted. Attacks related to both things and the IoT infrastructure are shown. In 2013, things and the IoT infrastructure were attacked for the first time and at the same time, while in 2015, compromised embedded things were used to attack the IoT infrastructure.

including IP address, location, open ports, credentials etc. In 2013, more than 1.2 billion Internet-connected machines were tracked down, with an average of 300 million new scan probes per month [98]. Table 1.2 shows results from a Shodan search of IoT devices, common services and ports, and major companies that provide networking equipment.

The results show that there are between 4 and 14 million possible IoT-embedded devices present in the database considering major companies. There are also around 16,000 possible Supervisory Control and Data Acquisition (SCADA) systems and 24,000 devices using Message Queue Telemetry Transport (MQTT), a well-known connectivity protocol for M2M communications accessible from the Internet.

1.4 Motivation, Aims and Objectives

The trend of attacks against the system and things is growing yearly, and security measures are therefore paramount to ensure their protection. Securing the IoT must be prioritised before continuing to deploy it in the real world and on a large scale. To continue to create an insecure IoT system can lead to an increasing number of attacks against the IoT infrastructure and IoT machines. This can affect human lives greatly; in fact, corruption of IoT systems such as ITS, IHS and IBS can endanger people's

Table 1.2: Shodan results for IoT devices, companies, common ports and embedded OSs. Results were retrieved on the 28^{th} of January 2017.

	Search	Results
Common devices	Camera	326,447
	Dreambox	4,876
	Firewall	101,169
	NAS	72,666
	Polycom	8,514
	Printer	153,359
	Switch	167,350
	Router	2,352,549
	Wireless	792,802
Major companies	ASUS	71,102
	Cisco	9,980,648
	Hewlett-Packard	28,371
	Huawei	525,838
	Linksys	34,303
	Netgear	246,236
	Panasonic	19,800
	Samsung	40,229
	Sony	4,793
	Synology	40,472
	TP-Link	493,590
	Ubiquiti	2,069,077
	VMware	172,184
	ZyXEL	522,041
Common TCP ports	port:22 - Secure Shell (SSH)	13,392,387
	port:23 - Telnet	6,842,709
	port:80 - Hypertext Transfer Protocol (HTTP)	65,716,279
	port:443 - HTTP Secure (HTTPS)	45,495,186
	port:502 - Modbus-SCADA	15,926
	port:631 - Internet Printing Protocol (IPP)	496,405
	port:902 - VMware ESXi	169,671
	port:1883 - Message Queue Telemetry Transport (MQTT)	24,314
	port:5001 - Synology Inc.	457,176
	port:5222 - Extensible Messaging and Presence Protocol (XMPP)	228,756
Embedded OSs	Android/2.2	52,660
	Contiki	202
	DD-WRT	52,287
	OpenWRT	19,215
	Raspbian	87,148
	Real-Time Operating System (RTOS)	971
	Tiny WebServer	2,617
	Embedded	325,475
	IoT	4,763

lives. Machines, and especially embedded machines, must have the capability to protect themselves and their communications. In this scenario, trust has an important role in assuring that the exchange of information is secure. Trust can be defined as "the belief that another party will behave according to a set of well-established rules and thus meet one's expectations" [99].

In the IoT paradigm, there are several open issues. The first issue relates to the security of data collected and transmitted by M2M devices inside the network. The second issue is network security, which refers to the integrity of communication between M2M devices and the core network. Another issue relates to the security scalability of the system involving billion of devices exchanging information. A further issue is the lack of a standardised framework for M2M communications in the IoT, including a Trust Management Framework (TMF). Moreover, major attention needs to be paid to the security of embedded machines and their communications due to their specific characteristics, such as low capabilities that make them more susceptible to attacks and easily compromised and most important, the fact that they manage a large amount of data.

These issues are enhanced if several M2M devices misbehave inside the network at the same time. In this situation, it will be difficult for other well-behaving M2M devices to exchange information securely and detect and isolate attackers. Recently, attackers have begun to use Virtual Machines (VMs) to launch attacks [100]. VMs allow an attacker to:

- Generate multiple distributed attacks at the same time by using one or more powerful machines;
- Be undetected by the victim, because each VM will be identified as a real machine;
- Remove attack traces easily by deleting a file or restoring the previous VM status.

At the moment, there are no mechanisms to properly detect VMs in a network, which makes these types of devices extremely vulnerable to attacks. Instead, most research efforts are focused on protecting VMs from attackers, as these are mostly used in cloud-based systems. Another aspect is ensuring VMs are isolated to prevent an attacker from seeing information belonging to other VMs in the same server. A third area of research is focused on the use of VMs to analyse behaviours of computer malware. In fact, these provide a perfect isolated environment for studying their behaviours without compromising the host system. Furthermore, VMs are used by researchers to simulate attackers and therefore launch attacks in a network [101–109]. However, there has been no special efforts thus far to detect and prevent attackers from using VMs as an attack vector. Table 1.2 shows that online there are at least 170,000 VMware ESXi Hypervisors that theoretically could be used to run multiple embedded systems for attacking the IoT network.

The aim of this thesis is to investigate new methodologies that permit machines to reliably trust each other in order to securely exchange information in IoT M2M

communications, with special attention to embedded systems. The objectives are firstly, to understand what the IoT architecture should be and how to establish reliable trust metrics among machines, and secondly, to evaluate and compare TMFs adopted with the goal of understanding their effectiveness. This will be achieved by considering the proposed IoT standards and models. Finally, an important issue related to the security in the IoT system will be identified and addressed. This consists of the opportunity for an attacker to use multiple virtual and emulated embedded systems (VESs) to replicate real embedded machines (REMs), and consequently to threaten the IoT system. This represents a problem for (i) the security of data collected and transmitted by M2M devices inside the network and (ii) the integrity of communication between M2M devices and the core network. The lack of framework standardisations for M2M communications in the IoT reduces the opportunity to find a solution to address this problem. Despite this obstacle, in the course of this research, a solution that can be used in different circumstances and is easily applicable to future IoT devices has been identified. The basis for this solution is the employability of mobile agents used as vehicles of trust. Mobile agents retrieve the information needed for performing the computation and evaluation of trust, which is the main aspect studied in this thesis. This is one of the most reliable mechanisms for creating M2M communications among IoT devices.

1.5 Contributions and Outline of the Thesis

Chapter 1: Introduction

A brief introduction to Internet-connected things and IoT was carried out by considering real-life scenarios and the threats associated with them. A context-aware classification and review of the capabilities of IoT devices was presented in Section 1.1, underlining the importance of securing embedded systems.

This chapter highlights the importance of securing the IoT system by looking at problems related to real-case scenarios (Section 1.2). In Section 1.3, an up-to-date review of attacks against IoT devices and the IoT architecture was carried out by looking at top news trends worldwide. In this way, the motivation, aims and objectives of this work were highlighted (Section 1.4).

Chapter 2: Background and Related Work

Background information related to the current status of IoT (Section 2.1), M2M communications (Section 2.2), emerging trends in IoT (Section 2.3) and TMFs (Section 2.4) applied to them are outlined in this chapter.

In Section 2.5, a study of current issues associated with deploying available TMFs in IoT/M2M scenarios is presented. Particular attention is paid to issues relate to deploy these solutions by using IoT/M2M-embedded devices.

Chapter 3: A New Threat and a Novel Solution: Machine Emulation Detection Algorithm

In this chapter, a new threat against IoT/M2M devices is studied. A background analysis of current approaches to detect this threat is provided. A novel detection method for addressing this problem is proposed, along with a comparison with other techniques.

In Section 3.1 the threat model is presented in which multiple VESs are used to attack the M2M communication. Moreover, worst-case scenarios applied to real life scenarios are shown to highlight the real threats that could occur in unsecured situations (Section 3.2). Section 3.3 provides a background of current solutions used for detecting VESs. In Section 3.4, a novel solution, called Machine Emulation Detection Algorithm (MEDA), to this problem is presented. Results are presented in Section 3.5, followed by a comparison with current methods to detect VESs and MEDA.

Chapter 4: A Classification Approach to Detecting Forged Embedded Machines

A new classification-based approach to detect illegitimate embedded machines in IoT M2M communications is described in this chapter. In order to choose the best solution, several steps were carried out. These steps include the creation of a dataset, the evaluation of several feature selection methods and classification algorithms, and a review of results from a simulated attack.

The background to and motivation for designing a new approach to detecting VESs in IoT is presented in Section 4.1. This approach resolves some issues related to the MEDA by using and testing several classification methods. A new classification-based approach is then studied and presented in Section 4.2. The detection of different VESs is described and an extensive analysis of simulation results is proposed in Section 4.3, followed by a comparison with MEDA.

Chapter 5: Attack and Defence in Behavioural Tests

An attack against the behaviour detection methods proposed in this work is shown in this chapter. A defence method resilient against this attack is then illustrated. Finally, the performance of this solution is tested using real and virtual embedded systems that are not present in the dataset. This gives the opportunity to study the applicability of the detection method to future IoT machines.

In Section 5.1, the threat model against behavioural tests is presented by highlighting a potential attack. A simulated fake timing attack in which the kernel of a powerful machine is modified by using information from the dataset in order to mimic the behaviour of an REM is studied and illustrated in Section 5.2. In Section 5.3, the detection model is presented which makes improvements to the classification-based approach presented in Chapter 4. Simulations and results obtained in detecting this attack are shown in Section 5.4, alongside a comparison with behavioural tests. In order to evaluate the

performance of the final detection method, unknown embedded VESs and REMs are tested (Section 5.5). This solution is proven to be applicable to different IoT M2M scenarios; it operates with different standards and IoT devices independently by the system architecture and the OS. It is also resilient against timing attacks and capable of classifying unknown IoT devices, making it easily applicable to future IoT-embedded systems. The applicability of the proposed solution to architectural reference models proposed, the feasibility of its implementation and its benefits in real-life scenarios are shown in Section 5.6.

Chapter 6: Conclusions and Future Work

Final conclusions and considerations in evaluating the applicability of the solution proposed in Chapter 5 are presented in Section 6.1. In this chapter, the contribution and findings of this thesis are shown (Section 6.2). In Section 6.3, further research directions on the subject are proposed. Finally, a summary of the research is presented in Section 6.4. Additional information and bibliographic references are provided after this chapter.

1.6 Publication List

The following publications are the outputs of my research findings:

Valerio Selis and Alan Marshall, 'MEDA: a Machine Emulation Detection Algorithm', in *Proceedings of the 12th International Conference on Security and Cryptography (SECRYPT 2015)* [6].

Valerio Selis and Alan Marshall, 'A classification-based algorithm to detect forged embedded machines in IoT environments', in *IEEE Systems Journal* (accepted for publication) [110].

Valerio Selis and Alan Marshall, 'A Fake Timing Attack Against Behavioural Tests Used in Embedded IoT M2M Communications', in *Proceedings of the 1st Cyber Security in Networking Conference (CSNet'17)* [111].

Chapter 2

Background and Related Work

In this chapter the background analysis related to the Internet of Things (IoT), Machine-to-Machine (M2M) communications and trust is provided. Current work in these fields is presented and the architectural reference models for IoT and M2M in IoT, emerging trends in IoT and Trust Management Frameworks (TMFs) for IoT are highlighted. An overview of the limitations of currently proposed solutions for trust in IoT are then identified.

2.1 Internet of Things

The impact of the IoT began to be acknowledged in 2010, when the first estimate of the number of Internet-connected devices was made by Ericsson, who envisaged that 50 billion devices would be connected to the Internet by 2020 [112]. In 2011, Cisco confirmed this estimate [113], while Machina Research (MR) forecasted around 24 billion, less than half the number forecast in other estimations. The following year, the International Business Machines (IBM) corporation expected 1 trillion "things" to be connected by 2015 [114]. This last estimation was too optimistic and has proven to be completely false, as shown in Table 2.1. Between 2013 and 2016, other organizations such as Global System for Mobile Communications Association (GSMA), International Data Corporation (IDC), Gartner, National Cable & Telecommunications Association (NCTA), Business Insider (BI) and IHS Markit have tried to estimate the number of IoT devices. The prediction of 50 billion devices connected by 2020 made by Ericsson and Cisco became very popular. While Cisco stands by that expectation, Ericsson has been adjusting their predictions yearly and reduced it by half in 2016. Today, the Cisco estimate seems to be unrealistic. Meanwhile, a figure of around 30 billion by 2020 seems to be more realistic. Recent estimates from IHS Markit [115], MR [116] and Ericsson [117] are very different, ranging from 29 billion in 2022 to 27 or 75.4 billion in 2025. These numbers underscore the high degree of uncertainty in relation to when the IoT will be ready to work properly and to be deployed worldwide. Trendlines of these forecasts are shown in Figure 2.1.

TABLE 2.1: Forecasts of connected IoT devices in billions.

Source (Date)	Year															
	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025
Ericsson (April 2010) [112]											50					
Cisco (April 2011) [113]											50					
MR (October 2011) [118]		9									24					
IBM (May 2012) [114]						1000										
Cisco (July 2013) [119]			8.7	10							50					
GSMA and MR (August 2013) [120]	8	9	10.2*	11.7*	13*	14.8*	16.5*	18.5*	20.6*	22.9*	25.2*	27.9*	30.6*			
IDC (May 2014) [121]				9.1							28.1					
Gartner (November 2014) [122] **				3	3.8	4.9					25					
NCTA (March 2015) [123]			8.7	11.2	14.2	18.2	22.9	28.4	34.8	42.1	50.1					
BI (April 2015) [124]					5.8*	7.5*	10.7*	15*	20.1*	26.2*	34*					
Gartner (November 2015) [125] **					3.8	4.9	6.4				20.8					
Ericsson (November 2015) [126]					14	16*	17.7*	19.5*	21.5*	23.7*	26.2*	28				
IHS Markit (April 2016) [115]						15.4					30.7					75.4
MR (August 2016) [116]						6										27
BI (September 2016) [127]						4	7	10	13	19	24					
Ericsson (November 2016) [117]					12	14*	15.6*	17.4*	19.3*	21.5*	23.9*	26.5*	29			

* Estimated values; ** PCs, smartphones and tablets are not included

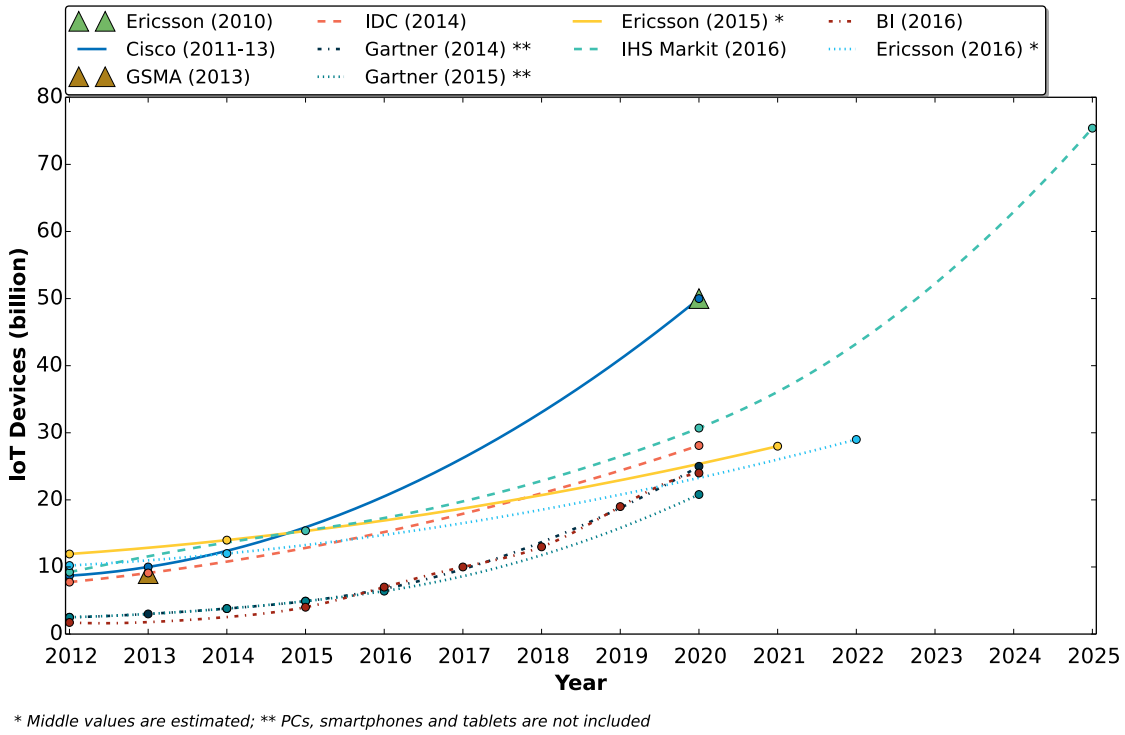


FIGURE 2.1: Forecast of connected IoT devices.

Currently, there is no agreed definition of the "Internet of Things" (IoT); the concept is continuously evolving. The origins of the term are also in question. According to [128], the term IoT was coined by Peter T. Lewis in 1985 and described as "the integration of people, processes and technology with connectable devices and sensors to enable remote monitoring, status, manipulation and evaluation of trends of such devices". However, most publications give credit to Kevin Ashton, who mentioned the term in association with Radio-Frequency IDentification (RFID) technologies during a presentation in 1999, in which he stated that "the IoT has the potential to change the world, just as the Internet did. Maybe even more so". Moreover, some researchers [129–132] emphasised that the concept was not new and that it was envisioned by Nikola Tesla in 1926 during an interview with *Colliers* magazine conducted by John B. Kennedy [133]. In that interview, Tesla stated that:

when wireless is perfectly applied the whole earth will be converted into a huge brain, which in fact it is, all things being particles of a real and rhythmic whole. [...] and the instruments through which we shall be able to do this will be amazingly simple compared with our present telephone. A man will be able to carry one in his vest pocket.

Today, the Oxford English Dictionary Online defines the IoT as "*the interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data*" [134]. Tan and Wang [135] described the IoT as a system with

new types of communication moving from a Human-to-Human (H2H) era to a Thing-to-Thing (T2T) era. At every time and in every place, there will be connectivity for everything, allowing information to be always available. Things will become smart in order to process information, self-configure, self-maintain and self-repair, and especially to make autonomous decisions. Authors in [136] described the IoT as a combination of people, processes and devices with the capability to sense and act. This will enable a variety of smart objects to make real-time decisions by using innovative addressing schemes, secure communications and standardised frameworks for their interactions. The Telecommunication Standardization Sector of the International Telecommunication Union (International Telecommunication Union (ITU)) defines the IoT as "*a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies*" [137]. In 2015, the Institute of Electrical and Electronics Engineers (IEEE) imagined the IoT as "*a self-configuring and adaptive complex system made out of networks of sensors and smart objects whose purpose is to interconnect 'all' things, including every day and industrial objects in such a way to make them intelligent, programmable and more capable of interacting with humans*" [138].

These definitions of the IoT imply several challenges that must be addressed such as: real-time processing, information distribution, automated system management and monitoring, interoperability of different systems, scalability, security, mobility, heterogeneity, service discovery and delivery etc. In this thesis, the ITU definition is used with the extension that interconnected objects must be smart to be part of the IoT system.

2.2 Machine-to-Machine Communications

While the IoT is the future, its foundations are a form of enhanced Machine-to-Machine (M2M) communications. An M2M communication refers to a type of communication that enables machines to communicate with each other mainly without human intervention. This type of communication allows machines to collect data from the real world; thereafter, through applications, this data is made available to the end-user. An M2M solution has four main components:

- M2M Device: this is a physical device used to collect data from the real world which it then sends to the network or to other M2M devices. As discussed in Chapter 1, these devices can vary from a low-end sensor to a complex, high-end system. However, not all M2M devices can be part of the IoT system, as emphasised previously in this chapter;
- M2M Application: this is the application that collects and manages data from M2M devices. This usually resides in a remote server;
- Network: this enables the connectivity among M2M devices and M2M applications;

- M2M Service Enablement: provides identical functionalities among different applications.

The main differences between the IoT and M2M communications are that the latter are mainly focused on point-to-point communications, while the former use the Internet to exchange data among devices and cloud-based systems. Moreover, in M2M communications, the information collected is used for a specific solution and a specific location, while in the IoT, information from several locations and devices are managed at the same time, generating big data and analytics and serving various applications. Despite these differences, in the IoT, machines communicate in an identical way as in the M2M scenario. Therefore, securing M2M communications in IoT is the initial step for securing the entire IoT system. In fact, if machines are not able to securely exchange information among themselves and the IoT core, the IoT system or part of it can be easily compromised.

2.3 Emerging Trends in the IoT

In the following subsections, an overview of emerging trends and extensions of the IoT, architectural reference models and standards that have been proposed for the IoT are presented. There are three main trends in the IoT that are influencing its evolution. The first is an emerging paradigm called the Social Internet of Things (SIoT); the second is the introduction of the Virtual Continuum (VC) and the last is the introduction of Fog Computing (FC).

2.3.1 Social Internet of Things (SIoT)

The SIoT paradigm was introduced by Atzori *et al.* in 2011 and it had a significant impact in the IoT area. This subdivide the IoT into social structures [139, 140] by applying the social networks of human beings to the IoT system. The definition of smart objects is enhanced by adding a social capability which enables objects not only to manage information in a smart way, but also to create social relationships with other objects. Each social object can be identified by its owner, a unique ID and a profile. This paradigm is based only on relationships among things, not among their owners, so things play a key role in the SIoT system. A representation of the SIoT architecture is shown in Figure 2.2.

This is a typical client-server architecture, in which the server (SIoT Server) can communicate with two types of clients, an IoT gateway and a social object. A social agent is used in servers in order to communicate with the client and other gateways and/or social objects. The SIoT architecture can be subdivided into different layers for both server and clients (gateway and object). In terms of the server, there are two layers:

1. Network layer: manages the physical communication;

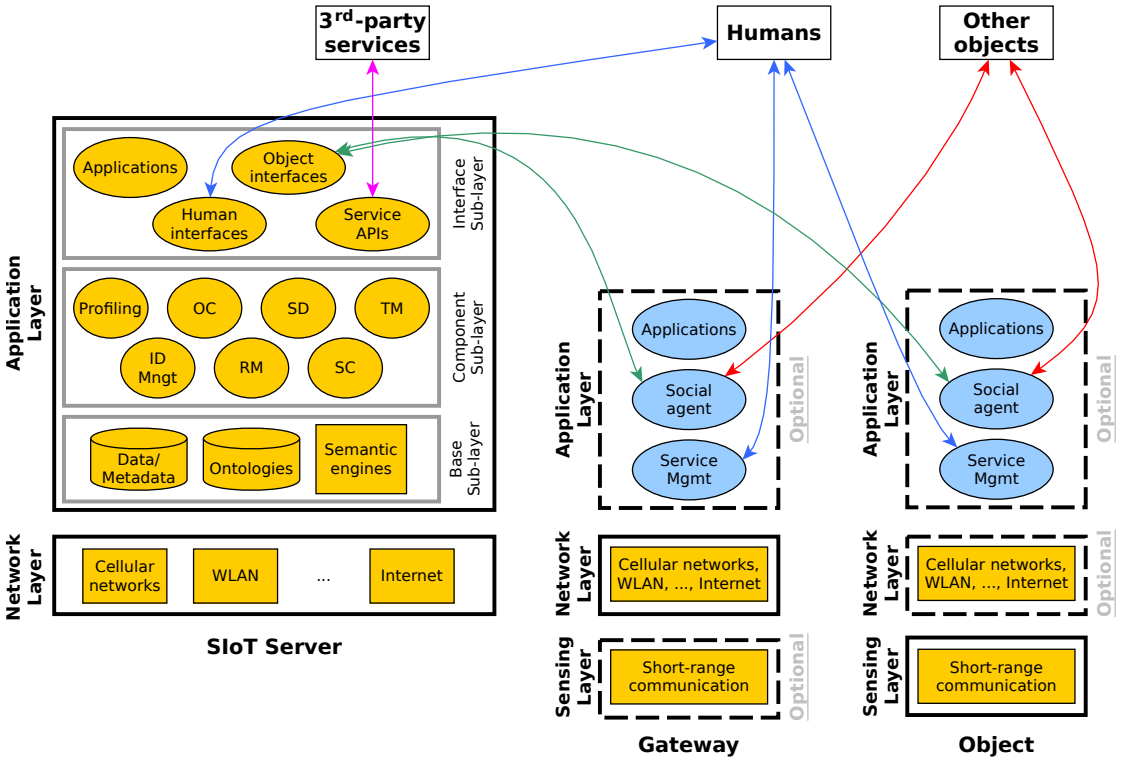


FIGURE 2.2: SIoT architecture.

2. Application layer: consists of three sub-layers:
   - Base sub-layer: data storage and management;
   - Component sub-layer: tools for component implementation;
   - Interface sub-layer: interfaces with SIoT components (objects, human etc.);

In terms of clients, there can be three layers:

1. Sensing layer: a physical object;
2. Network layer: manages the physical communication;
3. Application layer: this is composed of (i) applications, (ii) a social agent that manages the communication among objects and the server and (iii) a service management that provides the interface with humans.

In this scenario, there are new kinds of relationships, as shown in Figure 2.3. These relationships are:

- Ownership: among objects that belong to the same user;
- Co-work: among objects that collaborate to provide common SIoT services;
- Co-location: among objects that are in the same place;
- Social: among objects that interact with each other;

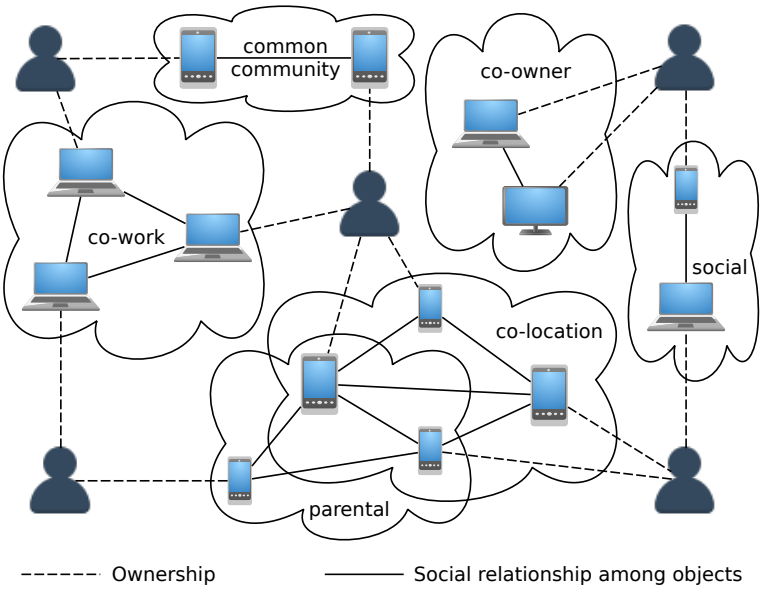


FIGURE 2.3: Types of relationship in the SIoT paradigm.

- Parental: among objects that belong to the same manufacturer.

The main role of the owner of the objects is to register them in the SIoT system (SIoT Server). Each object is assigned to a specific class according to its features:

- Class 1: mobile objects with large computational and communicational capabilities;
- Class 2: static objects with significant computational and communicational capabilities;
- Class 3: objects with only sensing capabilities;
- Class 4: Radio-Frequency IDentification (RFID) or Near Field Communication (NFC)-based objects.

Another role of the owner of the objects is to define policies governing the operations performed, i.e. what information to share, what relationships are permitted etc.

2.3.2 Virtualisation Continuum (VC)

In the same year that SIoT was introduced, Alam *et al.* [141] had the idea of introducing the Virtualisation Continuum (VC) to the IoT. As initially conceived, this involved the introduction of the IoT virtualisation framework that provides an interface between the application layer and the real devices. Subsequently, virtual objects were introduced in the Internet of Things - Architecture (IoT-A) European project; these were called virtual entities, as highlighted previously in this chapter. A virtual object can represent many real objects in the Internet, depending on how many functionalities these can provide. In 2014, the European project Internet Connected Objects for Reconfigurable

Ecosystem (iCore) [142] introduced into the architectural reference model the virtual object level, in which many virtual objects can be represented by many real objects. The VC allows a virtual representation of real objects in the digital world. Each virtual object provides augmented functionalities of real objects, thereby increasing their computational capabilities, data storage, network connectivity etc. The virtual object will be able to support different devices independent of the vendor, thereby facilitating the communication of real objects in the IoT system.

Figure 2.4 shows a possible IoT architecture within the virtualisation layer. The virtualisation layer has two levels. The first enhances the functionalities of the virtual object by providing authentication, addressing and naming, search and discovery, and mobility management. The second is the virtualisation specific level, which provides cognitive management, context awareness and semantic description capabilities. The virtual objects' management block provides functionalities related to the creation, maintenance and coordination, and deletion of virtual objects in the virtualisation layer.

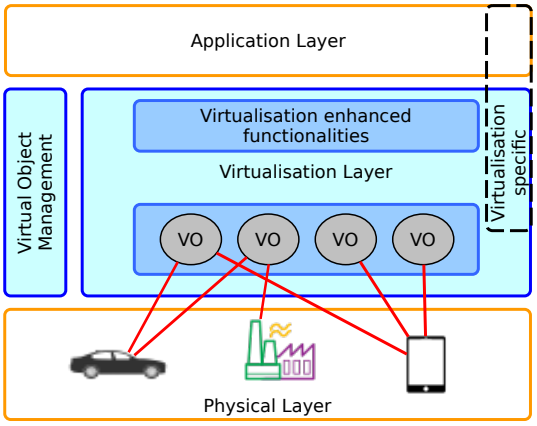


FIGURE 2.4: Virtual Continuum in the IoT architecture.

2.3.3 Fog Computing (FC)

In 2012, Cisco Systems [143, 144] introduced the concept of integrating Fog Computing (FC) into the so-called Internet of Everything (IoE). Cisco defines the IoE as an evolution of the IoT in which everything can communicate with everything through the Internet. The FC should help the scalability of the IoE system by decentralizing processes that do not require information from the Cloud. An important aspect of the Fog is its role in managing M2M interactions. It collects and processes the data and issues control commands to the actuators. It also filters the data that needs to be used locally and sends the rest to the Cloud. This solution aims to allow real-time operations and processing in M2M interactions. A representation of the FC is shown in Figure 2.5.

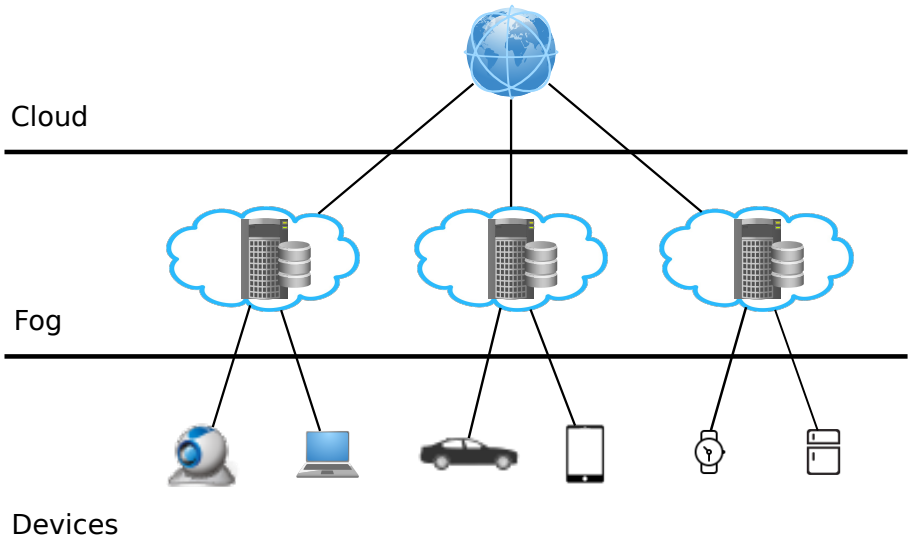


FIGURE 2.5: Fog Computing in the IoE system.

2.4 Information Security and Trust

As highlighted in the introduction and suggested by different organizations and researchers, security has an important role in M2M communications in the IoT. The main role of information security is to protect the information in a communication system. It refers to the conjunction of three components: confidentiality, integrity and availability. Confidentiality and integrity refer to the fact that all the private information must be protected from access and manipulation by unauthorised users, both internal or external. Availability indicates that all information must be accessible to authorised users during the normal activity of the system. Violations of these components caused by malicious users are called respectively disclosure (i.e. data poisoning), alteration (i.e. decryption and cracking) and denial (i.e. Denial of Service (DoS)) [145].

Attacks can be categorised as passive or active and the attacker can be inside or outside the network. A summary of the potential attacks against wireless networks used by Internet-connected things is shown in Table 2.2.

TABLE 2.2: Examples of attacks against wireless networks used by Internet-connected things.

RFID [146]		
Physical Layer	Tag and/or reader removal/destruction	Passive and active interference
	Relay attacks	KILL command
Network and Transport Layer	Tag cloning and spoofing	Network protocol attack
	Reader impersonation and eavesdropping	
Application Layer	Unauthorized tag reading	Tag modification
	Buffer overflows	Malicious code injection

Multilayer	Covert channel and DoS	Traffic analysis
	Crypto attacks	Side channel attacks
	Replay attacks	
Wi-Fi [147–152]		
Physical Layer	Passive and active scanning	Access Point (AP) removal/ destruction
	Radio Frequency jamming attack	
Data Link Layer	Association/ Authentication/ Deauthentication/ Disassociation/ Beacon/ PS Poll/ Clear to Send (CTS)/Request to Send (RTS) flood	
	Michael (MIC) shutdown exploitation Temporal Key Integrity Protocol (TKIP) attack	
	Unauthenticated association	Chopchop attack
	KoreK attack	Wi-Fi Protected Setup (WPS) attack
	Media Access Control (MAC) address spoofing	4-Way handshake blocking
Network Layer	802.1X Extensible Authentication Protocol (EAP) Length Attacks/ 802.1X EAP-of-Death	
	Ping of Death attack / LAND attack	
	EAP over LAN (EAPoL)-Start/Logoff Attack	Internet Protocol (IP)/DNS spoofing
	SYN Flooding attack	Smurf attack
	Premature EAP Success/-Failure attack	NULL data attack
Multilayer	Address Resolution Protocol (ARP) replay attack	
	Wi-Fi Protected Access (WPA)/WPA2 four-way handshake - dictionary attack	
	Rogue AP/Evil Twin	Chameleon attack
	MITM attack	802.1X RADIUS cracking
ZigBee - IEEE 802.15.4 [153, 154]		
Physical Layer	Jamming	Eavesdropping
	Node tampering	Nodes removal/destruction
Data Link Layer	Injection and alteration	DoS and integrity protection attack on Advanced Encryption Standard (AES)-CTR
	Back off interval manipulation	Guaranteed Time Slot (GTS) attack
	ACK attack	PANId conflict

	Denial of sleep	CCMP known plaintext recovery
Network Layer	Routing attacks	Hello floods
	Network flooding attacks	Wormhole attack
	Compromised nodes on path suppress or alter packets	
Transport Layer	Desynchronization attack	
Application Layer	Overwhelming sensors	Path-based DoS attack
	Deluge (reprogramming) attack	
Multilayer	DoS battery-drainage (if battery is present)	
	Node replication	
	Cellular [155–159]	
Mobile-Base Stations communications	MITM attack in GSM/UMTS scenario	
	Network bandwidth saturation with fake Channel Quality Indicator (CQI)/Signalling attack	
	Mobile Station impersonation	GSM Base Station impersonation
	Greedy handoffs attack	Proportional Fair based attack
Transport Layer	Dedicated CHannel (DCH) starvation DoS attack	
IP Multimedia Subsystem (IMS) layer	Routing attacks	DNS attacks
	DoS attack	
Multilayer	Multimedia Messaging Service (MMS) Battery draining attacks	
	Short Message Service (SMS) flooding	

Three key issues can be identified in relation to IoT security [160, 161]:

1. Privacy and data confidentiality: only authorised entities can access and modify data according to defined rules under which data referring to individual users may be accessed;

2. Security: refers to rules for securing the communication among IoT objects in order to eliminate network-based threats;

3. Trust: refers to security policies regulating access to critical resources and the credentials that are required to satisfy these policies.

2.4.1 Encryption Techniques in the Internet of Things

The aim of using encryption techniques during data transmission is to prevent unauthorised access to the information transmitted. Cryptographic algorithms are used to encrypt this information. Before encryption, this information is called plaintext; after encryption it is called ciphertext. There are two main layers in which encryption techniques are used in the IoT [162, 163]:

1. *Network layer*: a by-hop encryption mechanism is used to provide an encrypted network link among nodes. An example of link encryption for a Wi-Fi network is the Wi-Fi Protected Access 2 (WPA2) based on the AES;

2. *Application layer*: an end-to-end encryption mechanism is used to provide encrypted information. The most common cryptographic protocol used in the Internet for encrypting data at this layer is the Transport Layer Security/Secure Sockets Layer (TLS/SSL).

Encryption at the network layer is commonly based on a symmetric encryption, in which all nodes in the network use the same key to encrypt and decrypt the link, usually called a pre-shared key. If only the network layer encryption is used, nodes inside the network can read the plaintext. Commonly used to protect information from outsiders, this method is used only when all nodes in the network are trustworthy.

Encryption at the application layer is commonly based on an asymmetric/public-key cryptographic algorithm. This consists of different keys, called a private/public key pair, for encrypting and decrypting the information. In particular, each node in the network has a private key, which is known only by that node, and a public key, which can be seen by all the other nodes and is correlated to the private key. The main aspect is that nodes can obtain the public key of a node but not its private key. The most well-known asymmetric algorithm is RSA based on a factorisation problem.

A simplified explanation of this is as follows. A node in the network, "A", selects two large prime numbers, p and q, with $p \neq q$. It will then calculate the product of these prime numbers (n), which will be the common factor between the private and public keys. Furthermore, it will calculate the Euler totient function as:

$$\phi(n) = (p-1)(q-1) \tag{2.1}$$

Next, "A" will select an integer (e) that is relatively prime to the result obtained by the Euler totient function and is less than that result, as:

$$gcd(\phi(n), e) = 1;\ 1 < e < \phi(n) \tag{2.2}$$

Finally, "A" will determine a value d such that $d \equiv e^{-1} \pmod{\phi(n)}$. Therefore, "A" will have as its private key a pair obtained by d and n ($A_{PR_k} = \{d, n\}$) and as its public key a pair obtained by e and n ($A_{PU_k} = \{e, n\}$). At this point, a node "B" that

would like to securely communicate with "A" will retrieve A_{PU_k}, encrypt a randomly generated key (K) by calculating $C = K^e \bmod n$ and send it to "A". "A" can use A_{PR_k} for decrypting C by calculating $K = C^d \bmod n$. At this point, "A" and "B" can use K as shared key to symmetrically encrypt the information exchanged [164, 165].

This type of encryption can be subject to a so-called MITM attack in which an attacker node is between the communication of two nodes during the key exchange process. This attack is shown in Figure 2.6 and allows the attacker to eavesdrop, inject or alter the information exchanged.

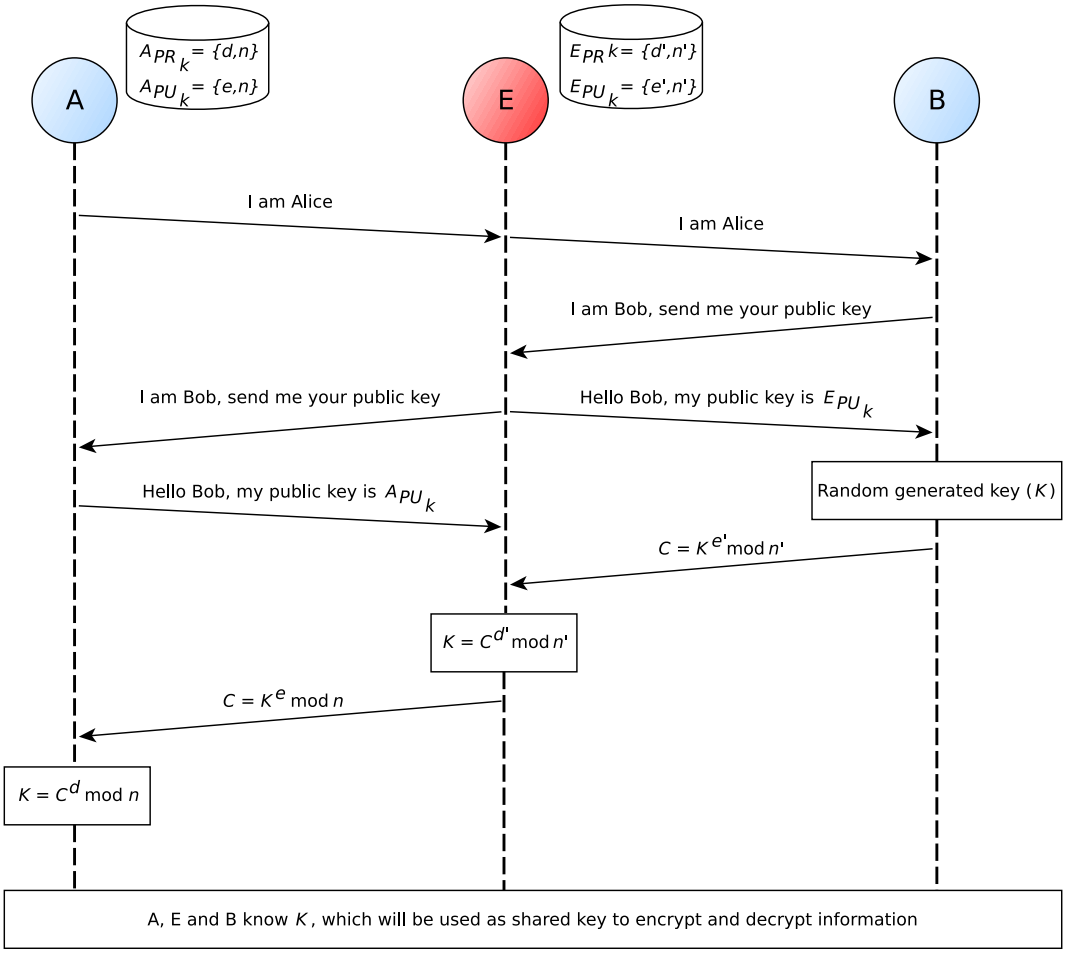


FIGURE 2.6: An MITM attack against RSA.

This attack can be overcome by using digital signatures or keyed hash functions. These mechanisms are used to provide proof of the origin of the message and its integrity in order to protect the message against forgery. Commonly, the digital signature is created by using the private key and the message. Therefore, as the private key is correlated to the public key, only the real owner of the public key can generate the signature of the message. The receiver of the message must know in advance the public key of the sender to properly verify the authenticity of the message and the signature, otherwise a different type of MITM attack can be perpetrated. This does not solve the

problem of associating a specific key to a specific individual, however. All it confirms is that the message is signed by that specific entity in the network.

This problem can be overcome by using trusted third parties such as Certification Authorities (CAs). The role of CAs is to provide digital signed certificates in order to associate the identity of an entity to its public key. For each entity to validate the public key with the digital signed certificate, the public key of the CA must be known, accepted and trusted. This will reduce the number of known public keys to be stored. A node sends its public key with the certificate and the receiver uses the CA public key to verify its authenticity.

These encryption mechanisms present some issues. First, they may require dedicated processors for encrypting and decrypting information and are therefore time- and power-consuming. Second, they can lead to attacks to retrieve the private key for decrypting the information. These performance issues are compounded in embedded systems at the base of IoT devices. Authors in [166–169] proposed a solution to these issues. This solution uses a Physical Unclonable Function (PUF) that is a challenge-response mechanism. In this, a specific challenge is mapped with a unique and specific response dependent on the physical material. The same challenge given in input to two different materials will produce to a different and unique response from each of them. The uniqueness is based on physical variabilities in the hardware components introduced during the manufacturing process. This makes the behaviour of each component unique and difficult to predict, leading to unclonable results. A PUF is a low-power, high-speed and low-cost solution. This can be used as a cryptographic mechanism in which the public key is called Public PUF (PPUF) and the private key is generated by the PUF itself.

For example, the PPUF (public key) of a node "A" is a set of well-known challenge-response pairs of its PUF. Node "B", which has the PPUF of "A" and would like to communicate with it, will challenge "A" by using a challenge in "A"'s PPUF. "A" will give the challenge in input to the PUF (private key), and its response will be sent back to "B". "B" can then verify "A"'s authenticity by comparing the obtained response with the response in "A"'s PPUF. If the responses match, "A" will be authenticated and marked as legitimate, otherwise the authentication will fail. It is clear that while this solution can be easily implemented, it requires that each node in the network has the PPUF of other nodes, otherwise MITM attacks can be performed using a similar pathway to that shown in Figure 2.6. A possible solution is to use trusted third parties with a database of PPUFs for each node in the network containing a large number of challenge-response pairs for each node. This does not solve the problem completely, however, as the trusted third parties must be trusted by using certificates.

Nodes using encryption only at the application layer can exchange information depending on application-based policies. If only this method of encryption is used, nodes can exchange information in unencrypted links, but this cannot hide the data traffic and

metadata. In particular, the source and destination addresses can be clearly identified, and can be used by malicious attackers to characterise the communication.

There are several limiting factors in relation to applying encryption techniques to the IoT and specifically with embedded objects. Network encryption can be used if nodes in the network are trustworthy, which is not always the case in IoT scenarios. For example, network encryption can be applied to controlled environments in which objects are owned by the same person/company, but its application is not feasible in public environments in which objects have different owners and link encryption is not feasible.

The application of digital signatures, certificates and/or PUF requires object to store known public keys/PPUFs of nodes and/or CAs' certificates. The certification process is time consuming and it is costly to certify the large number of objects in the IoT. Certificates also have an expiration date, which necessitates a constant update of objects' keys. This would lead to a large number of certificates, which would limit their applications into embedded objects with limited memory resources. Embedded objects using certificates must also have additional mechanisms to protect stored keys from malicious attackers. Moreover, a specific crypto-processor should be present in embedded objects with limited processing capabilities, to support encryption and decryption techniques when PUF are not used. A trusted third party is one point of failure, as if it is compromised, the entire network is also compromised. Digital signatures or certificates are not suitable for IoT-embedded things because of the large computational overheads and the low scalability of this method to billions of objects. Therefore, it is infeasible to apply these techniques to identify which objects belong to the IoT network, especially in uncontrolled and unmanaged environments.

Finally, authors in [170, 171] argued that authentication methods based on encryption only provide a binary authentication (i.e. accepted or not) and therefore are not suitable for distributed systems, plus they have scalability issues. Another issue with using these methods is that compromised nodes can actively be used to compromise the network. Selfish nodes could also take advantage of such mechanisms to disrupt the network. These authors highlight the importance of using trust instead of relying only on encryption techniques to give a full authorisation to the system.

2.4.2 Definition of Trust

Trust is a primary factor for securing future communications and it should be adopted as a standard. There is no unique definition of trust. Commonly, the word trust is associated with the Internet-based authentication method used to access secure servers. This method is based mainly on trust certificates provided by trusted CAs worldwide. Nevertheless, the fact that someone uses the correct trusted certificate does not actually mean that it is reliable a priori. In fact, an attacker can use the correct certificate, but at the same time can misbehave, perpetrating attacks against the system.

According to dictionary.com, "trust" is defined as the "reliance on the integrity, strength, ability, surety, etc., of a person or thing" [172]. The Merriam-Webster Dictionary defines trust as "assured reliance on the character, ability, strength, or truth of someone or something" [173]. This thesis adopts the definition of trust presented in [99]: "the belief that another party will behave according to a set of well-established rules and thus meet one's expectations". Trust can be also represented as a triad of three opinions belief, disbelief and uncertainty, which indicate the probability of trusting or distrusting an entity and the probability of doubt about whether to trust an entity [174]. Figure 2.7 shows the corresponding triads for information security and trust.

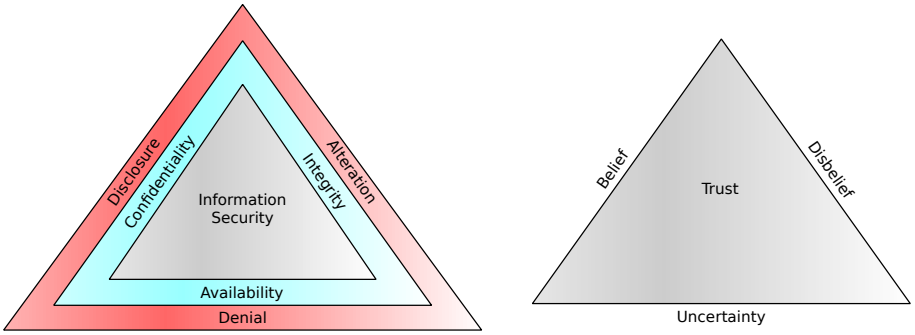


FIGURE 2.7: Summary of information security and trust triads, showing CIA and DAD triads for information security on the right and trust opinions triad on the left.

The definition of trust derives from the Human-to-Human (H2H) trust definition and is then applied to M2M communications. As in human societies, in a network there are several ways to create trusted relationships among entities. A trust relationship can be viewed as a set of characteristics about the relationship which is used to define how to trust another party. In a trust relationship there is an entity, called trustor or agent, which can create trust by providing a service to another entity, called the trustee. A trustee that is perceived as highly reliable by the trustor is called a trustworthy entity [175].

There are four ways to create trust relationships [175]:

i one-to-one: a relationship between two nodes,

ii one-to-many,

iii many-to-one: relationship between one node and a group of nodes and viceversa and

iv many-to-many: relationship between one group and another group of nodes.

There are also certain properties that a trust relationship has, such as reflexivity, subjectivity, asymmetry, transitivity, context relativity, measurability, uncertainty, dynamism and time-ageing [176–180]. Trust is primarily reflexive, as each entity trusts itself. Subjectivity refers to the fact that different parties can have different trust opinions about the same context. Asymmetry describes the scenario in which a node A trusts another node B, but this does not mean that B automatically trusts A. Transitivity is

when a node A trusts B and B trusts another node C, but because trust typically is not transitive, this does not necessarily mean that A trusts C. Typically a node A trusts another node B within a specific context or related to a specific environment; this is called context relativity. Measurability refers to the ability to specify the degree of trust one node has in another based on its assessment of the honesty, competence and dependability of the other node. When a node is not sure whether to trust another node, there is an uncertainty area that prevents the trustor from trusting or distrusting the trustee. Trust is dynamic and strictly dependent on the timing factor. In fact, a node A can trust another node B to a certain degree at a specific time, but this degree can change to uncertainty or distrust at another specific time. Finally, the trust is time-ageing, as its value decreases with the passage of time.

Trust relationships can also be defined using mathematical notations [181–183]. A relationship between an entity A and an entity B ($A \rightarrow B$) can be defined as $T_{A,B}$ with $T \in [0, 1]$. The properties of trust relationships can therefore be defined as follows:

- Reflexivity: $\forall A | G(T_{A,A} = 1)$ where Gx means always x;
- Subjectivity: $\exists T_A^{\alpha}, T_B^{\alpha} | \left(\exists T_A^{\alpha} \nRightarrow \exists T_B^{\alpha}\right)$ where α is the context in which parties are creating their trust opinions;
- Asymmetry: $\exists A, B | \left(\exists T_{A,B} \nRightarrow \exists T_{B,A}\right)$;
- Transitivity: $\exists T_{A,B}, T_{B,C} \wedge \nexists T_{A,C} \cup \exists P\left(A, B, C\right)$ where $P\left(A, B, C\right)$ is the trust path defined by the trust relationship between A and C through B. $T_{A,C} = R_B \cdot T_{A,B}$ if there is only one B, otherwise $T_{A,C} = \frac{1}{n} \sum_{n}^{i=1} R_{B_i} \cdot T_{A,B_i}$, where R_B is B recommendation trust value of C. These are valid only if B has some knowledge about C ($T_{B,C} \neq 0$) and A has some knowledge about B ($R_B \neq 0$);
- Context relativity: $\exists T_{A,B}^{c_i} \nRightarrow \exists T_{A,B}^{c_j}$ where c_i is the context i, c_j is the context j and $i \neq j$;
- Dynamic: $\left(T_{A,B}\right)_{new} \lesseqgtr \left(T_{A,B}\right)_{old}, \mu \lesseqgtr \left(T_{A,B}\right)_{old}$ where μ is the latest trust level;
- Time-ageing: $\forall c_i \left(\left(T_{A,B}^{c_i}\right)_t > \left(T_{A,B}^{c_i}\right)_{t+\Delta t}\right)$ where c_i is the context i and $i = 1, ..., k$.

2.4.3 Trust Management Frameworks

In order to collect information and to create and maintain trust relationships, a Trust Management Framework (TMF) is adopted. Blaze *et al.* [184] defined a TMF as a system based on a unified mechanism in which policies, credentials and trust relationships can be used in an algorithm to manage the security of the network. Jøsang *et al.* [185] used a more elaborate definition of TMF. Their TMF creates systems and methods that allow to entities in the network to decide and assess other parties on a specific critical subject. This also permits parties and their owners to specify their own reliability and that of

their systems. At the basis of a TMF there is a trust computation, through which trust is mathematically evaluated in a way that can be used by machines in the network. Trust computations can be subdivided into five main parts, as described below.

Trust Composition

At the core of the trust computation, there are several components that can be used to create trust, usually called trust metrics. A trust metric is specific information gathered from the observation of the behaviours of entities that is used to determine how one entity can trust another entity. An example of a trust metric is the Quality of Service (QoS) trust, which refers to the belief that an entity will deliver a service to a specified standard. In the IoT, authors in [186–192] measures the QoS trust using competence, cooperativeness, reliability, end-to-end packet forwarding, packet delivery ratio etc. Building on this approach, authors in [188–193] introduced the Social Trust (ST) metric in the SIoT.

The ST includes:

- Friendship: representing the degree of intimacy;
- Honesty: indicating if an entity is honest;
- Cooperativeness: denoting if an entity is socially cooperative;
- Connectivity: specifying the network connection with an entity;
- Social contact: representing proximity;
- Community of Interest (CoI): in this scenario, the trustor and trustee are in the same social communities/groups or have similar capabilities, such as the same knowledge of and standards in relation to a specific subject.

Trust Propagation

There are two different ways to propagate trust in a TMF: centralised and decentralised. In the first, trust relationships among entities are maintained and provided by a central entity, whereas in the second, each entity creates and maintains trust information about other entities [194, 195].

Trust Aggregation

In a TMF, trust aggregation is the method of combining trust information, which is collected in three ways: (i) direct observation, (ii) indirect observation or (iii) through recommendations. Direct trust is established by an entity through direct observation of another entity; indirect trust is established when one entity trusts another entity without directly observing it. In the case of three entities that can see each other, the

trustor can use the recommendations (direct observations) of other entities to trust a third entity.

A summary of these ways of collecting trust information in a network with the entity A as a trustor is shown in Figure 2.8. In this figure, A trusts the entity B on the basis of direct observations and C's recommendations, while A trusts D on the basis of indirect observations from B. Obviously, in order to use C's recommendations for trusting B, A must first evaluate the trustworthiness of C by using its direct observations and B's recommendations.

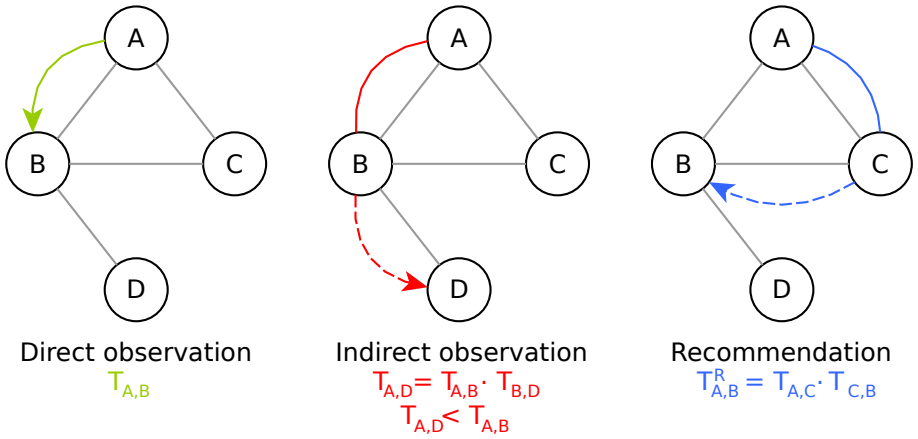


FIGURE 2.8: Examples of trust aggregation with the node A as a trustor.

There are different approaches presented in the literature to aggregating trust, including weighted sum, evidence theory, probabilistic estimation, information theory, game theory, fuzzy logic and grey theory [196–198]. As is evident from the discussion thus far, trust aggregation is strictly dependent on the network topology. In fact, depending on the network topology, some ways to collect information may not be available. For example, in a linear topology, a trustor will be unable to use recommendations as there will be no common neighbour entities.

Trust Update

The trust value of an entity can be updated in two different ways: event-driven and time-driven. In the first method, the trust value is updated after a specific event occurs, whereas in the second method, the value changes depending on the time. Usually in a time-driven TMF, recent information about an entity is more important than past information [199].

Trust Formation

Trust formation is the last step in evaluating trust in a relationship. Two aspects are described in the literature; single-trust and multi-trust. In a single-trust TMF, a unique trust value is used to create trust relationships, e.g. only QoS trust. In a multi-trust

TMF, multiple trust values are used simultaneously for creating trust relationships [199]. In this case, these trust values can be used in three ways:

- One-by-one: each trust value is compared with a threshold value depending on the application;
- Weighted sum: all trust values are summed by applying a weight factor to each one depending on its importance;
- Scaled trust: the trust values are scaled from the most to the least important property.

2.4.4 Attacks against Trust Management Frameworks

As with every networking-based system, TMFs may be vulnerable to several attacks, called misbehaviour attacks, that can be carried out by malicious entities [186, 189, 190, 200]:

- Selfish misbehaviour attack: the entity preserves its resources by claiming to offer some services without really providing them;
- On-Off Attack (OOA): an attacker behaves alternately well and badly for a period of time;
- Selective behaviour attack: an entity can behave well when providing some services and badly when providing others;
- Selective misbehaviour attack: the malevolent entity behaves well towards some important entities and bad towards other, less important entities;
- Self-Promoting Attack (SPA): an attacker promotes itself by projecting a good reputation in order to be used as service provider and then may provide bad services or may block service requests;
- Bad-Mouthing Attack (BMA): a malicious entity provides bad recommendations of other, well-trusted entities in order to reduce their trustworthiness;
- Ballot-Stuffing Attack (BSA) or Good-Mouthing Attack (GMA): the attacker increases the reputation of malicious entities in order to augment their trustworthiness;
- Opportunistic Service Attack (OSA): an attacker begins to provide good services to improve its reputation opportunistically, especially if its reputation is low;
- Conflicting behaviour attack: an entity behaves well with a group of entities and badly with another group;

- Sybil attack: the malicious entity creates multiple false identities and use them to manipulate the reputation of other entities;
- Newcomer or whitewashing attack: the attacker leaves the network and re-joins with a new identity in order to reset its reputation;
- Masquerade attack: an entity changes its identity to that of another entity (impersonation theft) and then uses its reputation (reputation theft) to behave badly.

2.4.5 Trust Models for IoT

Below, the main TMFs for the IoT proposed in the literature are listed. These are subdivided by following the trust propagation:

- Centralised. Saied *et al.* [186] proposed a TMF based on QoS and context-aware information such as service type and device capabilities (processing power, memory and battery level) for the trust composition. The central trust manager stores all the reputation reports sent by entities after a service is provided. When an entity requests a specific service, the central trust manager gives information about those trusted entities that can provide it;
- Distributed. D. Chen *et al.* [187], Z. Chen *et al.* [188], I-R Chen *et al.* and Bao *et al.* [189–192] proposed different approaches to distributed TMFs. Authors in [187] use QoS elements, such as end-to-end packet forwarding ratio, energy consumption and packet delivery ratio, as the trust metric, while authors in [188–192] use both QoS and ST metrics and relationships between IoT devices;
- Centralised and distributed. Nitti *et al.* [193] proposed a TMF based on QoS and ST as trust metrics.

Authors in [187] proposed the first TMF for IoT which adopted a fuzzy reputation mechanism to evaluate trust in the Wireless Sensor Network (WSN) of IoT systems. This is based on enforcing cooperation among entities and by checking their behaviours. They suggest that in IoT there will be smart nodes with resource constraints in terms of hardware, computation and power capabilities and a limited range of communication for wireless nodes. For wireless nodes specifically, they underscored the need to study mobility, dynamic topologies and limited physical security. Bao *et al.* [201] kept working in this direction, but they adopted a hierarchical TMF based on QoS in WSN. Consequently, in [189], authors suggested the first work in TMF for SIoT in which the trustworthiness of users is used to assist the service composition between objects. This is based on QoS and ST metrics. The same group of [189] adopted an enhanced TMF which considered mobility and evolving of environment conditions in the network such as increasing of nodes and attackers [190]. Moreover, they began to consider CoI of SIoT systems, and authors in [191] proposed the first TMF for this paradigm. In [192], authors considered a user-centric solution based on the SIoT paradigm, in which users

are the trustors and the devices owned by other users are the trustees. All the solutions proposed until [186] are based only on distributed TMF. In fact, authors of this last work proposed the first centralised TMF for IoT based on QoS as the trust metric. They underline the importance of considering the characteristics of the object during the trust evaluation, such as processing power, memory and battery capabilities [186]. Nitti *et al.* [193] suggested the first solution in which both centralised and distributed TMFs are used and applied to SIoT. In this solution, each node maintains the trust value of other objects. At the same time, a central Pre-Trusted Object (PTO) stores feedbacks from each node in the network and these are used as recommendations by objects in the network. Finally, authors in [188] proposed another TMF based on SIoT. They highlighted the necessity of using trust values obtained from the object's capabilities, e.g. a sensor will have a lower trust value than a laptop etc. A summary of proposed TMFs for IoT is shown in Table 2.3.

2.5 Limitations of Current Solutions

The works discussed in the previous section represent an important starting point in trust management for the IoT, however there are some issues that limit their applicability. First, a major assumption is made in [187, 190, 201]. In these works, each node in promiscuous mode in the network can overhear the traffic of its neighbours and therefore can understand if they are actively forwarding the packets. The disadvantage of detecting suspicious activities by snooping or overhearing a neighbour's transmission is that the nodes should keep track of every packet snooped to make sure that it was not modified during the transit. This is an additional load, especially for battery-powered devices. Moreover, node movements and node crashes will have a detrimental effect on the system by affecting the flow of traffic, and thereby leading to false alarms. This results in innocent nodes being mistakenly detected as intruders [202]. These works consider only a specific IoT environment consisting of wireless sensors with a QoS trust metric only (packet forwarding/delivery ratio and power consumption).

A second issue is that in some works [189, 190], only static environments are considered and thus the results are not applicable to many IoT applications in which environmental conditions are evolving. Moreover, the scalability issue has not being addressed, which hinders the applicability of this research to large-scale IoT systems.

A further assumption is made in [201], in which a hierarchical trust management for WSNs is proposed. It is based on Sensor Nodes, Cluster Heads and Base Station (BS). The BS is a Cluster Head commander and it is assumed to be infallible with physical protection. However, this assumption is not realistic in real environments and if the BS is compromised, the entire system will be compromised as well.

In addition, objects in [188–191, 193] use internal storage to save trust values and information about other entities in the network. In these, past and new events are

TABLE 2.3: Summary of TMFs for the IoT.

Work	Type	Paradigm	Trust Metric	Attacks considered	Main contributions
[187]	Distributed	WSN in IoT	QoS trust	SPA	First work that considered IoT. Underlined: smart wireless nodes with constrained resources and limited range of communication, mobility and dynamic topologies
[189]	Distributed	SIoT	QoS and ST	SPA, BMA, BSA	First work that considered the SIoT paradigm
[190]	Distributed	SIoT	QoS and ST	SPA, BMA, BSA	Consideration of mobility and evolving environmental conditions (increasing of nodes and attackers)
[191]	Distributed	CoI in SIoT	QoS and ST	SPA, BMA, BSA, OSA	First work that considered CoI of SIoT
[186]	Centralised	IoT	QoS trust	SPA. BMA and BSA only with some recommenders	First work to adopt a centralised TMF and used things characteristics during the trust establishment process
[193]	Centralised and distributed	SIoT	QoS and ST	SPA, BMA, BSA, OSA	First work to adopt both centralised and distributed TMFs
[188]	Distributed	SIoT	QoS and ST	SPA, BMA, BSA	Highlight the need to consider the characteristics of things

stored. If the network increases rapidly, this can be problematic for embedded devices with constrained resources, e.g. limited memory storage.

Another work that is based on two major assumptions is proposed by authors in [186]. First, during network initialization, all entities are assumed to be trusted and behaving properly. Second, it is assumed that entities cannot dynamically join and leave the network, in other words, that the topology is fixed. These assumptions are not realistic in an actual IoT environment in which network formations are dynamic and entities can join and leave the network at any time. Moreover, a centralised TMF is proposed, which creates a unique point of failure. This means that if the central entity is subverted, the entire network will be compromised.

The scope in [188] is to provide a QoS to end-users who own customised and specialised devices. An entity that requests a service will broadcast its request in the network, leading to network congestion if there are a lot of malicious devices or requests at the same time.

An equally important issue is evident in [193], whose authors introduced PTOs (Pre-Trusted Objects) as part of the TMF. The role of PTOs is to maintain the trust values of the entities in the network. However, an attacker can obtain its trust value from a PTO and change its behaviours accordingly. Moreover, if an attacker successfully impersonates a PTO, this can create false trust value entries, enabling it to attack entities in the network.

Table 2.4 shows a summary of gaps, major assumptions and other issues in the TMFs proposed in the literature for the IoT. It may be observed that none of them consider how to protect the TMF from On-Off Attacks (OOAs). Moreover, they do not consider worst case scenarios with linear topologies and virtual and emulated systems that can pretend to be real devices with different capabilities.

Most of the TMFs proposed in the literature do not consider attacks against the network communication, but only those against the TMF itself. A lot of basic assumptions are made to develop new TMFs. However, it is not possible to apply these in real IoT systems based on M2M communications, as these may include resource-constrained devices and have different network topologies. Moreover, no work has been done thus far where different trust metrics are combined; for example, a trust metric may be effective and efficient in Wi-Fi networks, but it will fail when applied in ZigBee networks [203]. Finally, these works do not address whether to distrust forged objects such as fake OSs, fake capabilities, etc. This issue allows attackers to use these forged objects to gain the trust of the TMF and hence to attack other nodes.

2.6 Summary

In this chapter an introduction to IoT was presented by looking at its evolution in terms of Internet-connected machines and architectural reference models. An introduction to M2M communications was also presented in conjunction with architectural reference

TABLE 2.4: Summary of gaps, major assumptions and issues in the proposed TMFs for the IoT.

Work	Traffic over-hearing	Static environment and fixed topology	Unlimited storage resources	Network congestion issues	Use of PTO or BS	OOA undetected	Trust metrics combinations not provided	Distrust forged objects not provided
[187]	✓					✓	✓	✓
[189]		✓	✓		✓	✓	✓	✓
[190]	✓	✓	✓			✓	✓	✓
[191]			✓			✓	✓	✓
[186]		✓				✓	✓	✓
[193]			✓		✓	✓	✓	✓
[188]			✓	✓		✓	✓	✓

models for IoT/M2M. Subsequently, emerging trends in IoT were shown. An overview of information security and trust was presented specifically in relation to TMFs for IoT. The limitations of current proposals to secure the IoT were highlighted.

It is important to note that the already high number of Internet-connected things that will manage our lives is growing, and that these things must be secured. As can be seen, having no standard architectural reference model within which to build security functionalities is a major problem. This is compounded by the continuous evolution of the IoT in that these models are changing, mainly guided by the current needs and objectives.

However, some architectural reference models incorporate trust for securing the communication of things. Trust is used by things to identify misbehaving machines in the network that do not follow rules honestly and most importantly do not need human help. After defining which information can be used to obtain trust values from the behaviour of these machines, it will be easy to determine whether a node is trustworthy or not. By evaluating these trust values, a measurement of the trustworthiness, reliability, and capacity of individual entities can be calculated based on previous, direct and indirect observations, and recommendations about their behaviours.

The continuous evolution of the IoT without a unique standard for managing M2M communications makes the creation of a TMF a very challenging task. Machines that look to compromise trust can perform various attacks against the TMF. Therefore, a context-aware approach must be used that considers the characteristics and computational limitations of an IoT "thing". The smartness of a thing must be improved by adding the ability to create trust relationships with other things, and IoT mobile agents could be fundamental in this role.

As highlighted previously, there are currently no complete solutions that can be applied in the real IoT environment. Some works on TMFs for IoT discussed in this thesis are focused only on a specific IoT paradigm, the SIoT, which limits their applicability. In this new paradigm a scenario in which the owner of an object is a company is not considered, making social relationships more difficult to apply in business-to-business applications. Moreover, this paradigm relies on the owner of the object assigning the object to the correct class representing its features, but an attacker may insert false information in order to take some advantage from the system. Finally, the owner of the object must define what relationships are allowed with other objects owned by other humans, therefore trust relationships among owners must be considered. In this situation, whether the owner of the object trusts other owners will affect whether his/her object trusts other objects.

Another problem that we face today is the growth of proprietary objects. These will be more difficult to integrate into an IoT system. Security is, unfortunately, the last aspect that is currently considered when creating these objects, meaning that attacks against them are easy and constantly increasing.

In the next chapter a new threat against the IoT system and specifically against M2M devices is presented. This attack can be used to subvert a network and obtain private information. An initial method for detecting and protecting a machine from this type of attack is also proposed.

Chapter 3

A New Threat and a Novel Solution: Machine Emulation Detection Algorithm

In this chapter, a new threat against M2M communications in the IoT is presented. This threat is based on using virtualised and emulated embedded systems for attacking the network or part of it. As shown in Chapter 2 (Section 2.5), current TMFs are not able to distrust forged objects, therefore they cannot recognise if a machine is based on a virtualised or emulated system. This attack is then applied to the real-life scenarios presented in the introduction (Chapter 1), in order to highlight the system vulnerabilities and to show what could happen if this attack is not detected and prevented. Existing works on detecting virtual and emulated embedded systems are shown later in this chapter. It is shown that these cannot be used in IoT/M2M communications. In this context, a novel solution is presented and evaluated, called Machine Emulation Detection Algorithm (MEDA). This solution was published by Valerio Selis and Alan Marshall as 'MEDA: a Machine Emulation Detection Algorithm' in *Proceedings of the 12th International Conference on Security and Cryptography (SECRYPT 2015)* [6].

3.1 Threat Model

Currently, an IoT-embedded object can use two main types of communications to exchange information with other objects, the Internet and the end user:

- Machine-to-Machine (M2M) communication;
- Machine-to-Human (M2H) communication.

These can be subverted by an attacker in order to create two types of communications:

- Machine-to-Fake Machine (M2FM) communication, and vice versa;
- Machine-to-Fake Human (M2FH) communication, and vice versa, where a fake human is a machine that is able to mimic the behaviours of humans.

This thesis focuses on M2FM communications.

Currently, an IoT-embedded object communicates with other embedded objects in the network on the assumption that these objects are real embedded machines (REMs). Therefore, M2FM communications are not considered at all in current research works. Of the works present in the literature focused on integrating trust mechanisms in the IoT, some underlined the importance of trusting embedded systems in a different way. These works use the characteristics of an object to weight their trust values as part of the trust computation, as shown in Table 3.1 [186, 188, 193, 204].

TABLE 3.1: Assigned trust weight values of published works, according to the computational capabilities of machines.

Device	**Trust weight value**	
RFID	0.2 [193, 204]	Device resources in percentage (0 to 100%) [186] [1]
Sensor	0.2 [188, 193], 0.4 [204]	
Recorder	0.2 [188]	
Set-top box	0.4 [188], 0.6 [204], 0.8 [193]	
Smart video camera	0.4 [188], 0.6 [204]	
Smart gateway and terminal	0.6 [188]	
Smartphone and tablet	0.8 [188, 193, 204]	
Laptop	0.8 [188]	

However, a potential attacker may create M2FM communications by using virtualised and emulated embedded systems. Thus, multiple forged objects can be simply created with a powerful machine. Moreover, as Table 3.1 suggests, it will probably choose to forge smartphones, tablets and/or laptops, because of the high initial trust weight value assigned to these objects. This does not stop an attacker from also using REMs to create M2FM communications, but this is outside the scope of this thesis as existent TMFs can be used to detect them.

Figure 3.1 shows an example of a threat model in which multiple virtualised or emulated embedded systems are used to compromise the M2M communication. In this case, the real embedded object (A) will presume that it is creating an M2M communication with other "real" embedded objects (B, C and D). These allow access to the Internet via another real embedded object (E), as shown in Figure 3.1(a). The attacker uses a powerful machine to forge "B", "C" and "D" in order to create an M2FM communication, as in Figure 3.1(b). This may lead to security issues in the communication, especially for the data transmitted by "A" to the Internet via the attacker(s). After the attacker successfully established the M2FM communication, it may begin to collect "A"'s data and to send to "A" false recommendations about "E".

[1] N.B. [186] uses the percentage of device resources for creating trust relationships rather than a specific trust weight value depending on the device type.

Figure 3.1 shows only an example of this threat, however; in fact, an attacker can create a large number of forged embedded objects with which to attack the TMF. By doing this, multiple forged embedded objects are able to attack the TMF and then the IoT network simultaneously by launching several misbehaviour attacks, as highlighted in Chapter 2 (Subsection 2.4.4). These can also send false information to "A" and/or poison the information it sends or receives. For example, nodes "B" and "C" can increase their reputations by sending good recommendations about each other to "A", therefore making "A" more prone to trust them.

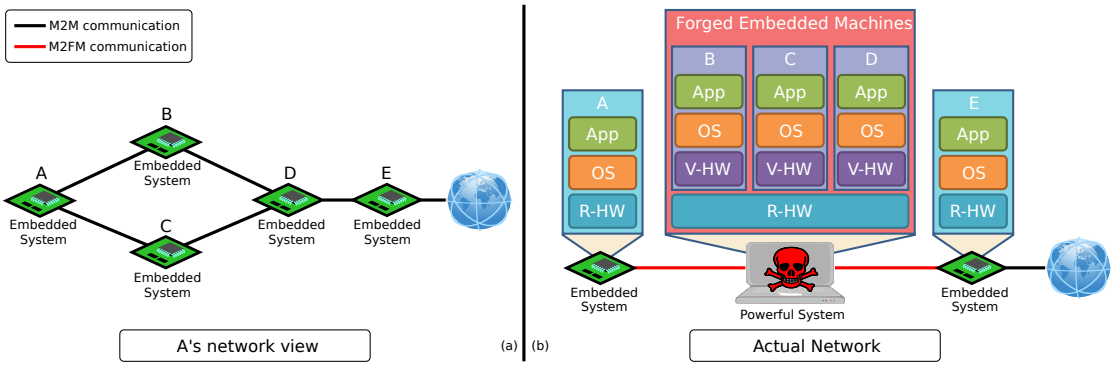


FIGURE 3.1: Threat model with a representation of multiple forged embedded machines attacking the IoT in order to create M2FM communications. (a) A's view of the network, from which there are apparently no issues. (b) The actual network topology, including the attacker forging B, C and D.

3.2 Real-Life Scenarios: Worst Cases

At the moment, there are several virtual and/or emulated embedded systems (VESs) available, such as VMware Player and ESXi, VirtualBox, VirtualPC, QEMU etc., that can be used by an attacker to create multiple forged objects. As shown in Table 1.2 in Chapter 1, there are 169,671 servers on-line that use VMware ESXi. This virtualisation system is a hypervisor that runs Virtual Machines (VMs) directly on the hardware without relying on an OS. Each VMware ESXi server can run simultaneously a maximum of 1,016 VMs by providing enhanced computational capabilities in respect of VMs running on top of an OS [205]. A possible attacker that uses all the VMware ESXi servers available on-line may be able to create around 172 million forged embedded objects. This number indicates a very high risk for the IoT infrastructure in the event that these servers are compromised and used as an IoT botnet. An attacker using this large number of forged embedded objects can launch a distributed attack simultaneously in several places. For example, in the ITS application an attacker could disrupt the road safety of an entire city, county or nation, with catastrophic economic impacts and loss of human life. This could be achieved simply by creating forged vehicles by feeding information into the ITS and ultimately causing chaos, i.e. real accidents, busy roads etc. Currently, it is not possible to detect such an attack, leading to a high security risk for the IoT system, as will be envisioned in the follow subsections.

It also must be acknowledged that some attackers are not driven by a desire to profit from the attack, but only by the fact that they can do it, in other words, by the pursuit of personal gratification, increasing the security risk to such a powerful system.

This powerful attack is strictly dependent on the IoT scenario and on the information the attacker can gather from the network. Moreover, the number of forged embedded systems can change according to the scenario and this information. There are two main categories of IoT scenarios in which an attack can occur: open or public networks and closed or private networks. These are presented in the following subsections and applied to real-life scenarios.

3.2.1 Open Networks

In this category, IoT objects can freely join and leave the network. Network encryptions are not used to encrypt/decrypt the communication link leading to all objects in order to see the traffic. Objects are also difficult to identify as a large number of them can be present in different areas at the same time and they can move across areas. In these real-life scenarios an attacker can easily create and masquerade multiple forged embedded machines in order to attack the IoT system. In fact, it would be very difficult to identify this kind of attack, especially in crowded areas. For example, real IoT applications that can be easily attacked are the ITS and, in specific circumstances, the IHS.

Intelligent Transportation Systems

In subsection 1.2.1, three possible future scenarios involving the ITS were presented, using a vehicle as an example: in the first, it is used to improve navigation, in the second, to improve road safety and in the third to improve passenger entertainment.

In the first scenario, vehicles communicate with the ITS and other vehicles in order to calculate the fastest route. In this scenario, an attacker that aims to disrupt the traffic can create a large number of forged vehicles in order to send fake information, such as detail of an accident that has not happened, false traffic information etc., to both the ITS and other vehicles. In response, the ITS will suggest alternative routes to other vehicles, adjust traffic signals to reduce the traffic on what appears to be the busiest road where in fact the attacker is, etc. In the meantime, other vehicles will avoid using the attacker's route, because it appears to be the worst and slowest route to be used to reach their destination.

In the second scenario, the ITS receives information from vehicles and humans-owned objects and sends recommendations to users for improving their road safety. In this scenario, an attacker that aims to disrupt the safety of users can forge a large number of vehicles and objects to send fake information to the ITS. These forged objects can send fake positions to the ITS and automatically it will alert vehicles to stop or decrease their speed because a human is crossing the street. Moreover, forged vehicles can send fake accident information to the ITS that will react by alerting emergency services.

Furthermore, in case of a real accident, forged vehicles can send false information to the ITS in order to lower the reputation of the involved vehicles, thereby preventing and therefore avoiding emergency services from being alerted.

In the final scenario, vehicles communicate with each other in order to improve the Internet connectivity for downloading entertainment content such as film, music etc, and adjusting audio and luminosity levels in the vehicle. An attacker can use forged vehicles to act as an intermediate node between the Internet and the victim vehicle. It can then inject a varying amount of fake information depending on the speed of the vehicle, in order to misrepresent what the users in the vehicle under attack are watching or listening to. In the case of music streaming, the attacker would need to inject only a few inappropriate audio samples at a high volume to distract the user from driving, possibly causing an accident.

Intelligent Healthcare Systems

In subsection 1.2.2, future scenarios involving the IHS were presented in which sensible human information was used to improve humans' health. Two main possible future scenarios can be attached: alerting first aid services and spreading new diseases using smart body sensors. In these scenarios, an attacker forges smart body sensors and send fake information to the IHS, such as false human temperature, false heart rate and false position, with the aim of alerting the nearest ambulance for provide assistance to a subject that does not require it. Moreover, by using a large number of forged smart body sensors, the attacker could replicate the severe symptoms caused by a pathogen in the population of certain areas. When received by the IHS, this information will be interpreted as the spreading of an infectious disease in these areas and competent authorities will be alerted, causing an escalation of other issues such as alerting residents in these areas to stay at home, alerting hospitals to get ready for a crisis, causing panic in the population etc. These attacks can be perpetrated only if the forged body sensors are connected to the open IoT network. In fact, real body sensors of patients that are under critical conditions are likely to be connected to closed home networks in order to access the IoT core.

3.2.2 Closed Networks

In this category, IoT objects are normally registered in a controlled central system. In this, encryption mechanisms may be applied, and the number of nodes is mostly known. Therefore, in order to communicate with other nodes, the attacker would need to attack encryption mechanisms. As the network is mostly known by the administrator, a small number of forged embedded machines should be used for launching the attack. Moreover, it may be necessary to attack the network encryption in order to encrypt/decrypt the communication link. It may also be necessary to tamper with objects in order to retrieve encryption keys for encrypting/decrypting the communication at the application layer. Depending on the type of environment, a large number of unknown (forged) embedded

machines appearing in the network may trigger an alarm, warning that the network is under attack. A real IoT application with a closed network that can be attacked is the IBS.

Intelligent Building Systems

In subsection 1.2.3, future scenarios involving the IBS were presented that included improving the energy consumption, building occupancy and entertainment of humans and protecting them in case of fire or other emergencies.

In the first and the second scenarios, the location of users and smart building sensors can be used, for example, to reduce the energy consumption in unused building areas. In this situation, after breaking the encryption, an attacker can use forged human devices to send fake positions or forged smart movement sensors to provide false information to the IBS. The IBS will adjust the room conditions in terms of luminosity, temperature etc., on the assumption that there are people in the room. Moreover, the attacker can also replicate a busy area in the building in order to concentrate people in specific areas. The IBS will then alert all people to avoid that area if possible by suggesting alternative areas.

In the third scenario, after breaking the encryption, an attacker uses forged smart building sensors and human devices to send false requests for accessing to the entertainment system in the building managed by the IBS. The IBS will then turn on the entertainment system, adjust luminosity level in the room etc., although no person is present to use the entertainment system.

In another scenario, after breaking the encryption, an attacker uses forged smart building sensors to indicate the presence of a fire to the IBS. This will alert the nearest fire station and will initiate the evacuation procedure.

3.3 Virtualisation and Emulation Detection

It is very clear now that the detection of forged machines and therefore virtualised and emulated embedded systems in the IoT must be a priority for securing information exchanged by IoT/M2M devices and the IoT core. To date, the detection mechanisms available have been mostly applied to x86/x64 architectures. Only a few works have focused on the detection of Android-based emulated environments. Considering the heterogeneity of the IoT, it is obvious that the available detection methods cannot be used to detect forged embedded devices. Moreover, most of them rely on the fact that the detection is made by an application launched manually in the system and that most of the time this requires a high level of privilege to be executed. Finally, none of them was applied to M2M communications and IoT scenarios in which the trustor is not in control of the machines running VESs. In the following subsections, a review of these detection methods is presented that highlights why they cannot be used in the IoT.

3.3.1 CPU and Memory Tests

This type of detection method gathers information from the memory and central processing unit (CPU) registers to detect VESs. This method was first introduced by Rutkowska in 2004 and it was named "The Red Pill", derived from the film "The Matrix" [206]. It was based on using the Store Interrupt Descriptor Table (SIDT) instruction in a x86 single-core CPU to access the Interrupt Descriptor Table (IDT) register. This register contains a unique value for each OS running in the system. The author discovered that the value obtained from the system under consideration differed in REMs and VESs. However, this is true only in a single-core machine, because there is only one IDT register per CPU core, therefore the VES must create an IDT register in another address. Obviously, this method cannot be applied in newer machines as these have a multi-core CPU.

Authors in [207] developed an automated method for generating random red pills by using CPU guided systems. This method was used for detecting emulators in x86 architectures such as QEMU and BOCHS. This method checks if a known instruction (red-pill) with a known CPU state returns a known value in the memory. If there are discrepancies when running this instruction in a system, it will submit a detection.

An enhanced method based on CPU and memory tests was proposed by authors in [208] that involved using all definitions in the IA-32 manual for generating red-pills, called the cardinal pill.

These detection mechanisms can fail to detect VESs that use real CPUs to execute instructions instead of virtualising or emulating them. Moreover, these tests were only applied to system architectures based on the Intel 32-bit architecture. Furthermore, in the IoT there are several different architectures, such as embedded x86, MIPS, ARM, PowerPC etc., and it is not reasonable to test all of them to find, if any, specific CPU instructions and memory information that can lead to the detection of VESs. For these reasons, CPU and memory tests cannot be used by IoT/M2M machines to detect VESs. Finally, in future, system-on-chip with customised architectures can be used to create IoT devices. Therefore, this detection method cannot be applied as it requires previous knowledge of the architecture, which would be impossible with customised architectures.

3.3.2 Architecture-based Timing Tests

In this method, specific CPU instructions are used to perform a time analysis. Authors in [209] and [210] showed that by timing the access to CPU control registers in x86/x64 architectures, such as CR0, CR2 and CR3, and the execution of a No Operation (NOP) instruction, it is possible to detect if the machine is an REM or a VES. These timing tests were performed in both REMs and VESs, and differences in time values lead to a detection of VES. However, as noted in the previous section, in the IoT there can be customised architectures, and as this detection method is architecture-dependent, it cannot be used by IoT devices for detecting VESs.

3.3.3 Remote Tests

A remote test involves collecting information from a machine in the network without relying on running a specific application inside the machine itself. The authors in [211] proposed a remote test based on a method created by [212] to remotely detect VESs. The method proposed uses the Transmission Control Protocol (TCP) timestamp option, defined in RFC 1323 [213], to determine the clock skew of the remote machine. The clock skew of REMs is then compared with the clock skew of the machine under test; discrepancies will indicate a VES. However, the TCP timestamp option is used only in Linux-based systems, therefore, machines running other OSs, such as Windows-based systems, cannot be evaluated. Moreover, authors in [214, 215] demonstrated that by using the "ntp" daemon application in Linux-based systems it is possible to adjust the TCP timestamp option value in order to remove the clock skew introduced by a virtual or emulated system. Usually, a machine uses the "ntp" application based on the Network Time Protocol (NTP) for synchronising its timestamp. This application retrieves the time from an Internet-based NTP server with a very accurate external clock. Authors in [216] demonstrated that it is possible not only to adjust the TCP timestamp option value, but also to mimic the clock skew of a specific machine. For these reasons, remote tests presented in the literature are not suitable for detecting VESs in the IoT.

3.3.4 Fingerprinting Tests

This is the most common and easy test used for detecting VESs and it is based on collecting specific "signatures" from the system, such as driver names, running applications, system registry keys, hardware IDs (i.e. CPU ID, MAC addresses etc.), system Application Programming Interfaces (APIs) access where available etc.

A fingerprinting method that uses the media access control (MAC) address of a machine to obtain the vendor name was proposed by authors in [209]. However, it can be easily faked, for example, by using an application such as MAC-Changer [217].

Further works improved on this solution by collecting information from system drivers used for specific hardware devices, OS registry keys or applications running [210, 211]. Another work shows that it is possible to use the APIs provided by a virtual machine to detect it [218].

There have only been a few studies related that use the fingerprinting test for the detection of embedded emulated environments, and these are focused on Android-based devices. A heuristic detection method was proposed by [219]. It is based on combining the Android API, Android system properties and the system hardware information and the detection takes around 20 minutes. An enhanced method was proposed by [220] which uses information about the CPU and graphical performances to detect the Android's Dalvik virtual machine.

However, these "signatures" can be easily faked in VES, especially with open source OSs. For example, I was able to modify the kernel of an embedded Linux OS in order to display fake CPU information that can be obtained from "/proc/cpuinfo", as shown in

Figure 3.2 in system type. This can also be done for every module, driver and application by modifying their information and recompiling them.

```
root@OpenWrt:/# cat /proc/cpuinfo
system type             : GOD!!
processor               : 0
cpu model               : MIPS 24Kc V0.0  FPU V0.0
BogoMIPS                : 1053.49
wait instruction        : yes
microsecond timers      : yes
tlb_entries             : 16
extra interrupt vector  : yes
hardware watchpoint     : yes, count: 1, address/irw mask: [0x0ff8]
ASEs implemented        : mips16
shadow register sets    : 1
kscratch registers      : 0
core                    : 0
VCED exceptions         : not available
VCEI exceptions         : not available

root@OpenWrt:/#
```

FIGURE 3.2: Faked CPU information in the OpenWRT embedded Linux system obtained from "/proc/cpuinfo".

3.4 Solution and Algorithm Design

In the previous section, existing methods for detecting VESs were presented and evaluated. As highlighted previously, these methods cannot be applied to the IoT system, leaving it unprotected when forged machines are used for perpetrating attacks. For this reason, a new approach is needed to enable M2M-embedded devices to detect forged systems.

VESs are normally used by developers, anti-virus companies and researchers as test environments for testing applications without a real embedded hardware and for studying malware behaviour. This is because the execution of a set of instructions in a VES should be the same as the execution in an REM. When a VES is used, it translates all instructions from the host architecture to the virtualised or emulated system, called the guest. For example, if the VES is running in a 64-bit architecture and virtualising an ARM embedded architecture, it must translate everything from the 64-bit to the ARM architecture. Each part of the embedded system must be translated, including CPU instructions, memory management, physical devices management etc., which requires time.

Authors in [221, 222] identified that VESs introduce various behaviours that cause differences in system resource availability, timing dependences and I/O device communication. These behaviours were also exploited by architecture-based timing tests. However, these tests are significantly limited by the fact that previous knowledge of the architecture is required to compare timing access to specific CPU registers and memory locations.

For these reasons, in this thesis, a new generalised and novel approach to detecting forged embedded machines is presented. This is based on the premise that the machine architecture should be treated as a black-box. The detection method uses the timing

behaviours of embedded machines. It can be subdivided into two parts: the characterisation of embedded machine behaviours and the machine emulation detection algorithm (MEDA), as presented in the following subsections.

The first part can be implemented securely by using a Mobile Agents Platform (MAP), which is a system based on IoT Mobile Agents (IoT MAs) with the ability to migrate from one device to other devices in the network in order to perform specific tasks. A MAP creates an opportunity to reduce the network load and latency, encapsulate different protocols, act asynchronously and autonomously, be dynamically adaptable, work in heterogeneous contexts, be robust and fault-tolerant, provide security mechanisms and work in several application scenarios with different interactions [223, 224].

The MAP architecture consists of several components according to the IEEE Foundation for Intelligent Physical Agents (FIPA) standard [225]:

- *Agent management* or *agency* manages mobile agents and controls their creations and operations within and across the MAP;
- *Agent communication* allows a mobile agent to communicate with its creator and the agent management;
- *Agent transport* is responsible for managing the transportation of information, including security mechanisms, across the MAP;
- *Agent security manager* is responsible for maintaining the security of the MAP;
- *Applications* provide the execution environment for a mobile agent and deal with different application scenarios in which there could be several types of interactions.

Figure 3.3 shows the detection model which employs IoT MA. Specifically, the agency in node "A" creates an IoT MA and sends it to node "B". The IoT MA runs the characterisation algorithm in node "B" and the results are sent back to the agency in node "A". "A" then runs MEDA to determine if "B" is a REM or VES. As trust is asymmetrical, the agency in node "B" will perform the same steps to evaluate the trust of node "A".

For security reasons, the agency creates a secure environment, giving IoT mobile agents the opportunity to perform only the characterisation test. This can be achieved by implementing resources access control with limited privileges and limiting the execution time [226]. A trusted third party can create an IoT MA with the capability to generate random numbers and publicly release its hash sum value using, for example, Secure Hash Algorithms (SHAs). This can then be included in agencies and used to verify the authenticity of the IoT MA before its execution. In fact, if the IoT MA is modified by an attacker, this will result in a different hash value.

Two steps are performed to create a secure connection between the agency that created the IoT MA and the IoT MA: (i) the agency generates a public/private key pair

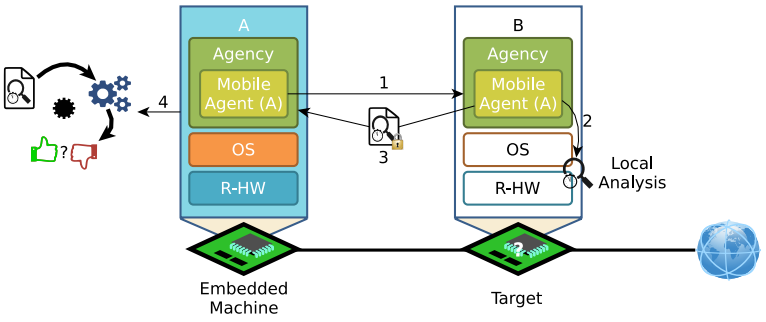


FIGURE 3.3: Representation of the detection model. The agency in "A" sends the IoT MA to "B" (1). IoT MA runs the characterisation algorithm locally in "B" (2) and then sends the results back to the agency in "A" (3). "A" performs the final detection (4).

and (ii) the IoT MA carries the generated public key of the agency. This public key will be used by the IoT MA to generate a shared key for encrypting and decrypting the characterisation results by using, for example, RSA, as discussed in Subsection 2.4.1. The steps to secure the communication are summarised in Figure 3.4. In order to be effective, it is assumed that the MAP satisfies two basic requirements: (i) the target agency allows the IoT MA to run the characterisation and (ii) the IoT MA must have a connection with the originator agency. If these conditions are not satisfied, the trustor will not trust the trustee and will stop the communication.

In this scenario, the communication between the agency and the IoT MA cannot be eavesdropped by using MITM attacks as the IoT MA knows the agency's public key. Security policies should be applied to protect the system against DoS attacks and avoid battery draining (if present) caused by multiple IoT MAs running the characterisation algorithm. For example, an IoT-embedded machine can decide not to allow an IoT MA to run if, for example, system resources (battery, CPU level etc.) are below a specific threshold or if there are several IoT MAs running at the same time.

3.4.1 Characterisation Algorithm

The characterisation algorithm extracts timing behaviours from embedded machines. It is based on Hypothesis 1 and represented in Figure 3.5.

Hypothesis 1. Every VES introduces delays caused by translating and executing instructions from the host to the guest architecture. This causes differences in timing behaviours between a VES and an REM.

The characterisation algorithm is based on pinging the localhost (127.0.0.1)[2] in the system under consideration many times (e.g. 1000) in order to characterise behaviours related to the ping response time (P.), timestamp values (T.) and CPU usage (C.) for each ping, as shown in Figure 3.6. Timestamp values are used as control information;

[2]This is the default address for localhost for the IPv4 protocol [https://tools.ietf.org/html/rfc5735].

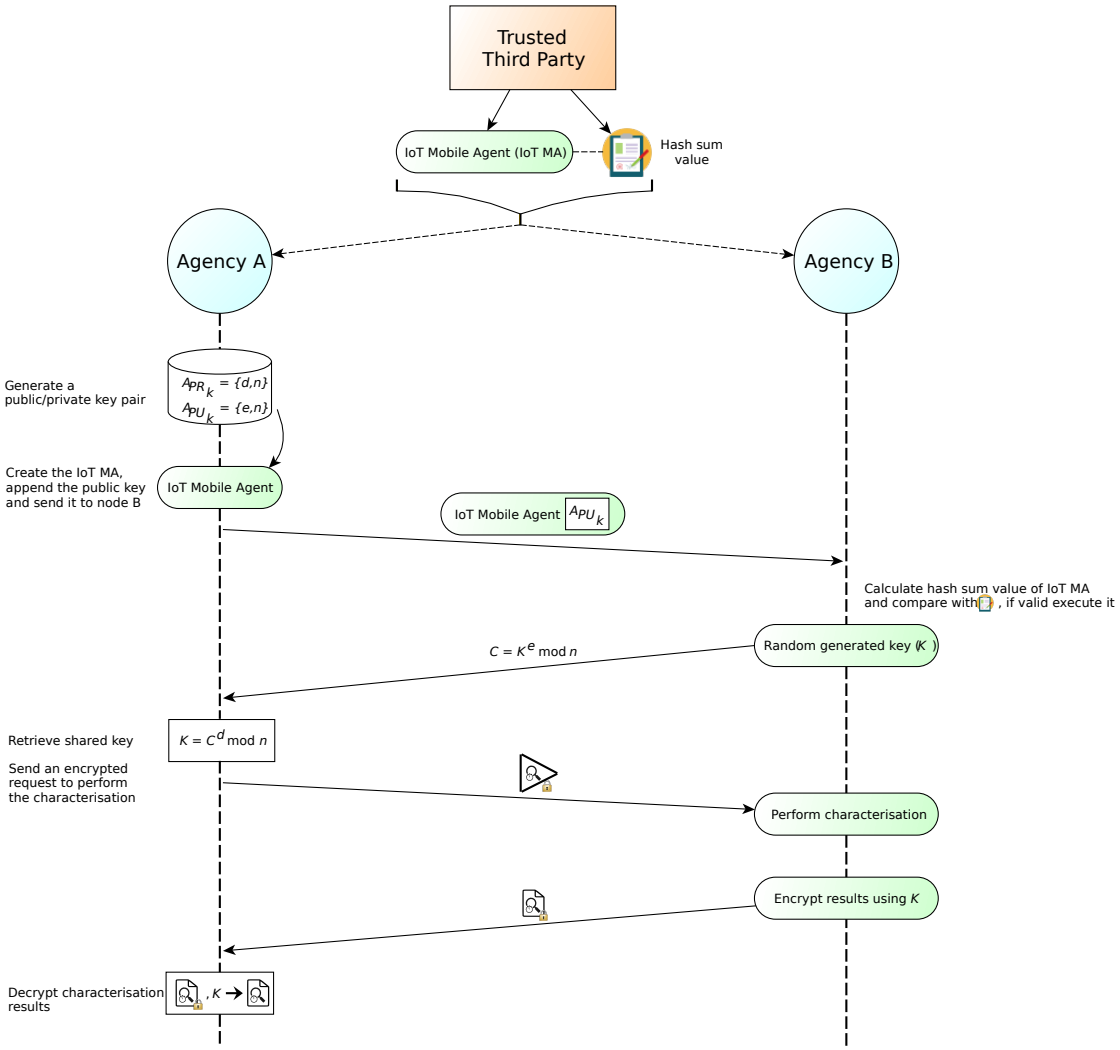


FIGURE 3.4: Representation of security mechanisms in MAP for characterising IoT-embedded machines.

if an attacker tries to fake the ping response time, the algorithm will rely on changes in timestamp values and vice versa. In fact, in order to fake both or one of these values, an attacker must implement new functionalities. These functionalities will be executed after they are translated from the host architecture to the virtualised or emulated system. This translation will require some time, which will increase the time between two pings (T.) and the characterisation time. These timing discrepancies can be used by the trustee to detect that the attacker is trying to fake the characterisation result.

The ping response time was obtained by using the "ping" command, the timestamp values were obtained by using the "date" command, and the CPU usage was determined by extracting information from "/proc/stat" file or the "iostat" command depending on the OS used. However, CPU usage is mentioned only for reference and is not used for detecting forged embedded machines as this could be easily faked by running several concurrent applications.

The "ping" command is used for several reasons. Firstly, it is available in almost

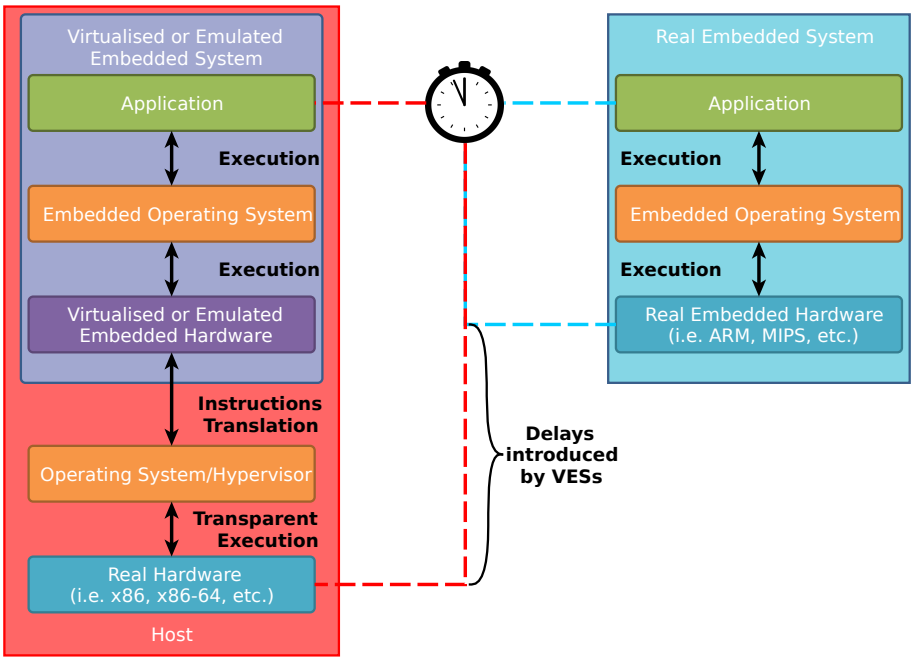


FIGURE 3.5: Hypothesis behind the characterisation algorithm concerning the difference in time behaviours for translating and executing instruction in REMs and VESs.

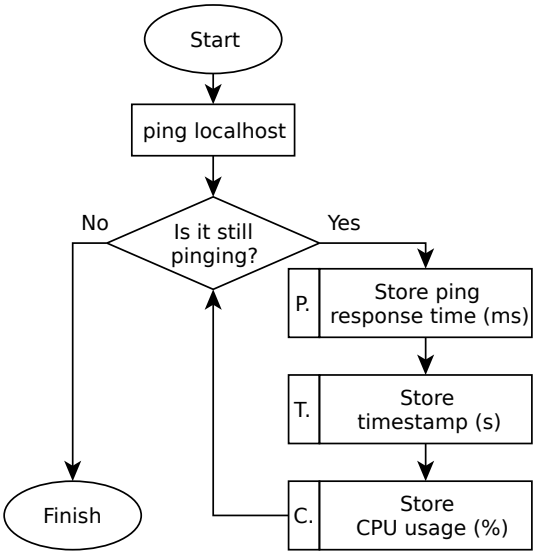


FIGURE 3.6: Characterisation algorithm flowchart. The ping command is used locally and information from ping response time (P.), timestamp (T.) and CPU usage (C.) is stored.

all systems with Internet connectivity, including IoT/M2M devices. It is obvious that it cannot be applied to IoT-embedded devices that do not support the ping command. For these devices a different method for extracting timing behaviour is required. Secondly, it is used to obtain precise timing information that is strictly related to the networking stack used by IoT/M2M devices. Another reason is that it will be difficult for attackers to fake the timing information provided by pinging locally, as it uses network sockets for managing the ping packets. These sockets reside in the kernel space of an OS and only its modification may allow an attacker to fake the timing information. Attackers

that try to fake the ping characterisation results must actively modify part of the kernel of a VES by checking when a network socket is created and handle it every time. This task can increase the overall delay and decrease system performance, especially in the presence of embedded environments. Furthermore, modifying the kernel is not always possible, especially if the system is not open source.

In fact, when the system in a VES receives ping packets, it will automatically create ping request and response packets, and manage these in both kernel and user space. As described previously, these operations require translating every instruction from the host to the guest architecture. Therefore, it is unlikely that the behaviours will be the same in VESs and REMs.

Moreover, if the attacker tries to slow down the VES response time in order to fake the characterisation, the agency that requests the characterisation can use its local time to detect this. This can be achieved by measuring the difference between the request (T_{Req}) and the response (T_{Resp}) of the characterisation as follows:

$$T_{Resp} - T_{Req} \approx T_{Ch} + \Delta T_{Ch} + \Theta(\Delta T_{RTT}) \tag{3.1}$$

where T_{Ch} is the time required for the characterisation, ΔT_{Ch} is a very small amount of time (around 2 seconds) in which the IoT mobile agent receives the request, launches the characterisation, receives the results and sends them back to the trustor IoT agency, and $\Theta(\Delta T_{RTT})$ is the estimated round-trip times after T_{Ch} plus ΔT_{Ch} seconds are elapsed. For example, if the characterisation is supposed to take 3 minutes, the difference should not be longer than 3 minutes plus network communication delays and the trustee IoT mobile agent delays, but not less than that.

For each target, eight tests were performed in which the ping command was tuned with different options and stressing the CPU (whereby the CPU usage is maintained around 100%), as shown in Table 3.2.

TABLE 3.2: List of ping characterisation tests performed.

Test#	**Ping option**	**CPU stress**
1	-c 1000 -i 0.2	No
2	-c 1000 -i 0.2	Yes
3	-c 1000	No
4	-c 1000	Yes
5	-c 1000 -s 20000	No
6	-c 1000 -s 20000	Yes
7	-c 1000 -s 20000 -i 0.2	No
8	-c 1000 -s 20000 -i 0.2	Yes
-c: stop after sending n ping packets -i: wait n seconds between sending each packet -s: specifies the number of data bytes to be sent		

In order to stress the CPU, the "dd" command was employed using the random device as input and the NULL device as output (dd if=/dev/urandom of=/dev/null). Multiple

instances of the "dd" command were executed in embedded machines with multi-core CPU. This stressed not only the CPU but also the kernel, because in Linux, random and NULL devices are managed in the kernel space. During the simulations, systems were performing SSH connections with a server which was used to request the characterisation and collect its results. The test-bed used during the simulation is shown in Figure 3.7. In this, systems were connected to a switch via Ethernet cables. High-priority traffic was not considered during the simulations, and it cannot be excluded that this may have affected the kernel behaviour.

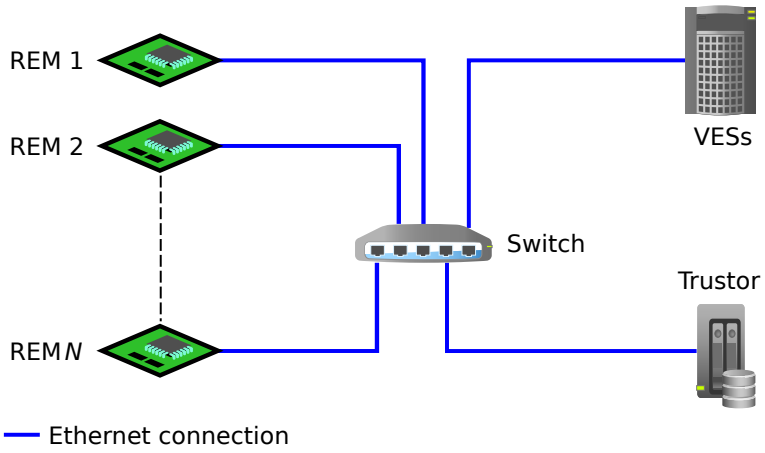


FIGURE 3.7: Test-bed used during the simulations for performing characterisation requests and collect their results from REM and VES systems.

3.4.2 Machine Emulation Detection Algorithm

The task of the machine emulation detection algorithm (MEDA) is to use the output of the characterisation algorithm that is applied to the target machine, and then to predict it to be either an REM or VES. To do this, Characterisation Metrics (CMs) are extracted from behaviours of REMs, as shown in Table 3.3.

The REMs used for obtaining the CMs are ALIX 6F2 [227], Google Nexus 5 and 7 [228, 229], Carambola [230], Arduino Yún [231] and Raspberry Pi [232]. A range for each CM is obtained by applying the characterisation algorithm to every REM considered. Results are shown in Table 3.4, and these ranges of CMs are then used as threshold values for detecting VESs.

In this table, results from different tests were merged in order to evaluate the different behaviours of each machine. This was done because a machine can have several applications running at the same time. Therefore, by merging results from tests 1 and 3 (normal operation after booting) and tests 2 and 4 (intensive operation), it is possible to obtain useful data independently from the machine operational status. As the behaviours of the ping response time from all the tests were not very different (with and without stressing the CPU, and with different delays between two pings), these were merged to obtain a unique range. For the same reason, the results from tests 1 and 2, and tests 3 and 4, in terms of timestamp values also were merged. Moreover, this was

TABLE 3.3: Characterisation metrics.

Metric	**Equation**
Minimum (Min)	$\min = \min(x_i)$
Maximum (Max)	$\max = \max(x_i)$
Range	$range = \min - \max$
Sum	$sum = \sum_{i=1}^{N} x_i$
Mean	$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$
Variance	$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2$
Standard deviation (SD)	$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2}$
Mean-Standard deviation (Mean-SD)	$\mu - \sigma$
Mean+Standard deviation (Mean+SD)	$\mu + \sigma$

$x_i = i$-th sample; $N =$ number of samples; $freq =$ most frequent value; $sort =$ values in sorted order.

done also for CPU usage results, but only for tests in which the CPU was not under stress.

As shown in Table 3.4, the results from test 1 to 4 show three behaviours: ping response time (P.) for all tests; the behaviours of timestamp values (T.) when the same interval between pings is chosen; and the behaviours of the CPU usage (C.) when the CPU was not under stress conditions. Note that tests 5 to 8 are not shown in this table; these were excluded because they do not give reliable results for detecting VESs. Specifically, there was an issue related to the ping option used to specify the ping packet size. In fact, large-sized ping packets require high computational resources and some embedded systems are not able to handle them properly.

The behaviours of REMs shown in Table 3.4 are used by MEDA as a threshold value for detecting VESs. Let $TMin_{REMs}(CM_i)$ and $TMax_{REMs}(CM_i)$ be the minimum and maximum value in the range for the i-th characterisation metric from Table 3.3. Let $T_T(CM_i)$ be the i-th characterisation metric obtained from the target system. A VES is considered detected if one of the following equations is valid:

$$T_T(CM_i) < TMin_{REMs}(CM_i) \tag{3.2}$$

$$T_T(CM_i) > TMax_{REMs}(CM_i) \tag{3.3}$$

MEDA is summarised in Figure 3.8.

TABLE 3.4: Range of behaviours of real embedded devices (REMs) obtained during the characterisation.

Characterisation Metrics	**P. (ms) Tests 1 to 4**	**T. (s) Tests 1 and 2**	**T. (s) Tests 3 and 4**	**C. (%) Tests 1 and 3**
Min	0.067-0.193	0	0	0-75
Max	0.140-2.060	1-2	1-3	19-100
Range	0.061-1.993	-	-	-
Sum	99.064-288.117	199-201	999-1001	-
Mean	0.099-0.288	0.199-0.201	0.999-1.001	5.223-79.042
Variance	0-0.034	0.159-0.162	0.001-0.059	4.928-459.201
Standard Deviation (SD)	0.002-0.183	0.399-0.402	0.032-0.243	2.220-21.429
Mean-Standard Deviation (Mean-SD)	0.060-0.215	-	-	-
Mean+Standard Deviation (Mean+SD)	0.110-0.452	-	-	-

Input: set of characterisation metrics from the target system
Output: true or false

```
for each CM_i ∈ CM :
    if (T_T(CM_i) < TMin_REMs(CM_i) ||
        T_T(CM_i) > TMax_REMs(CM_i))
        return true          // illegitimate REM
return false                 // legitimate REM
```

FIGURE 3.8: Pseudo code for the Machine Emulation Detection Algorithm (MEDA).

3.5 Results and Discussion

To evaluate the capability of MEDA to detect forged embedded machines, several VESs were tested; these are shown in Table 3.5. All the VESs were implemented using a machine running Linux Mint 17 (qiana) with kernel 3.13.0-24-generic OS, an Intel(R) Core(TM) i3-4130 CPU @ 3.40GHz CPU with 4 cores and 7897 MiB of RAM, with default configurations.

TABLE 3.5: List of virtualised and emulated embedded systems (VESs) tested.

VES	Notation	Architecture	OS
Android Emulator [233]	AE	ARMv7	Android 4.4.2
Genymotion [234]	GN	Embedded x86	Android 4.4.4
GXemul [235]	GX1	MIPS	NetBSD 5.0.2
GXemul [235]	GX2	MIPS	NetBSD 6.1.5
OVPsim [236]	OVP	MIPS	Debian
QEMU [237]	Q1	MIPS	OpenWrt 12.09
QEMU [237]	Q2	MIPSel	OpenWrt 12.09
QEMU [237]	Q3	ARMv6l	Raspbian
VirtualBox [238]	VB	Embedded x86	OpenWrt 10.03
VMware [239]	VM	Embedded x86	OpenWrt 14.07

Results of MEDA are shown in Figures 3.9 and 3.10. In order to obtain each detection, ranges of CMs from Table 3.4 are used. For example, results from tests 1 and 2 (Figure 3.9) show that behaviours of GXemul with NetBSD 6.1.5 (GX2) were outside ranges of P.Sum, P.Mean and P.Mean-SD. Therefore, the number of detections by applying MEDA is three, giving as a final result, that GX2 is a VES.

Furthermore, these results show that the Android Emulator (AE) is more easily detected than GX2. At the same time, it is possible to see that changing the OS version can result in an easier detection, as shown by GX1 and GX2 results. Finally, it is clear that all VESs can be detected by using only four CMs: P.Sum, P.Mean±SD and T.Sum.

Moreover, by comparing the results from Figures 3.9 and 3.10, it is possible to see that MEDA works better with the information retrieved by the characterisation algorithm during tests 3 and 4 rather than tests 1 and 2. Furthermore, tests 3 and 4 require

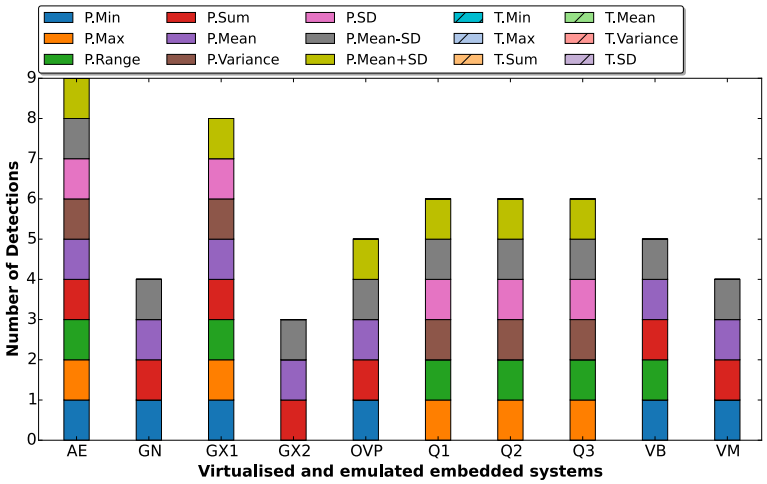


FIGURE 3.9: MEDA results for tests 1 and 2.

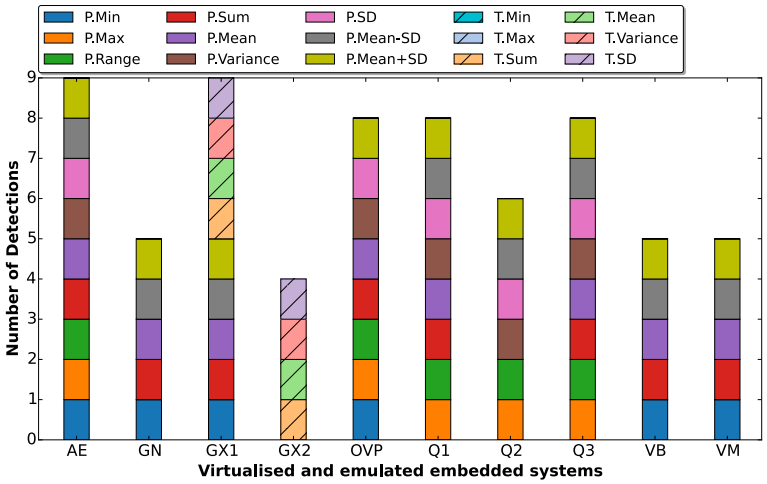


FIGURE 3.10: MEDA results for tests 3 and 4.

only around 3 minutes, while tests 1 and 2 require around 17 minutes to detect forged embedded machines.

Therefore, by using the characterisation algorithm with a ping interval of 200 ms (tests 3 and 4), MEDA can detect a forged embedded machine within 3 minutes with a detection accuracy of 79.21%. Moreover, MEDA associates every item labelled as REM belonging to REMs (precision = 1), but almost half of the items from REMs were labelled as belonging to VESs (recall = 0.4455). While, almost every item labelled as VES belongs to VESs (precision = 0.7504) and all of the items from VESs were labelled as belonging to VESs (recall = 1). Detailed results can be found in Appendix B.1. From these results, MEDA is capable of detecting VESs within a certain degree of accuracy, but it has problems in recognising REMs. These aspects and detection time are very important for minimising the power consumption of battery-powered embedded devices, and also for enabling a system to be trusted easily and quickly.

3.5.1 Comparison with Other Techniques

For the reasons explained in Section 3.3, methods available in the literature for detecting VESs cannot be used in the IoT. In fact, these are mostly architecture- and OS-dependent, and they cannot easily be applied in embedded devices due to their heterogeneity. IoT devices should be treated as black-boxes without previous knowledge of their architectures and OSs. This is a very important aspect for two main reasons: (i) the architecture can be hand-crafted using Field-Programmable Gate Array (FPGA)-based boards, and (ii) future IoT devices may have a different or an improved architecture, which may lead to different instruction sets, e.g. instructions used until ARMv4 were not able to access half-word objects in the memory, although this is not true for newer ARM versions [240]. However, the only detection method that could be used in all situations is the fingerprinting test, but it can be easily attacked as demonstrated in Subsection 3.3.4.

The fingerprinting test was able to extract "signatures" from only the Genymotion, VirtualBox and VMware systems. Results from this test are shown in Table 3.6. The "signatures" used for detecting these VESs are vbox, virtualbox, virtualized, oracle, innotek, intel, genuineintel, genymotion, vmware and their variants.

Despite how difficult it is to use existing methods to detect VESs, the architecture-based timing test [209, 210] has been successfully implemented in the Raspberry Pi 2 system model B based on an ARM Cortex-A7 CPU. This has been tested in both the real hardware and the QEMU v2.12.0-rc0 emulated system. This test has been implemented as a loadable kernel module, as this is the only way to access the CPU registers of the Raspberry Pi 2. As highlighted in Subsection 3.3.2, this test consists of timing the access to CPU control registers and an NOP operation. In the Raspberry Pi 2, there is only one available system control register (SCTLR) [241], and this has been used during the timing test. The timing information is retrieved with the cycle count register (CCNT). Moreover, the QEMU has been modified in order to adjust the cycle count register because the frequency of the ARM processor is not configurable. This modification helped to defeat the timing test. The kernel module used to perform the test and a patch for QEMU are available in Appendices A.5 and A.6. Results from the Raspberry Pi 2 and QEMU were the same. The time required by an NOP operation is equal to 1 clock cycle and the time required to access the SCTLR registers is equal to 2 clock cycles. These results show that the architecture-based timing test is not a viable test for detecting VESs when embedded machines are used. The application of this test is also limited by the fact that, in order to access the SCTLR and the CCNT registers, super user access is required. This is very dangerous as an attacker can use this access to destroy the system or gather all types of information.

Table 3.6: Fingerprinting information from Genymotion, VirtualBox and VMware.

Signatures from files or applications	**Information**	**GN**	**VB**	**VM**
/proc/cpuinfo	Real CPU characteristics	Yes	Yes	Yes
/proc/version	Linux version	Yes	No	No
/proc/misc	Virtual users	Yes	No	No
/proc/ioports	Virtual devices	Yes	No	No
/proc/kcore	Physical memory and system message	Yes	Yes	Yes
dmesg				
/sys/devices/pci*XXXX*:*XX*/*XX XX*:*XX*:*XX.X*/*XXXX*:*XX*:*XX.X*/usb1/1-2/product	Hard disk, USB and CD-ROM devices	Yes	Yes	Yes
/sys/devices/pci*XXXX*:*XX*/*XX XX*:*XX*:*XX.X*/*XXXX*:*XX*:*XX.X*/usb1/1-2/configuration				
/sys/devices/pci*XXXX*:*XX*/*XX XX*:*XX*:*XX.X*/*XXXX*:*XX*:*XX.X*/usb1/1-2/1-2:1.0/interface				
/proc/scsi/scsi				
/sys/sys/devices/virtual/dmi/id/sys_vendor	System vendor	Yes	No	No
/sys/sys/devices/virtual/dmi/id/board_name	Board version	Yes	No	No
/sys/sys/devices/virtual/dmi/id/board_vendor	Board vendor	Yes	No	No
/sys/sys/devices/virtual/dmi/id/bios_vendor	BIOS vendor	Yes	No	No
/sys/firmware/acpi/tables/DSDT	ACPI table information	Yes	Yes	Yes
/sys/firmware/acpi/tables/FACP				
/sys/firmware/acpi/tables/SSDT				
/fstab.vbox86	Virtual machine boot files	Yes	No	No
/init.vbox86.rc				
lsmod	Virtual modules	Yes	No	No
/system/lib/vboxsf.ko				
/system/lib/vboxguest.ko				
/system/lib/vboxvideo.ko				
/system/bin/androVM-prop	Virtual machine software	Yes	No	No
/system/bin/androVM-vbox-sf				
/system/bin/androVM_setprop				
ps	Virtual machine running applications	Yes	No	No
/proc/*XXX*/mem				
/system/build.prop	Android information	Yes	No	No
/system/etc/init.androVM.sh	Boot scripts	Yes	No	No

X is an integer value identifying the bus number or the process number.

3.6 Summary

In this chapter, a new threat was presented that uses forged embedded machines to attack M2M communications in the IoT. Hypotheses of using this attack in real-life scenarios were also given to underscore its dangerousness when applied to the IoT. A review of methods for detecting forged embedded machines was carried out and this demonstrated that current solutions cannot be used to detect this new threat. Therefore, a new detection method based on embedded machines behaviours called MEDA was proposed for detecting forged embedded machines in M2M communications. This method allows M2M-embedded machines to perform the detection without relying on the machine architecture and the OS. Finally, evaluation of the proposed method showed that it is effective in detecting VESs. However, limitations related to the time required to identify forged embedded machines and to properly detect REMs underline the toned for further study this field. In the next chapter a new method will be proposed which aims to address these limitations.

Chapter 4

A Classification Approach to Detecting Forged Embedded Machines

In Chapter 3, a threat against M2M-embedded machines in the IoT was presented. This threat is based on the use of forged embedded machines to subvert M2M communications and create M2FM communications. Several detection methods present in the literature were reviewed, but these proved to be inapplicable to IoT/M2M-embedded machines. Therefore, a novel detection method, called MEDA, was presented that uses a range of behaviours obtained from REMs as threshold values for detecting VESs. In this chapter, the limitations of MEDA are presented and a new detection method based on a classification approach is proposed. This solution has been accepted for publication by Valerio Selis and Alan Marshall in *IEEE Systems Journal* under the title, 'A classification-based algorithm to detect forged embedded machines in IoT environments' [110].

4.1 Background and Motivation

In Chapter 3, the available methods for detecting VESs were categorised as:

- *CPU and memory tests*: memory states, CPU registers and instructions are used;
- *Remote tests*: information obtained from the network are used;
- *Architecture-based timing tests*: time analysis using CPU instructions;
- *Fingerprinting tests*: specific "signatures" from the system are extracted;
- *Behavioural tests*: system behaviours are used.

Research into detecting forged IoT/M2M-embedded machines is at an early stage. As shown in Table 4.1, the method proposed in this thesis (MEDA) is the first to address this specific problem.

TABLE 4.1: Summary of methods available in the literature for detecting virtual and emulated systems.

Reference	Architecture/OS/VES	Detection method	Key part	Suitable for IoT/M2M REMs
Rutkowska (2004) [206]	x86 (single core)	CPU and memory tests	"The Red Pill": value of the IDT register	No
Quist and Smith (2006) [218]	VMware/VirtualBox	Fingerprinting tests	Access to virtual machine APIs	No
Raffetseder *et al.* (2007) [209]	x86	Architecture-based timing tests	Time to access control register CR0 and to execute a NOP instruction	No
	OS information	Fingerprinting tests	MAC address used to obtain vendor name	No
Chen *et al.* (2008) [211]	Linux-based OSs	Remote tests	TCP timestamp option to detect the clock skew	No
	OS information	Fingerprinting tests	MAC address and driver names	No
Martignoni *et al.* (2009) [207]	x86	CPU and memory tests	Same instruction will give same results in CPU and memory states	No
Jia-Bin *et al.* (2012) [210]	x86-64	Architecture-based timing tests	Time to access control registers and to execute a NOP instruction	No
	OS information	Fingerprinting tests	MAC address, driver names, registry keys and running applications	No
Shi *et al.* (2014) [208]	x86	CPU and memory tests	"Cardinal Pill": all definitions in the IA-32 manual are used to generate pills	No
Jing *et al.* (2014) [219]	Android emulator	Fingerprinting tests	Android APIs, system properties and hardware information (20 minutes)	No
Vidas and Christin (2014) [220]	Android emulator	CPU and memory tests	CPU and graphical performances	No
	Android emulator	Fingerprinting tests	Android APIs, system properties and hardware information	No
Selis and Marshall (2015) [6]	Independent	Behavioural tests	Behaviours of IoT/M2M-embedded machines	Yes

However, as highlighted in Section 3.5, MEDA requires approximately 3 minutes to identify a forged embedded machine with an accuracy of roughly 79% and has problems recognising real embedded machines. These performances can lead to limitations in the applicability of MEDA in IoT environments for three main reasons: (i) moderate detection uncertainty, (ii) low recognition of real embedded machines and (iii) slowness in the detection. These issues are related to the time required for the characterisation and the fact that MEDA represents an embedded machine with a feature space of one feature per unit time by applying pre-defined threshold values. This can be seen in the accuracy value, which could include REMs misclassified and/or undetected VESs. For these reasons, another approach for detecting forged embedded machines in M2M communications in the IoT was developed. This approach consists of the implementation of a new detection method based on a classification algorithm. In fact, using a classification algorithm provides an opportunity to represent an embedded machine with a feature space of two or more features per unit time. In order to speed up the detection, the characterisation algorithm, presented in Subsection 3.4.1, is tuned (Figure 4.1). The main difference is that the number of pings for detecting VESs will be determined by changing the n value. Also, as is expected, by reducing this number it is possible to decrease the overall detection time.

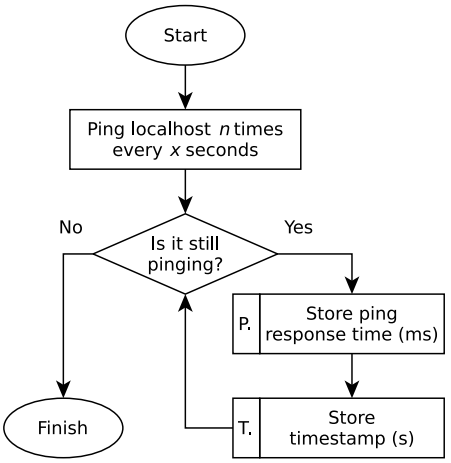


FIGURE 4.1: Modified characterisation algorithm for reducing the overall detection speed by changing the number of pings, in which x is fixed to 0.2.

4.2 Classification-based Algorithm

Classification methods provide an opportunity to assign an observation to a well-known set of categories. In the case presented here, there are only two categories, real embedded machines and forged embedded machines. This type of classification is well-known as a binary classification problem. Classification methods use a dataset with a series of features extracted from past observations for each category. However, choosing a classifier for detecting forged embedded machines is not a trivial task as it is not possible to know a priori which one is best to use. Therefore, several supervised learning

classification methods are used and evaluated, as shown in Figure 4.2. The following subsections highlight the steps required to efficiently choose the classification method in order to properly detect forged embedded machines in the IoT. All these steps will be performed for different numbers of pings, as noted later in this chapter.

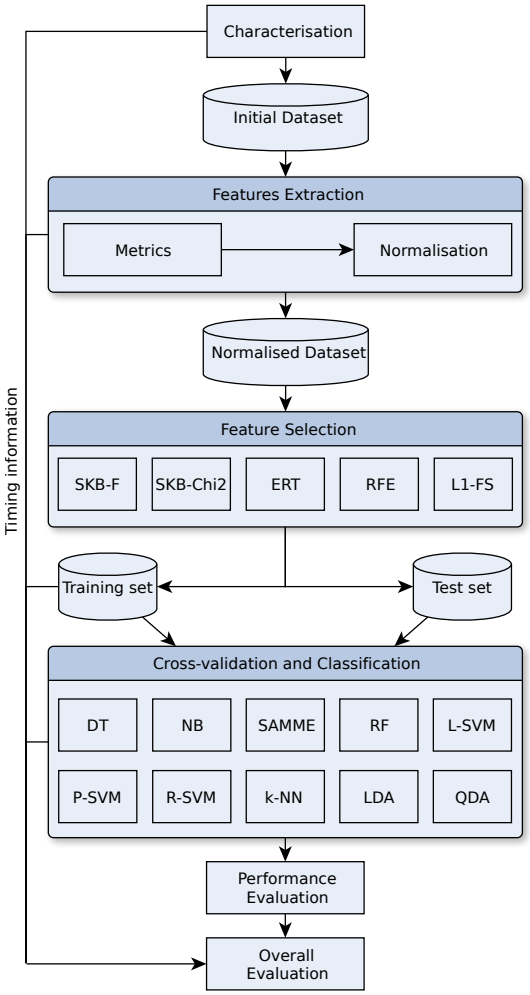


FIGURE 4.2: Steps required for selecting the best classification method.

4.2.1 Initial Dataset

Creating the initial dataset is the first important step for detecting forged embedded devices in the IoT and is the basis for the classification. It is composed of information gathered from REMs and VESs using the modified characterisation algorithm. In order to create a valuable dataset, other REMs and VESs are characterised compared to the those used in Chapter 3, as summarised in Table 4.2.

An important aspect of creating a good dataset is the number of observations made of each type of system in order to properly characterise them. Therefore, for each system, 500 characterisation tests were performed with normal CPU usage and 500 characterisation tests with the CPU under stress (CPU level around 100%). This was done in order to replicate different behaviours that could occur when a system is running.

TABLE 4.2: List of real, virtual and emulated embedded systems characterised.

System	Architecture	Operating System	Type
ALIX 6F2	Embedded x86	Debian	REM
Android Emulator	ARM	Android	VES
Arduino Yún	MIPS	OpenWRT	REM
Carambola	MIPS	OpenWRT	REM
Genymotion	Embedded x86	Android	VES
Google Nexus 5	ARM	Android	REM
Google Nexus 7	ARM	Android	REM
GXemul	MIPS	NetBSD	VES
Netgear Centria WNDR4700	PowerPC	OpenWRT	REM
Open Mesh OM2P-LC	MIPS	OpenWRT	REM
OVPsim	MIPS	Debian	VES
QEMU	ARM	Debian	VES
QEMU	MIPS	OpenWRT	VES
QEMU	MIPSel	OpenWRT	VES
QEMU	PowerPC	OpenWRT	VES
Raspberry Pi	ARM	Debian	REM
Samsung Chromebook Series 3	ARM	Chrome OS	REM
Samsung Galaxy Tab GT-P1000	ARM	Android	REM
VirtualBox	Embedded x86	OpenWRT	VES
VMware	Embedded x86	OpenWRT	VES

REM: real embedded machine; VES: virtual or emulated embedded system.

As this is a binary classification problem, there are 10,000 characterisation tests for REMs associated to class 0 and 10,000 characterisation tests for VESs associated to class 1. The dataset is composed of 20,000 observations in total. For each observation, information about ping response time (P.) and timestamp value (T.) is collected.

4.2.2 Feature Extraction

The next step for classifying REMs and VESs is to extract features from each observation in the dataset, and in particular from P. and T. measurements. These measurements are related to the timing behaviours of each system under consideration. Therefore, in order to calculate their variability, statistical methods are used, as shown in Table 4.3. Features are extracted from central tendency, dispersion and distribution of the data after considering Hypothesis 1 (Subsection 3.4.1), in which delays introduced by VESs can change the timing behaviours over time.

TABLE 4.3: Characterisation Features.

Feature	**Equation**
Basic summary statistics	
Minimum	$\min = \min(x_i)$
Maximum	$\max = \max(x_i)$
Sum	$sum = \sum_{i=1}^{N} x_i$
Range	$range = \min - \max$
Central tendency	
Mean	$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$
Mode	$mode = freq(X)$
Median	$median = \frac{1}{2} \left(x_{\frac{N}{2}} + x_{\frac{N}{2}+1} \right)$ of $sort(X)$
Dispersion	
Variance	$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2$
Standard deviation	$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2}$
Shape	
Upper/Lower bounds (95%)	$U/L = \mu \pm 1.96\sigma$
Skewness	$SKEW = \frac{\sum_{i=1}^{N} (x_i - \mu)^3}{N\sigma^3}$
Kurtosis	$KURT = \frac{\sum_{i=1}^{N} (x_i - \mu)^4}{N\sigma^4}$
Correlation	
Pearson correlation coefficient	$r = \frac{\sum_{i=1}^{N} (x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^{N} (x_i - \mu_x)^2} \sqrt{\sum_{i=1}^{N} (y_i - \mu_y)^2}}$

X = all samples; x_i = i-th sample; N = number of samples;
$freq$ = most frequent value; $sort$ = values in sorted order.

To avoid zero entries in sparse data and improve further steps against small standard deviations of features, all features are pre-processed by using the Min-Max normalization method:

$$\tilde{x}_i^j = \frac{x_i^j - min^j}{max^j - min^j} \tag{4.1}$$

where x_i^j is the i-th value of the j-th feature and max^j and min^j are respectively the maximum and minimum values of the j-th feature. This method normalises the features in a range between 0 and 1. Therefore, a new dataset is created, called the normalised dataset, in which there are 14 features for P. and 14 features for T., with a total of 28 features available for characterising each observation.

4.2.3 Feature Selection

At this stage, the dataset is ready to be used for classification and it contains 28 features, half from P. and half from T. However, it is possible that not all these features are needed to properly classify a forged embedded machine. Therefore, a feature selection step is required. This step is used for selecting the most relevant features present in the dataset. In fact, by selecting only some of the 28 features, the computational requirements needed for the detection will be lower than it would be if all of them were used. This not only reduces the complexity but also speeds up the detection. By looking to Hypothesis 1 (Subsection 3.4.1), it is possible to formulate Hypothesis 2, based on the importance of features.

Hypothesis 2. REMs are expected to behave in a similar way during their functional operation because these have analogous capabilities, as highlighted in Table 1.1 (Introduction). Therefore, basic summary features can play an important role in detecting VESs.

Several feature selection methods are used to create a rank of the best features in the normalised dataset:

- *Extremely Randomized Trees ERT*: fits a number of randomized decision trees on various sub-samples of the dataset [1, 242];
- *L1-based Feature Selection L1-FS*: removes features by selecting non-zero coefficients and by adopting a linear Support Vector Machine Classifier (SVC) penalised with the L1 norm [1, 243];
- *Recursive Feature Elimination (RFE)*: removes features with low weights by adopting a linear SVC [1, 244];
- *Select-K-Best (SKB)*: selects features by looking to the k highest scores with both Select-K-Best with ANOVA F-value (SKB-F) and Select-K-Best with Chi-squared stats of non-negative features (SKB-Chi2) [1].

By applying these methods to the normalised dataset, the best k-th features from P. and T. are extracted, where k is a value between 1 and 14. Obviously, as previously noted, at least one feature from P. and T. will be used during the classification in order to avoid misdetections caused by attackers faking P. or T.

4.2.4 Classification

This step is the core of the classification-based algorithm, in which observations from a target embedded machine are assigned to a specific category or class of REM or VES. The classification of a target embedded machine is carried out in two steps: learning and testing.

Firstly, the classifier is trained to learn how to identify whether the observation belongs to a REM or a VES. This is done using pre-collected data (a training set) in which for each sample the category is well-known. Classifiers can be tuned by using specific parameters to improve the classification. The best way to select these parameters is to estimate the classifier performances by using a cross-validation over the training set. In this thesis, a stratified K-fold cross-validation is used with K equal to 10. It consists of randomly subdividing the training set into k parts, with equal quantities of samples in each category, then $k-1$ parts are used to train the classifier, and one part is used to test it. This step is then iterated k times in order to obtain a good estimation of the classifier performances; in this case, $k = 10$.

Secondly, the classifier uses its training to classify new observations present in new data collected (the test set). However, in order to test how well the classifier assigns new observations to the correct category, the categories in the test set are known, but these are not used by the classifier.

The normalised dataset is therefore subdivided in two parts: 75% is assigned to the training set and 25% is assigned to the test set. At this stage, everything is ready for the classification step. As noted previously, it is not possible to know a priori which classifier is the best one for detecting forged embedded machines. Therefore, several supervised learning algorithms for classification problems are used:

- *Decision Tree (DT)*: this is a predictive modelling approach based on a flow chart-like structure in which internal nodes contain classification rules and leaf nodes contain class labels. The classification rules are obtained from the data features. The predicted class is obtained by testing the value with the decision tree starting from the root [1, 245–247];

- *Naïve Bayes (NB)*: this is based on Bayes' theorem with the "naive" independence assumptions between every tuple of features [1, 248];

- *k-Nearest Neighbour (k-NN)*: this method creates instances of the initial dataset (normalised training set) by assigning a vote based on k neighbour tuples. An unknown tuple is classified by using these instances and in particular by searching the k neighbour tuples nearest to it [1];

- *Linear Discriminant Analysis (LDA)*: finds a linear decision boundary that is successively used to separate classes that are assumed to have the same covariance [1, 247];

- *Random Forest (RF)*: uses a collection of decision tree classifiers, termed a forest, on random sub-samples of the dataset (normalised training set). This uses the average of probabilistic predictions from classifiers to combine them in order to control the over-fitting [1, 249];

- *Stagewise Additive Modelling using a Multi-class Exponential Loss Function (SAMME)*: this is based on the AdaBoost method for multiclass generalisation of the exponential loss. It consists of fitting a set of weak classifiers on the dataset (normalised training set) and then it adjusts the weights depending on the classification results, in order to achieve the best result for subsequent classifications. The base estimator used for building the boosted ensemble is the optimal DT [1, 250];

- *Support Vector Machine (SVM)*: this is an algorithm in which the initial dataset (normalised training set) is transformed to a higher dimension by using non-linear mapping. A hyperplane is then searched in order to properly separate the classes. The tuples that fall on this hyperplane are called support vectors [1, 251, 252]. SVM is usually used by default to solve linear problems. However, it is not always possible to separate data linearly and for this reason kernel methods can be used to work in high-dimensional spaces. Three kernel methods were used with SVM: Linear kernel-based SVM (L-SVM), Polynomial kernel-based SVM (P-SVM) and Radial kernel-based SVM (R-SVM). SVM supports two parameters, C and γ, that are respectively the penalty for misclassification and the deviation of the kernel;

- *Quadratic Discriminant Analysis (QDA)*: this uses the same concept as LDA, but a quadratic decision boundary is adopted to separate classes that could have a different covariance [1, 247].

Table 4.4 shows the tuning parameters used during the classification step for each classifier according to the Scikit-learn Python module [1].

4.2.5 Performance Evaluation

After classification, another step is required to evaluate the performances of each classifier in detecting VESs. Each classifier, when assigning samples in the test set to a specific class, gives classification values. This is a binary classification problem with two classes: REM, known as positive class (0), and VES, known as negative class (1). These values are then compared to the real classes for each sample, from which it is possible to create a confusion matrix (Table 4.5) where:

- *True Positives (TP)*: positive samples that are classified as positive (correct result);

- *True Negatives (TN)*: negative samples that are classified as negative (correct absence of result);

- *False Positives (FP)*: negative samples that are classified as positive (unexpected result);

- *False Negatives (FN)*: positive samples that are classified as negative (missing result).

TABLE 4.4: Summary of parameters used for each classifier in the Scikit-learn Python module [1].

Classifier	**Parameter**	**Value**
DT	criterion	gini, entropy
	max_features	1, sqrt, log2, None
SAMME	base_estimator	the best DT for each feature tuple
	algorithm	SAMME, SAMME.R
	n_estimators	1, 5, 10, 50, 100, 200, 500, 1000
RF	max_features	1, sqrt, log2, None
	n_estimators	1, 5, 10, 50, 100, 200, 500, 1000
SVM	C	10^{-2}, 10^{-1}, $\cdots$, 10^{3}
	γ	10^{-9}, 10^{-8}, $\cdots$, 10^{3}
k-NN	k neighbours	1, 2, $\cdots$, 30
	weights	uniform, distance
LDA	solver	svd, lsqr, eigen
	shrinkage	None, auto

TABLE 4.5: Confusion Matrix.

		Predicted class	
		Yes	No
Actual class	Yes	TP	FN
	No	FP	TN

In order to evaluate each classifier, four performance measures are used:

- *Accuracy*: measures how well the classifier classifies the samples;
- *Precision*: measures how many samples are classified as positive and are actually positive as a percentage,
- *Recall*: measures how many positive samples are classified as positive as a percentage;
- *F1-score*: measures the harmonic mean of precision and recall.

Results from these performance measures are between 0 (worst result) and 1 (best result). The last three performance measures are calculated for both REMs and VESs in order to understand how well each class is classified. To produce a final performance value for each classifier, the average of these performance measures is calculated. This is the Overall Detection Performance (ODP).

Another aspect to be considered when using classification methods is the time required to properly classify each observation as this can affect the detection speed. Times that must be considered are the characterisation time (T_{Ch}), the feature extraction time (T_{FE}) and the classification time (T_{Cl}). The total time required is then calculated; this is termed the ODS. A summary of the equations used are shown in Table 4.6.

TABLE 4.6: Classification evaluation measures.

Measure	**Equation**
Accuracy	$A = \frac{TP+TN}{TP+TN+FP+FN}$
Precision	$P_m = \frac{TP}{TP+FP}$
Recall	$R_m = \frac{TP}{TP+FN}$
F1-score	$F1_m = 2\frac{P_m \times R_m}{P_m+R_m}$
Characterisation Time	$T_{Ch} = n_{pings} \cdot 0.2s$
Feature Extraction Time	$T_{FE} = FT_{FE} - ST_{FE}$
Classification Time	$T_{Cl} = FT_{Cl} - ST_{Cl}$
Overall Detection Performance	$ODP_i^j = \frac{A_i^j + \sum_{m=0}^{1} P_{m_i}^j + R_{m_i}^j + F1_{m_i}^j}{7}$
Overall Detection Speed	$ODS_i^j = T_{Ch} + \frac{T_{FE}^j + T_{Cl_i}^j}{N}$

$m = 0$ (REM) or 1 (VES); i is the i-th combination of feature selection method and classifier; j is the j-th tuple of features selected (kP, kT), where k is the number of the k-th best features; n_{pings} is the number of pings used in input to the characterisation algorithm; FT and ST are respectively the final and the starting times for extracting or classify a tuple of features; N is the number of samples in the test set.

4.2.6 Overall Evaluation

The overall evaluation is the final step needed to combine ODS and ODP and obtain a final value for identifying the best classifier for different numbers of pings. To do this, the minimum value of ODS and the maximum value of ODP are calculated for each tuple of features (j) and each combination of feature selection methods and classifiers (i) as:

$$ODPmax_i^j = \max(ODP_i^j) \tag{4.2}$$

$$ODSmin_i^j = \min(ODS_i^j) \tag{4.3}$$

A score is assigned to ODS and ODP for each tuple of features and each combination of feature selection methods and classifiers between 0 (worst result) and 5 (best result), as shown in Table 4.7.

TABLE 4.7: Evaluation scores for the Overall Detection Performance and Speed.

Ranges	$ODPscore_i^j$
$ODP_i^j = ODPmax_i^j$	5
$ODPmax_i^j < ODP_i^j \leq ODPmax_i^j - 0.01\%$	4
$ODPmax_i^j - 0.01\% < ODP_i^j \leq ODPmax_i^j - 0.1\%$	3
$ODPmax_i^j - 0.1\% < ODP_i^j \leq ODPmax_i^j - 1\%$	2
$ODPmax_i^j - 1\% < ODP_i^j \leq ODPmax_i^j - 10\%$	1
$ODP_i^j > ODPmax_i^j - 10\%$	0
Ranges	$ODSscore_i^j$
$ODS_i^j = ODSmin_i^j$	5
$ODSmin_i^j < ODS_i^j \leq ODSmin_i^j + 0.001s$	4
$ODSmin_i^j + 0.001s < ODS_i^j \leq ODSmin_i^j + 0.01s$	3
$ODSmin_i^j + 0.01s < ODS_i^j \leq ODSmin_i^j + 0.1s$	2
$ODSmin_i^j + 0.1s < ODS_i^j \leq ODSmin_i^j + 1s$	1
$ODS_i^j > ODSmin_i^j + 1s$	0

i = i-th combination of feature selection methods and classifiers; j = j-th tuple of features selected (kP, kT), where k is the number of features.

TABLE 4.8: Values of ODP and ODS depending on the security level of the IoT application scenario.

Security Level	**ODP**	**ODS**
Extreme	100%	0%
Very high	90%	10%
High	80%	20%
Medium	70%	30%
Low	60%	40%
Very low	50%	50%

It is important to note that it is preferable to give more importance to ODP than to ODS, because it is better to properly detect forged embedded machines than be fast and risk not detecting them properly. Towards this end, a ratio of 70% ODP to 30% ODS was chosen. This corresponds to a medium level of security as per Table 4.8. Although other ratios were not tested in this work, the level of security can be adjusted according to the IoT application scenario; for example, in a military application the extreme security level is recommended, as it confers the highest detection rate. Using the selected ratio, another measure is calculated, the Final Evaluation Score (FES), as follows:

$$FES_i^j = \frac{ODPscore_i^j \times 70}{\max(ODPscore_i^j)} + \frac{ODSscore_i^j \times 30}{\max(ODSscore_i^j)} \tag{4.4}$$

where i is the i-th combination of feature selection methods and classifiers and j is the j-th tuple of features selected.

FES will return a value between 0 (worst result) and 100 (best result). After obtaining the final overall evaluation of feature selection and classification method, the rank of the best features used is calculated by:

$$rank\left(X_i\right) = \bigcup \begin{cases} rank\left(y_0 = 1\right) \\ rank\left(y_j\right) = rank\left(y_{j-1}\right) + 1 \end{cases} ; \text{ for } j = 1, ..., n \tag{4.5}$$

where X is a sorted set composed by the sum of how many times each feature is selected, by considering the i-th combination of feature selection and classifier methods which return the highest FES, y_j is the j-th element in X_i and n is the number of elements in X_i.

4.3 Simulation and Results

Simulations have been performed by using the Scikit-learn module in Python [1]. To evaluate the classification algorithm for detecting forged embedded machines, VESs were used with default configurations under a machine running Linux Mint 17 (qiana) with kernel 3.13.0-24-generic OS, an Intel(R) Core(TM) i3-4130 CPU @ 3.40 GHz CPU with 4 cores and 7897 MiB of RAM.

In this section, results from each step are presented by changing the number of pings in the characterisation algorithm 7 times with values of 1000, 500, 200, 100, 50, 25 and 15.

Table 4.9 shows information related to the times required for extracting all features from the initial dataset and for characterise a target embedded machine. As the timing information shows, changing the number of pings decreases the detection time.

TABLE 4.9: Timing information related to the characterisation and features extraction steps.

Number of pings	**Characterisation time (s)**	**Extraction of all features from the dataset (s)**	**Extraction of all features from a target (s)**
1000	200	43.229	0.002
500	100	29.558	0.001
200	40	21.304	0.001
100	20	18.778	0.001
50	10	18.100	0.001
25	5	17.651	0.001
15	3	16.651	0.001

The time required by feature selection methods for selecting the best tuple of features and by changing the number of pings is shown in Figure 4.3. The fastest feature selection method is SKB-F; the slowest is RFE. It is also possible to see that changing the number of pings does not affect the time for selecting the tuple with the best features. The time required by classification methods for classifying the best tuple for each number of pings is shown in Figure 4.4. In this figure, only results from classifiers that return the best classification for at least one tuple for the specified number of pings are shown. It is possible to see that SAMME returns the best classification of only one best tuple for 50 pings. The fastest classification method and the slowest on average are k-NN and RF respectively.

By summarising timing information from Table 4.9, Figure 4.3 and Figure 4.4, it is clear that characterisation time is the liming factor. This is related to the number of pings. Therefore, it is possible to drastically speed up the detection of forged embedded devices and reduce the time by reducing the number of pings in input to the characterisation algorithm.

The rank of the features selected using Equation 4.5 for different numbers of pings is shown in Figure 4.5. Because the lower value means that the feature was selected by the best combination of feature selection and classification methods, the features P.min and T.sum are the best. Therefore, these features are key for detecting VESs, thus confirming Hypothesis 2.

Important results are obtained from two main steps, feature selection and classification. Results from the first step are shown in Figure 4.6. The feature selection method that works well for selecting the best 2 and 4 features is SKB-F. L1-FS selects the best features from 10 to 18. ERT works very well for all different numbers of pings in selecting the best 2 to 26 features. SKB-Chi2 was used only to select the best 22 features. Finally, RFE was able to select the best features only five times (2, 4, 6, 24 and 26 features) for 100 and 200 pings. Results are not shown for the (14P, 14T) tuple as all features are used; therefore, in this case, this tuple is not necessary for the selection step.

In the second step, the best tuple of features selected from the first step and for different numbers of pings is used by each classification method for the classification. Results from this step are shown in Figure 4.7, in which the classification methods that return the highest FES value are shown. For this reason, the classification methods not present returned the lowest FES value and are not considered for classifying target embedded machines. From Figure 4.7, it is possible to see that k-NN is able to classify REMs and VESs by using the best 2 and 4 features for high and small number of pings. Meanwhile, RF is the best classification method overall as it is able to properly recognise REMs and VESs for a wide range of numbers of pings. SAMME and DT classify the best tuple of features only sporadically.

The results for ODP obtained by the best classifier for different numbers of pings are shown in Figure 4.8. These results show that a saturation point is reached when the best (5P,5T) tuple is selected. In Figures 4.6 and 4.7, the best feature selection and

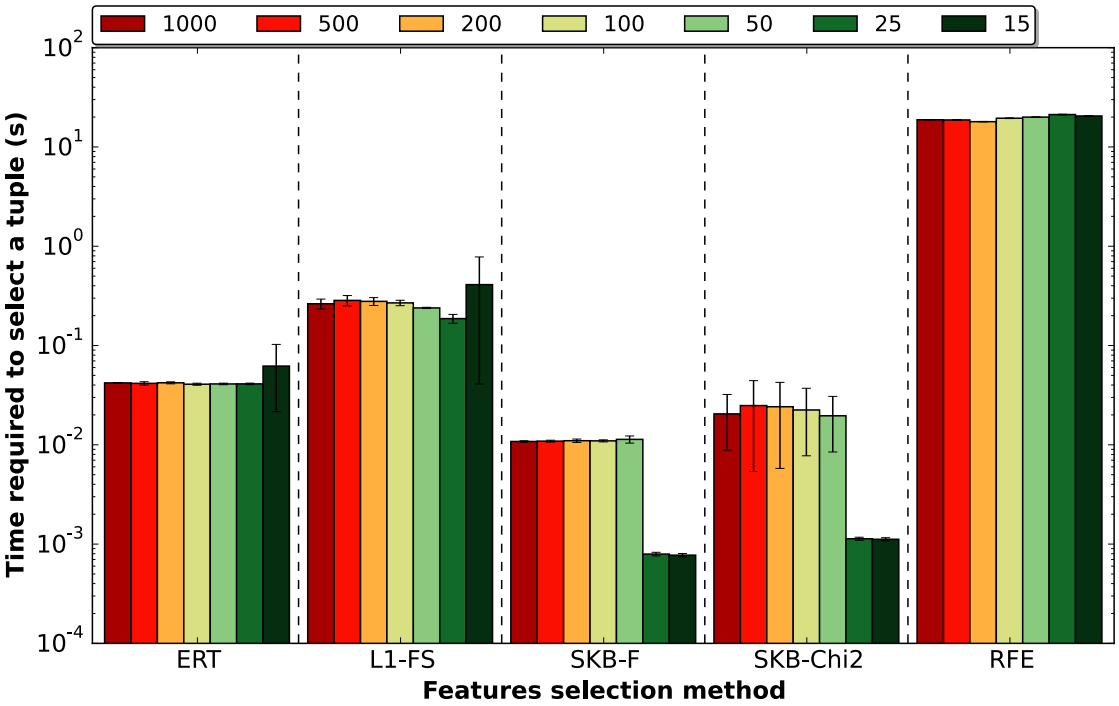


FIGURE 4.3: Time required by feature selection methods to select the best features from the training set and for different numbers of pings.

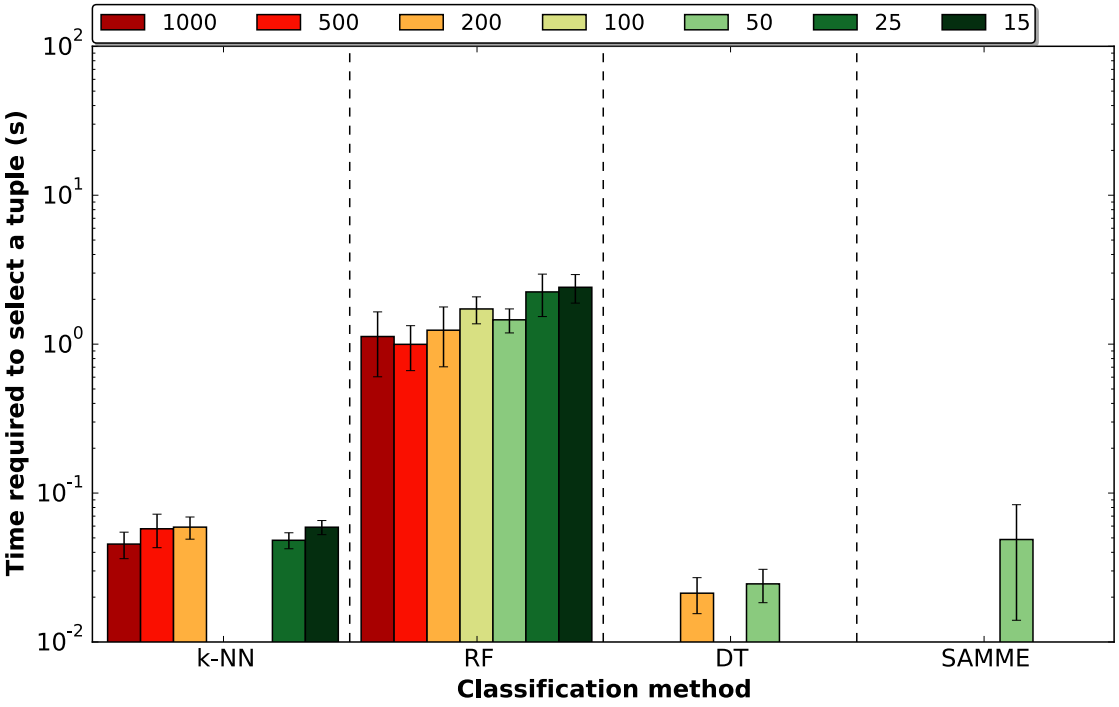


FIGURE 4.4: Time required by the best classification methods to classify all data present in the training set for different numbers of pings. Missing values mean that the classification method was not used for classifying a best tuple for that number of pings.

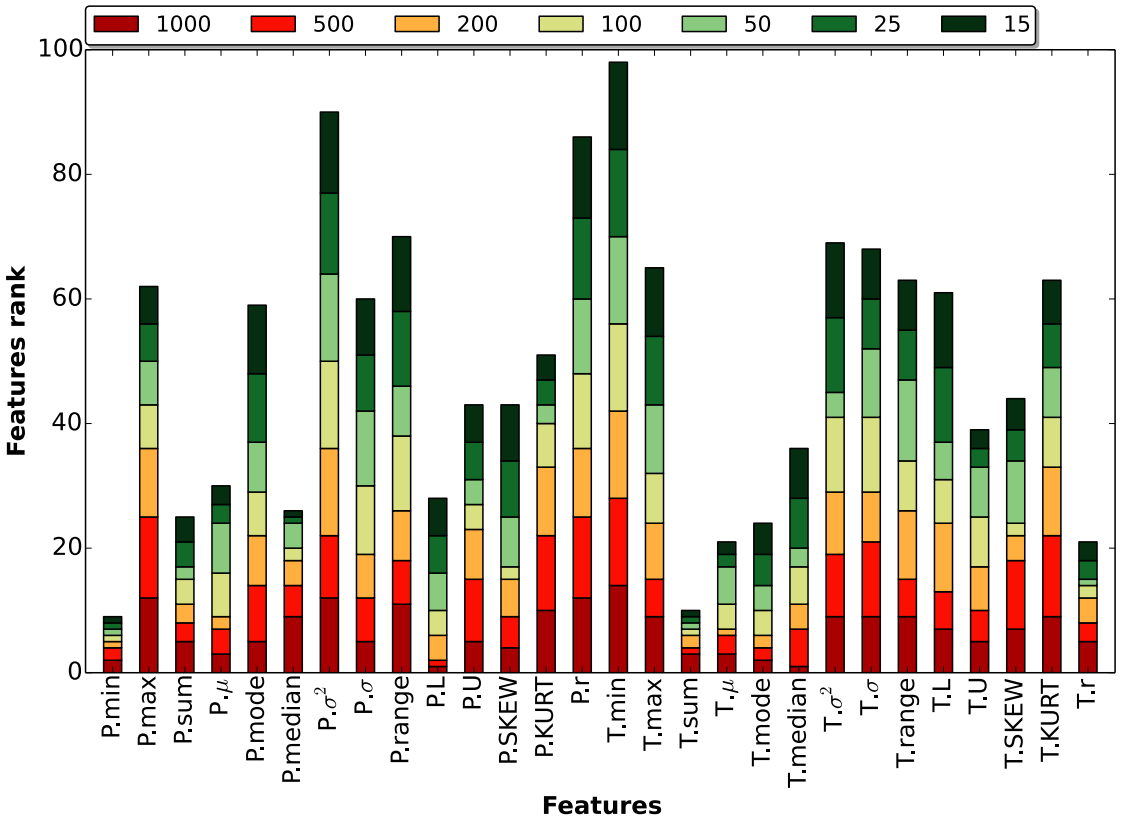


FIGURE 4.5: Features ranked by how many times they were selected by the best feature selection method for different numbers of pings. P. refers to ping response time, T. to timestamp values, and the abbreviations for features from Table 4.6 are used. For example, P.L means lower bound value of ping response time.

classification methods for the best 10 features are ERT and L1-FS, and RF respectively. Detailed results obtained by combining these methods are shown in Figure 4.9. In this, k-NN is also shown as it is the second classification method that gave a good classification. However, both ERT and L1-FS with RF perform better than ERT and L1-FS with k-NN. It is very interesting to note that k-NN, with the 10 features selected by L1-FS and for 15 pings, returns an ODP value of around 6 percentage points less compared to RF. This may be related to the relatively small variations obtained from 15 pings. In this and subsequent figures, the relation between number of pings and time is also shown for reference, which can be derived using:

$$time = n \cdot x \tag{4.6}$$

where n is the number of pings and x is the waiting seconds between sending each ping that is equal to 0.2 seconds. Table 4.10 shows the parameters used by k-NN and RF in the Scikit-learn Python module [1] for obtaining the best results for both ERT and L1-FS feature selection methods and 25 and 200 pings. Information about the time required for the combination of the best feature selection and classification method is shown in Figure 4.10, in which k-NN is faster than RF, but RF is preferred over k-NN because more weight is given to ODP than ODS.

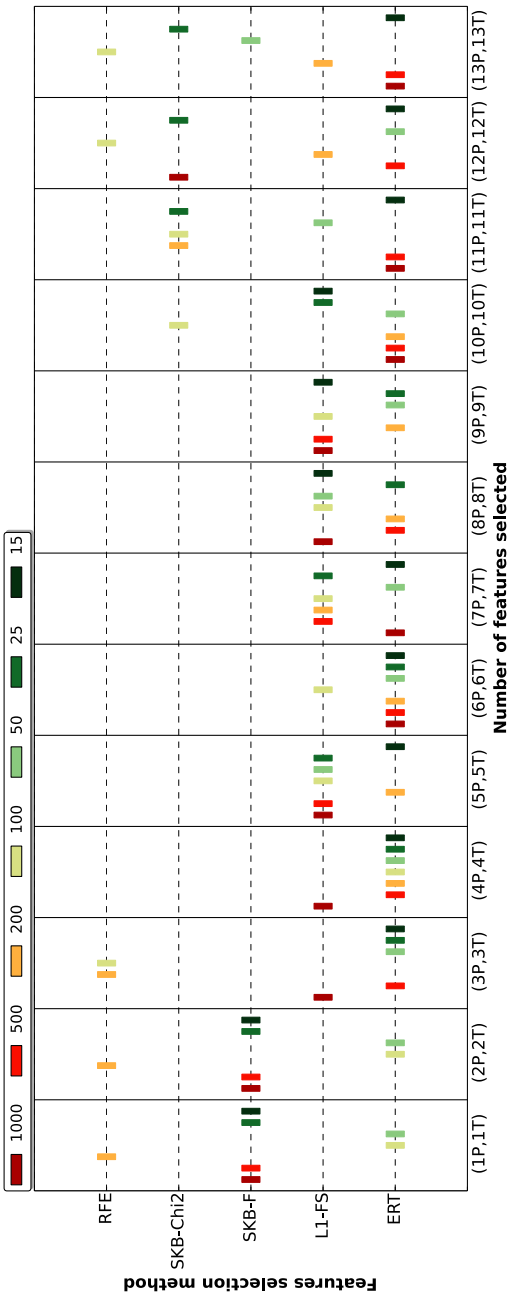


FIGURE 4.6: Feature selection methods used for selecting the best tuple of features.

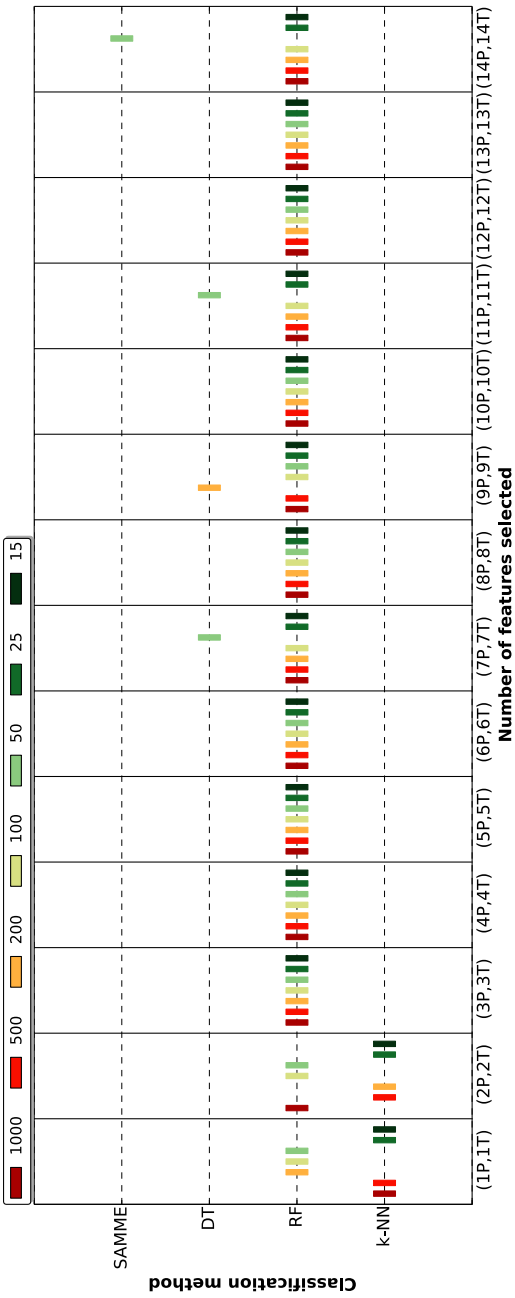


FIGURE 4.7: Classification methods used for classifying the best tuple of features.

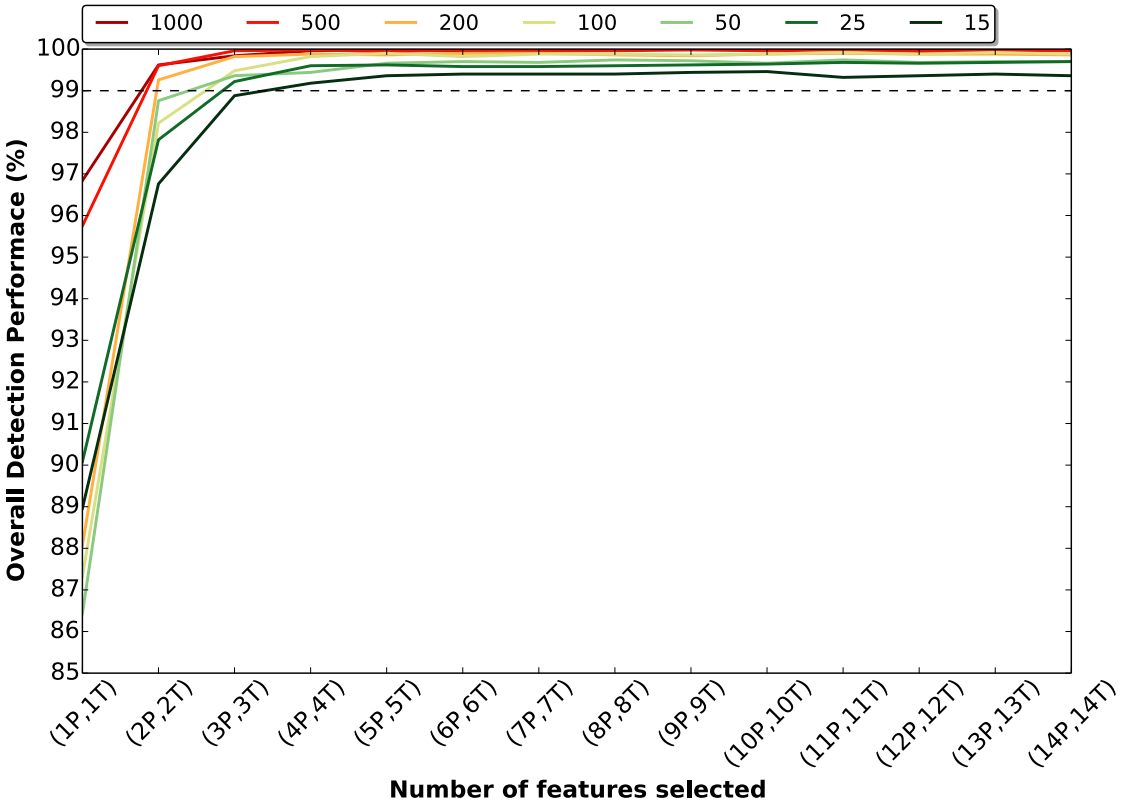


FIGURE 4.8: ODP value related to best combination of feature selection methods and classifiers that give the highest FES for different numbers of pings and features.

TABLE 4.10: Summary of parameters used by the best feature selection methods and classifiers for 25 and 200 pings in the Scikit-learn Python module [1].

Classifier	**Parameter**	**Feature Selection Method / Number of pings**			
		ERT		**L1-FS**	
		25	**200**	**25**	**200**
RF	max_features	sqrt	sqrt	sqrt	sqrt
	n_estimators	500	1000	1000	500
k-NN	k neighbours	4	2	1	1
	weights	distance	uniform	uniform	uniform

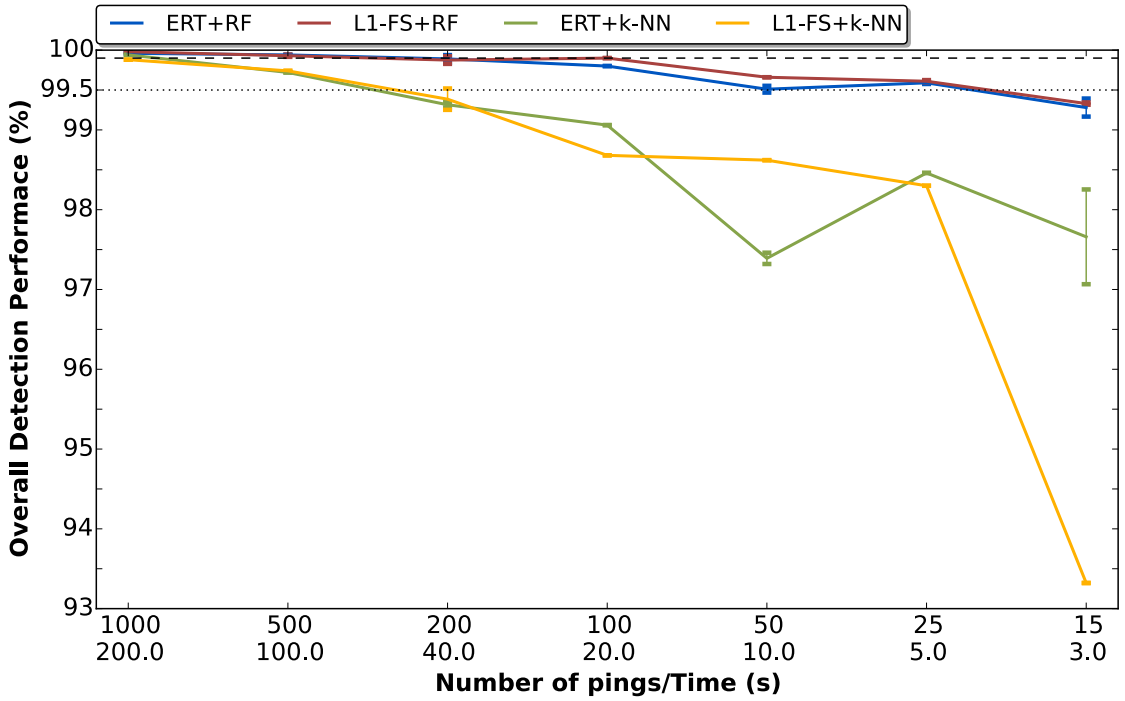


FIGURE 4.9: ODP value related to best combination of feature selection methods and classifiers for different numbers of pings and for the (5P, 5T) tuple.

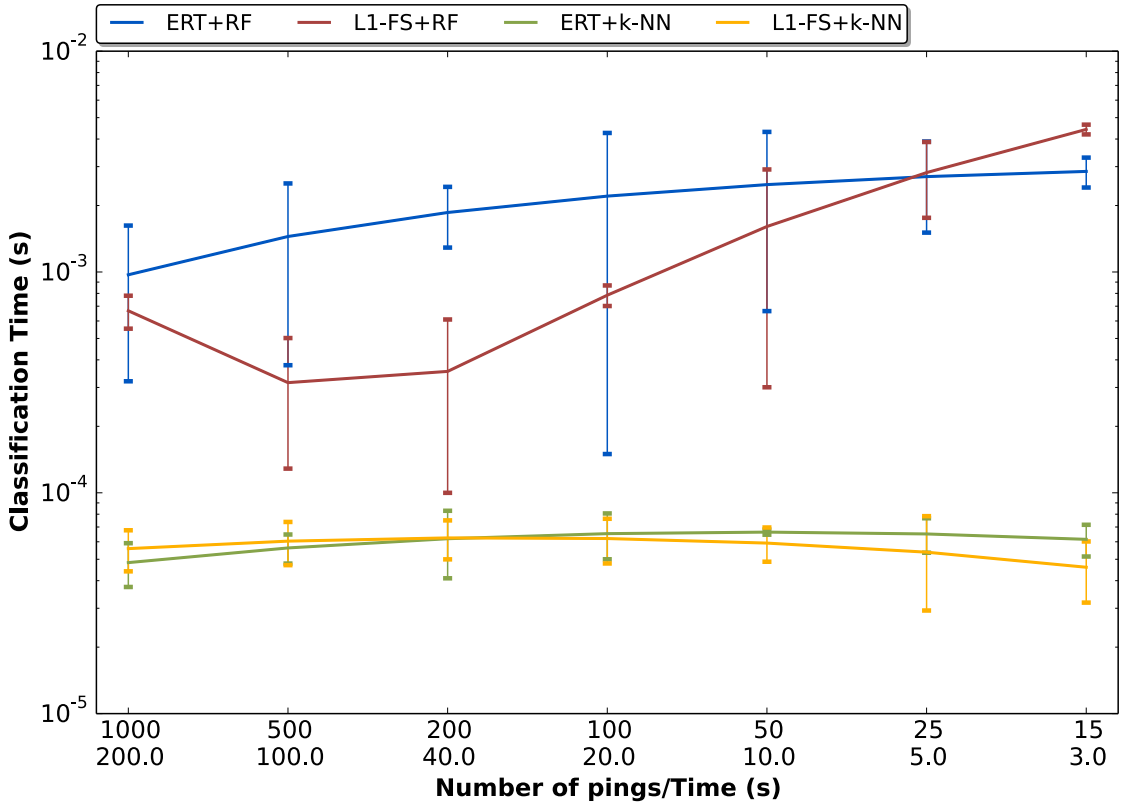


FIGURE 4.10: Time required to classify a sample using the best combination of feature selection methods and classifiers for different numbers of pings and for the (5P, 5T) tuple. The average of the time required with its standard deviation is shown.

As these results demonstrate, it is possible to clearly classify REMs and VESs with an ODP value greater than 99.5% with only 25 pings by using L1-FS and RF. Meanwhile, an ODP value of around 99.9% is reached with 200 pings using ERT and RF. The best 10 features selected by L1-FS are P.min, P.median, P.L, P.SKEW, P.KURT, T.sum, T.μ, T.range, T.L and T.SKEW, whereas for ERT they are P.min, P.sum, P.μ, P.median, P.L, T.sum, T.mode, T.median, T.U and T.r. These features are different because increasing the number of pings generates more data, therefore, some features can increase accuracy during classification, whilst other features can reduce it by introducing uncertainty.

4.3.1 Comparison with MEDA

To evaluate the improvements introduced by the classification methods that are able to properly classify REMs and VESs, a comparison with MEDA is shown in Figure 4.11. To obtain comparable results, the threshold values used by MEDA during the detection are re-evaluated using behaviours from REMs in Table 4.2 (Section 4.2.1). It is quite clear that the classification methods are much better than MEDA by a difference of 13 percentage points using 1000 pings, 21 percentage points using 200 pings and 28 percentage points using 25 pings. Despite the increasing percentage points, it is easy to note that classification methods always give an ODP value above 99%, while MEDA is lower, ranging from 86.5% to 68.7%.

Figure 4.12 compares the average classification time of the classification methods with that of MEDA. While MEDA is shown to be faster than the classification methods by changing the number of pings, the final evaluation score (FES) calculated for MEDA is very low, as more importance is given to the detection of forged embedded machines. For this reason, detection based on the classification methods presented in this chapter is a better solution. This figure also shows that the average time required is dependent on the number of pings. In fact, by using a small number of pings, features extracted could have close values for both REMs and VESs. This aspect will require more computational resources to classify properly forged embedded machines and therefore more time is required. Furthermore, the trend of the average time required by RF using features selected by ERT is logarithmic, while the trend from using features selected by L1-FS is exponential. Features selected by L1-FS help to speed up the classification for almost every number of pings compared to ERT, until the intersection point of 25 pings, at which the time required doubled and increased rapidly.

4.4 Summary

In this chapter, a short summary of current solutions for detecting VESs was presented and their current issues was highlighted. A novel solution based on classification methods for detecting forged embedded machines in the network was presented. Each step needed to properly classify REMs and VESs was described. Results show the high efficiency of this approach in terms of time required for the detection and the high detection

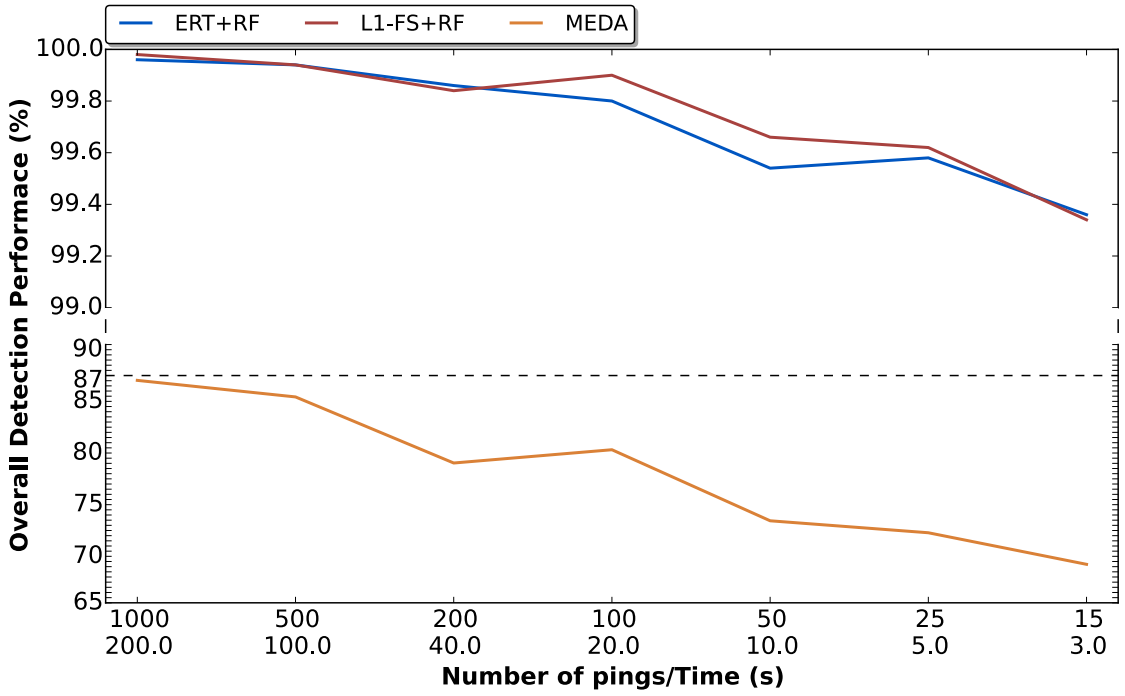


FIGURE 4.11: Comparison of ODP for MEDA and the best classification methods.

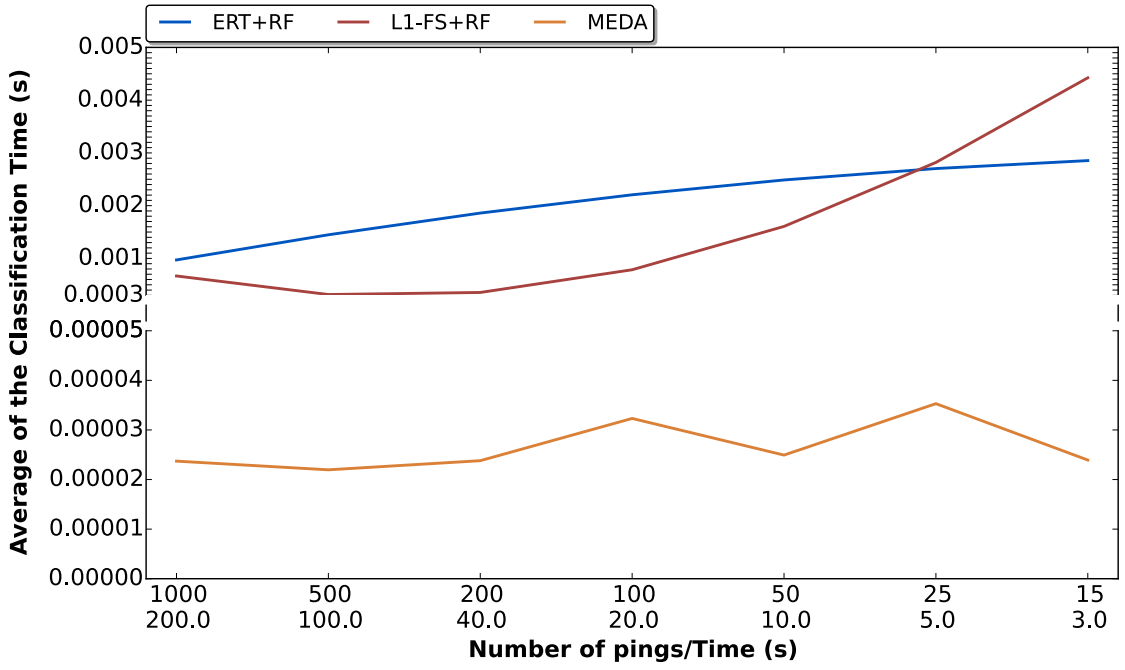


FIGURE 4.12: Comparison of the average classification time for MEDA and the best classification methods.

performance. A comparison with a previously proposed detection method showed that this new approach is better.

It was demonstrated that it is possible for an IoT/M2M-embedded machine to recognise forged embedded machines in only 5 seconds with a detection rate of around 99.5%. Moreover, this detection rate can be increased up to 99.9% by increasing the wait to only 40 seconds. These aspects show that this solution can be easily used by IoT/M2M-embedded machines to eliminate possible threats, especially in relation to power-constrained machines and dynamic networks. Another important aspect of using the proposed method is that it is completely agnostic from the machine architecture and the OS used.

In the next chapter, an attack against these classification-based methods is presented which involves using a powerful machine to mimic behaviours of embedded machines. Moreover, a method of defence is studied and presented in order to detect the attack, thereby enabling recognition of forged embedded machines in the IoT network. Finally, the applicability of this solution to the IoT is discussed.

Chapter 5

Attack and Defence in Behavioural Tests

In the previous chapters, the importance of securing M2M communications from attackers using forged embedded machines in the IoT was presented. Available methods in the literature for detecting virtual and/or emulated embedded systems in the IoT, such as CPU and memory tests, remote tests, architecture-based timing tests and fingerprinting tests, proved to be inapplicable in real scenarios. In this thesis, two behavioural tests based on characterising behaviours of IoT/M2M-embedded machines, both real and forged, were presented and studied. These methods extract timing behaviours from the system using a characterisation algorithm. This algorithm retrieves information related to ping response time (P.) and timestamp values (T.) provided by pinging localhost. Statistical features from P. and T. are then extracted and used for detecting forged embedded machines in the network. The method proposed in Chapter 4 has proven to be efficient and quicker than the method proposed in Chapter 3. However, these methods do not consider an attacker using machines with higher capabilities than embedded machines to mimic the behaviours of real embedded machines. This issue can be used by an attacker to create M2FM communications in the IoT.

In the next sections, an attack against these methods is evaluated and a defence mechanism is designed, tested and compared with existent methods for detecting forged embedded machines. This solution was published by Valerio Selis and Alan Marshall in *Proceedings of the 1st Cyber Security in Networking Conference (CSNet'17)*, under title 'A Fake Timing Attack Against Behavioural Tests Used in Embedded IoT M2M Communications' [111]. The applicability of the proposed solution to real-life IoT scenarios is also discussed at the end of this chapter.

5.1 Threat Model

Thus far in this thesis, real embedded machines (REMs) have been able to properly detect forged embedded machines in M2M communications by assuming that an attacker uses virtual and/or emulated embedded systems (VESs). However, if an attacker

knows that behavioural tests are used for detection, instead of using VESs, it could use a powerful machine to mimic the timing behaviours of an REM, as shown in Figure 5.1. Figure 5.1(a) shows an example of threat model in which the attacker is able to create M2FM communications. In this example, real embedded machines (A and C) will presume that they are communicating with another "real" embedded machine (B). Therefore, these will begin to trust it and will establish an M2M communication. In reality, as shown in Figure 5.1(b), the attacker is using a modified powerful machine in order to be recognised as a real embedded machine without using VESs. This attack is termed as a Fake Timing Attack (FTA) and it is explained in the following section. This kind of attack is detected by following Hypothesis 3.

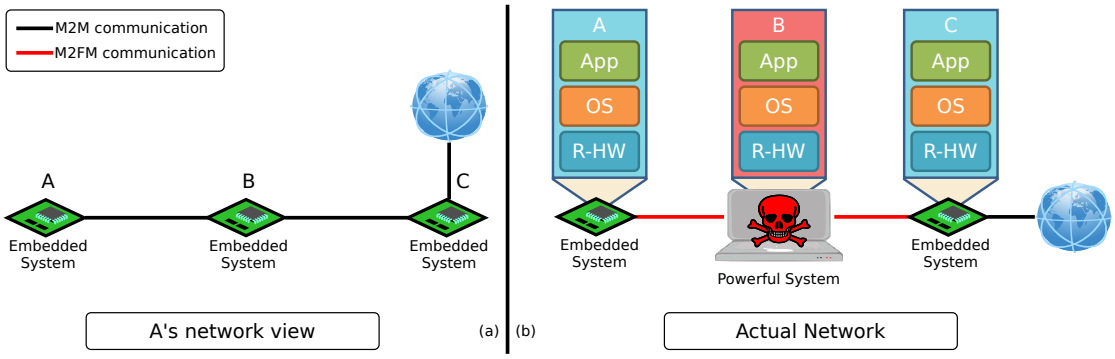


FIGURE 5.1: Attack scenario in which an attacker uses a powerful machine to mimic the timing behaviours of a real embedded machine in order to create M2FM communications. (a) A's view about the network is represented, in which the powerful machine is seen as a real embedded machine. (b) The actual network is shown with the attacker "B" launching the attack against "A" and "C".

Hypothesis 3. An attacker, no matter how powerful his/her computing capabilities are, is not able to perfectly replicate the timing behaviours of a machine.

5.2 Fake Timing Attack

As stated previously, behavioural tests are based on timing information obtained from P. and T. Features from this information are then stored in a dataset and used for training a classification-based algorithm for a later classification of target embedded machines. In this scenario, a clever attacker could use one entry from the training set related to an REM and replicate its ping response time and timestamp values.

The aim of the attacker is to conceal itself by pretending to be an REM in order to create M2FM communications. The only way to achieve this attack is to have the capability to actively modify the kernel of the powerful machine. This can be done by using Linux-based OSs. The rationale behind this attack is to insert delays when a ping request packet is received. This can be achieved because a powerful machine is faster

than an embedded machine, whereas this method cannot be used in VESs because these have timing-related problems as per Hypothesis 1 (Subsection 3.4.1). For these reasons, this attack is called an FTA. An FTA can be very dangerous in scenarios in which there is a medium to extreme level of security (Table 4.8). In fact, information collected by one attacker node may be used against businesses and/or governments, affecting their reputations and security.

An attacker can modify the kernel for the purpose of changing the timing behaviours of a powerful machine, as shown in Figure 5.2. In this figure, three delta times are highlighted as follows:

- ΔT_0: time required for the ping request packet to reach the kernel space from the user space;
- ΔT_1: time required by the kernel to receive and parse the request packet, create a ping response packet, calculate and insert the delay for mimicking the real embedded machine, and finally send the response packet to the user space;
- ΔT_2: time required for the ping response packet to reach the user space from the kernel space.

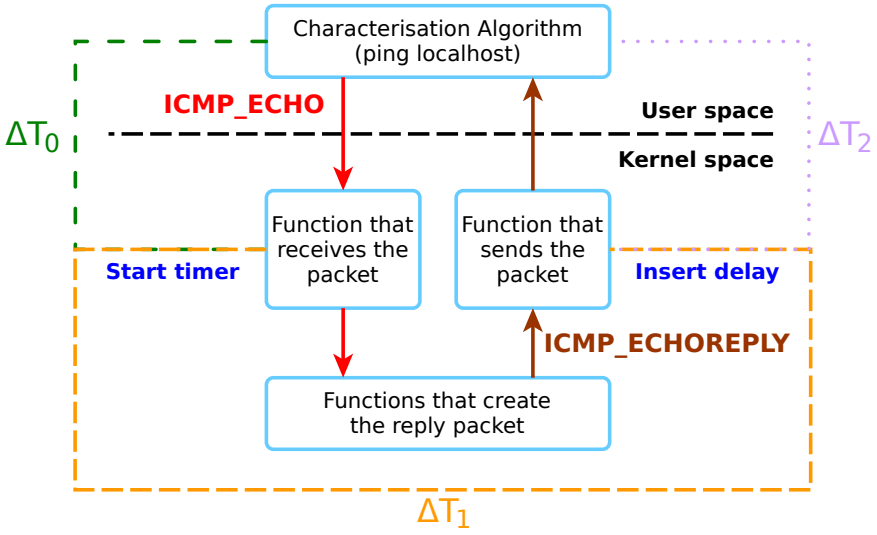


FIGURE 5.2: Kernel modification of a powerful machine for the purpose of faking its timing behaviours in order to mimic behaviours of a real embedded machine.

As Figure 5.2 indicates, the attacker may change its ping response time ($Tping_{Eve}$) by considering the ping response information of an REM ($Tping_{REM}$) and following these steps:

1. Start a timer when the ping request packet (ICMP_ECHO) arrives in the kernel space;
2. Stop the timer before sending the ping response packet (ICMP_ECHOREPLY) to the user space;

3. If $Tping_{Eve}$ is smaller than $Tping_{REM}$, delay sending the response packet by $delay_{Eve}$ milliseconds;

4. Send the ping response packet to the user space.

Therefore, the equations used by the attacker for mimicking the timing behaviours for these steps are:

- The powerful machine response time shown in Figure 5.2 can be expressed as:

$$Tping_{Eve} = \Delta T_0 + \Delta T_1 + \Delta T_2 \tag{5.1}$$

- If $Tping_{Eve} < Tping_{REM}$, the delay that must be inserted before sending the response packet must be:

$$delay_{Eve} = Tping_{REM} - Tping_{Eve} = Tping_{REM} - (\Delta T_0 + \Delta T_1 + \Delta T_2) \tag{5.2}$$

- However, the attacker can only manage ΔT_1 as this is strictly related to the kernel space. Meanwhile, ΔT_0 and ΔT_2 can vary for each ping, making their timing values unknown and unmanageable. Therefore, a clever attacker will estimate them, resulting in an estimated delay as follows:

$$\ominus\,(delay_{Eve}) = Tping_{REM} - [\Delta T_1 + \ominus\,(\Delta T_0 + \Delta T_2)] \tag{5.3}$$

  where:

$$\ominus\,(\Delta T_0 + \Delta T_2) = average\,(Tping_{Eve} - \Delta T_1) \tag{5.4}$$

- From the equations above, it follows that the faked ping response time that the characterisation algorithm will receive is:

$$FakedTping_{Eve} = Tping_{Eve} + \ominus\,(delay_{Eve}) \tag{5.5}$$

It is clear that the attacker will be unable to fake its timing behaviours if $Tping_{Eve}$ is greater or equal to $Tping_{REM}$. The equal condition is valid because the attacker needs at least some microseconds to calculate its ΔT_1, with the result that its $FakedTping_{Eve}$ will always be greater than $Tping_{REM}$. These discrepancies will cause the attack to fail, giving the trustor node an opportunity to detect it. From Equation 5.5 it is possible to show that theoretically Hypothesis 3 is valid, because the faked ping is based on an estimation of ΔT_0 and ΔT_2.

Furthermore, it follows from Equation 5.5, the attacker will have an average trend of ping response times close, but not identical, to the average trend of the REM. Therefore, the data distribution obtained by the attacker will be different compared to that of the REM. In particular, this issue will change the central tendency of the data distribution obtained from the attacker timing behaviours. Specifically, the frequency of values with

the same time and the middle timing value will be different with a high probability, as shown in Figures 5.6(E) and 5.6(F), which show the median and mode ping response time respectively (see Section 5.4 for further details). This leads to Hypothesis 4:

Hypothesis 4. An FTA can be detected using statistical measurements of the central tendency of a data distribution, such as median and mode.

5.3 Detection Model

The process of detecting an FTA can be subdivided into three main parts: (i) characterisation of the behaviours of the target embedded machine, (ii) VES detection and (iii) the actual FTA detection.

The first part involves of the agency in the trustor node "A" which creates an IoT MA and sends it to the agency in the target embedded machine "B". The IoT MA runs the characterisation algorithm shown in Figure 5.3 upon the request is received by the agency in "A". This will provide information about ping response time (P.) and timestamp values (T.) related to the target embedded machine. At the same time, the agency in "A" will start a timer. After, it receives the information about P. and T. from the IoT MA, it will stop the timer and will calculate the elapsed time as per Equation 3.1. If the elapsed time is around 40 seconds (200 pings at 0.2 seconds delay as per Equation 4.6) plus ΔT_{Ch} and ΔT_{RTT}, the second part will be used, otherwise B will be distrusted.

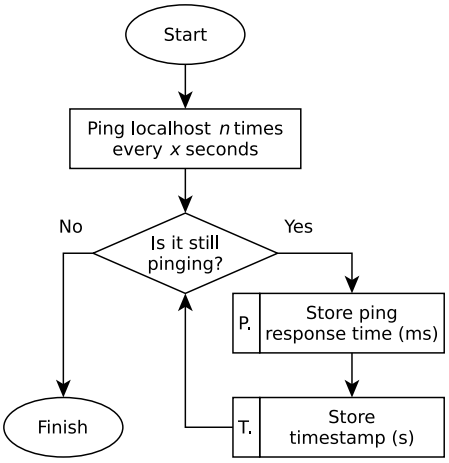


FIGURE 5.3: Characterisation Algorithm in which n is fixed to 200 and x is fixed to 0.2.

The second part involves the classification-based detection algorithm presented in Chapter 4. This uses an RF-trained classifier with the features presented in Table 5.1. If the value of ODP obtained from the classifier is below a threshold value α, B will be distrusted and targeted as VES, otherwise the third part will be used. It is important

TABLE 5.1: Characterisation features used by RF.

Feature	Equation
Minimum (P)	$\min = \min(x_i)$
Sum (P and T)	$sum = \sum_{i=1}^{N} x_i$
Mean (P)	$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$
Lower bounds (95%) (T)	$L = \mu - 1.96\sigma$
Upper bounds (95%) (P)	$U = \mu + 1.96\sigma$
Mode (P and T)	$mode = freq(X)$
Median (P and T)	$median = \frac{1}{2}\left(x_{\frac{N}{2}} + x_{\frac{N}{2}+1}\right)$ of $sort(X)$
Pearson correlation coefficient (T)	$r = \frac{\sum_{i=1}^{N}(x_i-\mu_x)(y_i-\mu_y)}{\sqrt{\sum_{i=1}^{N}(x_i-\mu_x)^2}\sqrt{\sum_{i=1}^{N}(y_i-\mu_y)^2}}$

X = all samples; x_i = i-th sample; N = number of samples;
$freq$ = most frequent value; $sort$ = values in sorted order.

to note that these features were normalised in respect to the normalised dataset used in Chapter 4 (Subsection 4.2.2). This is not feasible in a real-life scenario, because it requires that each machine in the IoT holds a copy of the normalised dataset. However, the normalisation of the dataset is not essential for detecting VESs, as close results are obtained when this step is not taken, as shown in Appendices B.1 and B.2.

Finally, the actual detection of the FTA is carried out using a classification-based algorithm trained with median and mode features as per Hypothesis 4. Doing so, it detects if a target embedded machine is faking its timing behaviours. If the value of ODP obtained from the classifier is below a threshold value β, B will be identified as a powerful machine launching an FTA and therefore will be distrusted. Otherwise, A will begin to trust B with a trust value equal to β, and an M2M communication could be established provided B trusts A.

The final detection algorithm used by an M2M-embedded machine to trust another M2M-embedded machine in the IoT network is shown in Figure 5.4.

5.4 Simulations, Results and Comparison with Other Algorithms

Simulations are used to determine if the proposed previous algorithms are able to detect an FTA. They are also performed to evaluate if Hypotheses 3 and 4 are valid (Sections 5.1 and 5.2). The new detection model subsequently is tested and the results from the simulations are shown. To evaluate the algorithms, a Linux-based OS with a modified kernel running on a workstation with quad-core i7@2.70GHz CPU and 6 GB of RAM is used. The kernel is modified by inserting delays, calculated using Equation 5.3,

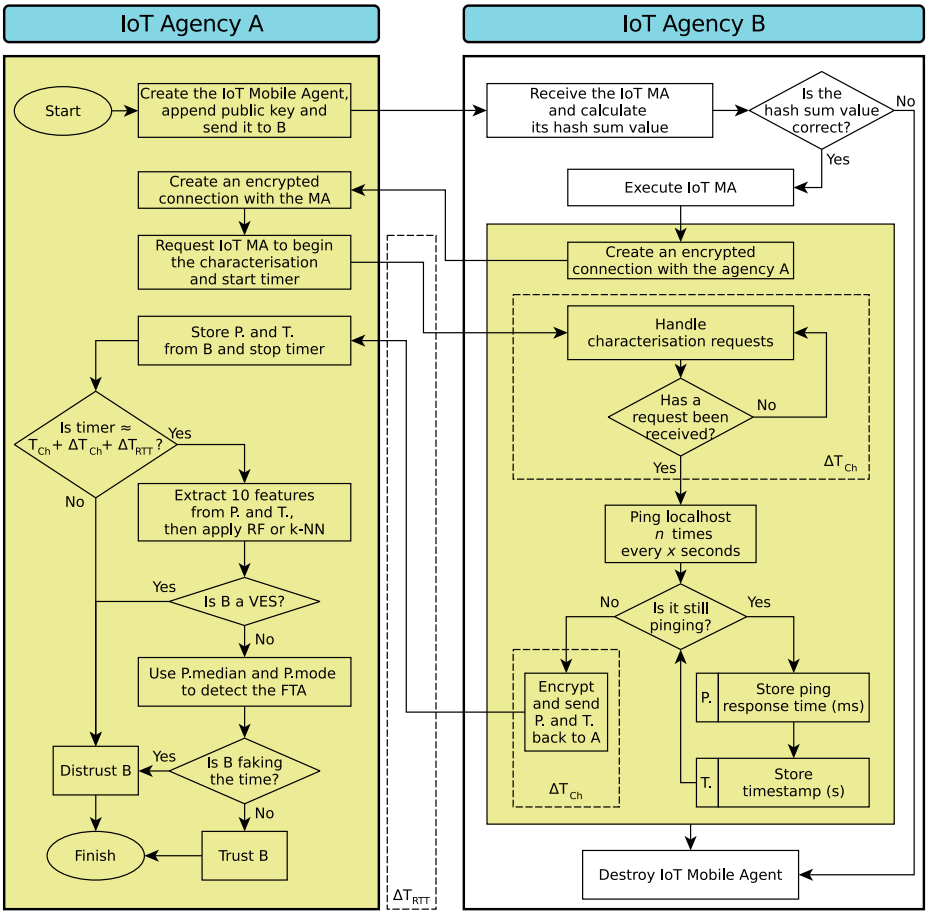


FIGURE 5.4: Model used for detecting a FTA in which n is fixed to 200, x is fixed to 0.2 and T_{Ch} is fixed to 40 seconds.

before a response ping packet is sent to the user space, as shown in Figure 5.2. Before modifying the kernel, Equation 5.4 was used to estimate $\Theta(\Delta T_0 + \Delta T_2)$ by performing 1000 characterisation tests. As a reference REM, a characterisation test present in the dataset from an Arduino Yún was used for modifying the kernel.

Timing behaviours obtained by performing the characterisation tests from the powerful system, with and without the kernel modifications, are shown in Figure 5.5. In this figure, the timing behaviours of the reference REM are shown for completeness. As expected, the average timing behaviour of the powerful machine with the modified kernel is close to that of the Arduino Yún. It is also clear that the timing behaviours with the modified kernel are very unstable with high standard deviation spikes in respect to the mean, varying between 0.01 and 0.05 ms.

The characterisation test results obtained by launching an FTA were evaluated against those produced by MEDA (Chapter 3), and RF and k-NN trained classifiers with the best 10 features from ERT used in Chapter 4. Results obtained from these evaluations show that none of the VES detection methods proposed is capable of detecting an FTA. In fact, all these methods return a similarity value as output for a VES equal to 0% for each characterisation test. Therefore, an attacker using a powerful

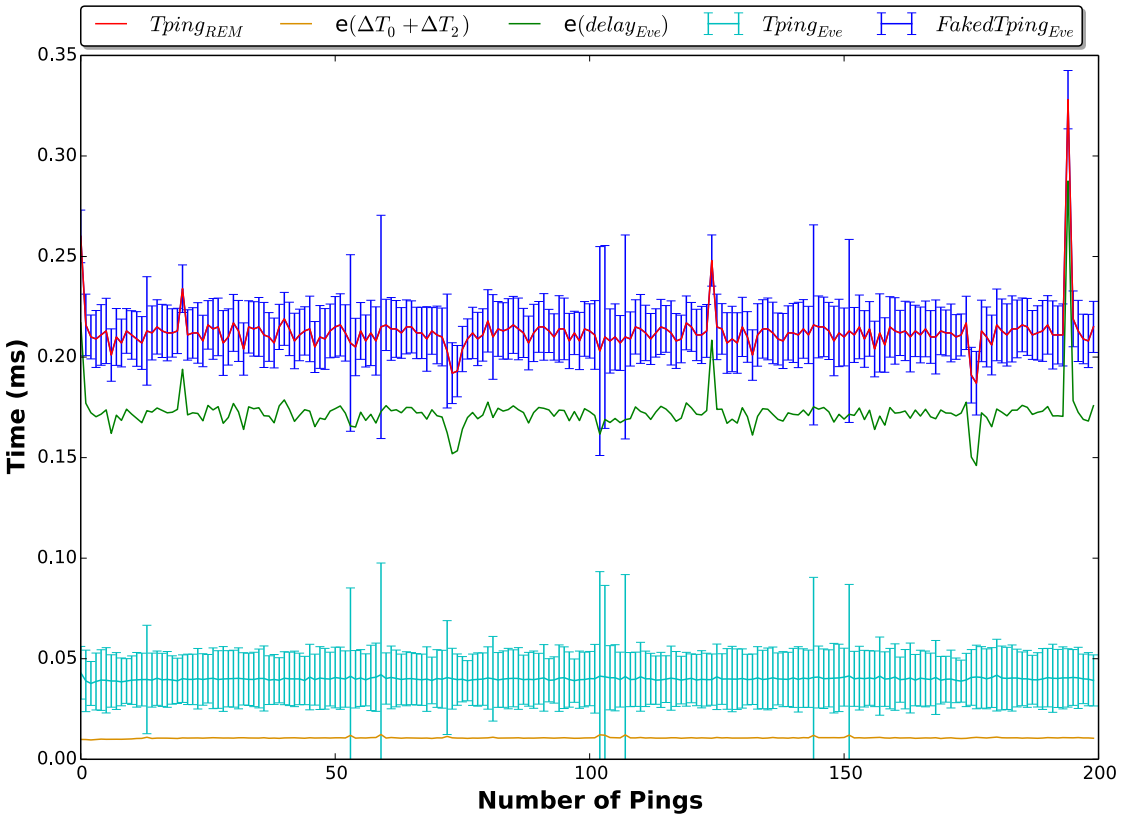


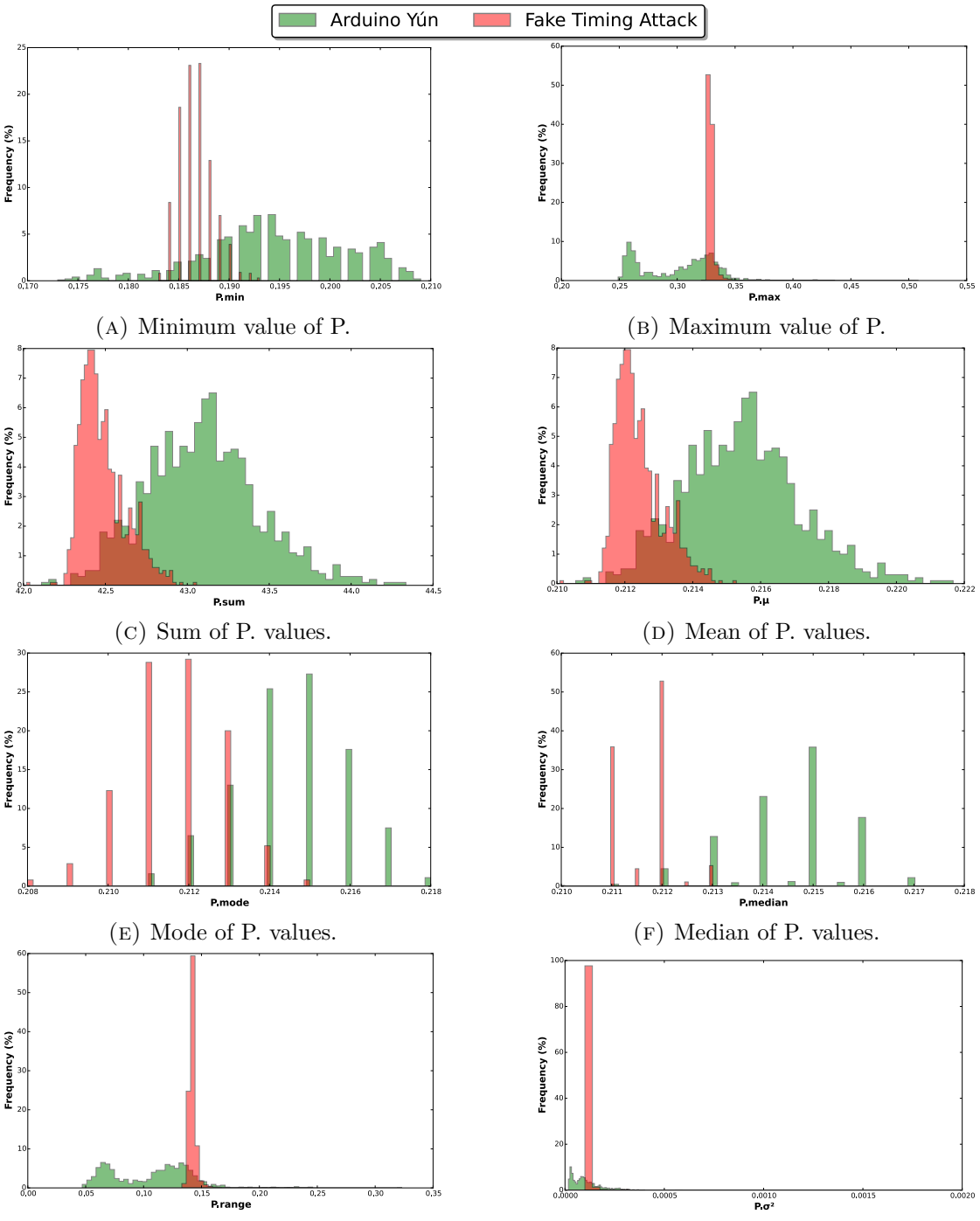
FIGURE 5.5: Timing information related to the Arduino Yún and the FTA for 200 pings. The ping response time of the Arduino Yún used as the reference REM is shown in red. The estimation of the sum of ΔT_0 and ΔT_2 used by a possible attacker to modify its ping response time is shown in orange. The estimated delay introduced by the attacker in the kernel is shown in green. The ping response time of the powerful machine is shown in light blue. The faked ping response time obtained from the powerful machine launching the FTA is shown in blue.

machine and launching an FTA is recognised as an REM all the time and will receive a trust value of 100%.

An extensive evaluation of the features (Table 4.3) extracted from P. after the characterisation tests are carried out is presented in Figure 5.6. In this figure, cumulative frequency histograms are shown for P. features from the Arduino Yún and the powerful machine launching the FTA. The corresponding histograms for T. features are not shown here (see Appendix B.3) as these are similar for the Arduino Yún and the powerful machine launching the FTA, meaning that the FTA does not affect the timestamp values.

The histograms of most of the features obtained from the powerful machine launching the FTA overlap the corresponding Arduino Yún histogram. Therefore, these features cannot be used to detect an FTA. Meanwhile, histograms from P.mode and P.median features mostly do not overlap each other, as shown in Figures 5.6(E) and 5.6(F). It is clear that these features can be used to detect an FTA, as stated in Hypothesis 4.

From the information obtained, RF and k-NN are trained by using P.mode and P.median features. The dataset and the same steps presented in Section 4.2 are used for



(A) Minimum value of P.

(B) Maximum value of P.

(C) Sum of P. values.

(D) Mean of P. values.

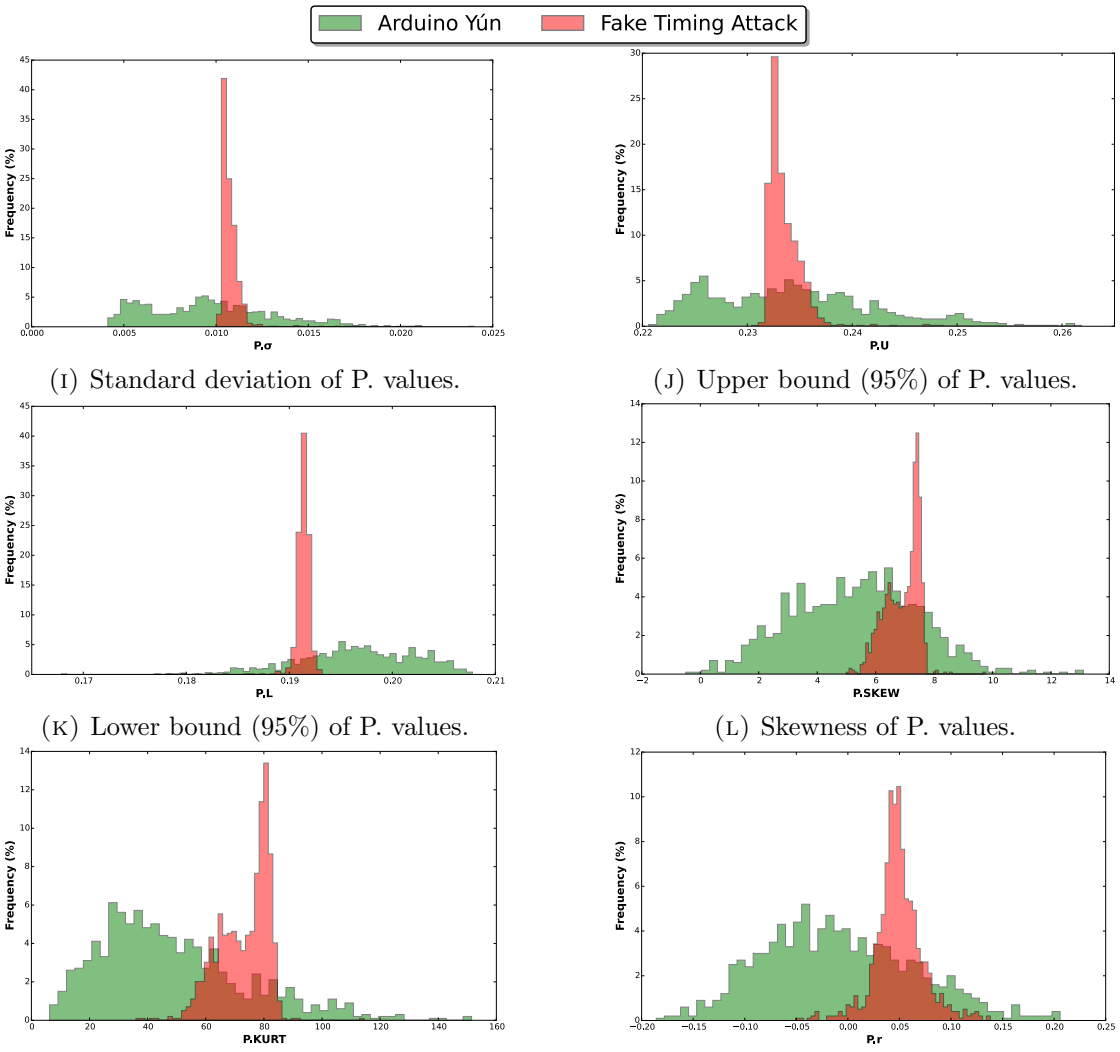
(E) Mode of P. values.

(F) Median of P. values.

(G) Range of P. values.

(H) Variance of P. values.

FIGURE 5.6: Cumulative frequency histograms for P. features obtained from the Arduino Yún and FTA for 200 pings and 1000 characterisation tests.



(I) Standard deviation of P. values.

(J) Upper bound (95%) of P. values.

(K) Lower bound (95%) of P. values.

(L) Skewness of P. values.

(M) Kurtosis of P. values.

(N) Pearson correlation coefficient of P. values.

FIGURE 5.6: (continued) Cumulative frequency histograms for P. features obtained from the Arduino Yún and FTA for 200 pings and 1000 characterisation tests.

training these classifiers, as highlighted in Figure 4.2. Results obtained from algorithms for detecting forge embedded machines are shown in Table 5.2. Here, the last four rows show the results for detecting the FTA obtained by combining the classification-based algorithm proposed in Chapter 4 and the trained classifiers with P.mode and P.median features. It is possible to show that RF for step (ii) and k-NN for step (iii) (Section 5.3) give the best results for correctly detecting VESs and the FTA. This confirms that Hypothesis 4 is correct and that it is possible to detect an FTA with a high accuracy. Therefore, IoT/M2M-embedded machines can use the proposed solution to securely exchange information in the network.

TABLE 5.2: Comparison of methods for detecting forged embedded machines in the IoT.

Algorithm	**Forged Embedded Machine Detection (%)**		
	REM	**VES**	**FTA**
MEDA [6]	21.92	78.32	0
k-NN* [110]	0.68	99.32	0
RF* [110]	0.14	99.89	0
k-NN*+k-NN** [111]	0.68	99.32	93.70
k-NN*+RF** [111]	0.68	99.32	61.11
RF*+k-NN** [111]	0.14	99.89	93.70
RF*+RF** [111]	0.14	99.89	61.11

*Classification-based algorithm presented in Chapter 4;

**RF and k-NN trained with P.mode and P.median features.

5.5 Classification of Unknown Devices

The last point to be determined is if the proposed algorithm is capable of detecting REMs and VESs that are unknown. This is the same as detecting black-box architectures. In fact, all behavioural tests proposed in this thesis are based on trained algorithm within input REMs and VESs behaviours. To determine if these are capable of detecting REMs and VESs that are not present in the initial dataset, unknown embedded systems (UESs) are tested (Table 5.3). These UESs are chosen on the basis of their difference from the REMs and VESs present in the initial dataset. The Google Nexus 5X has a six-core CPU with a 64-bit architecture, whereas other Android-based devices in the initial dataset had fewer cores with a 32-bit architecture. Whereas other VESs in the initial dataset run under a Linux-based host system, the VirtualPC runs under a Windows-based host system. Meanwhile, the VMware EXSi does not run under another OS, but uses a hypervisor which runs directly on the real physical hardware. Finally, the Zsun WiFi Card Reader is a very simple card reader which shares files via Wi-Fi, whereas other REMs in the initial dataset were mostly development boards, smartphones and tablets with more resources and capabilities. Furthermore, this card reader has a completely different OS compared to other REMs in the initial dataset.

Figure 5.7 shows a complete summary of results obtained from all behavioural tests proposed by including the classification of UESs. Results show that the final detection algorithm proposed is not only capable of detecting VESs and the FTA, but also is able to recognize REMs and VESs that were not present in the initial dataset. This proves that the final detection method shown in Figure 5.8 is completely independent of the machine architecture and its OS.

TABLE 5.3: List of unknown real and virtual embedded systems.

System	Architecture	Operating System	Type
Google Nexus 5X	ARM	Android	REM
VirtualPC	Embedded x86	OpenWRT	VES
VMware EXSi	Embedded x86	OpenWRT	VES
Zsun WiFi Card Reader	MIPS	Zsun OS	REM

REM: real embedded machine; VES: virtual or emulated embedded system.

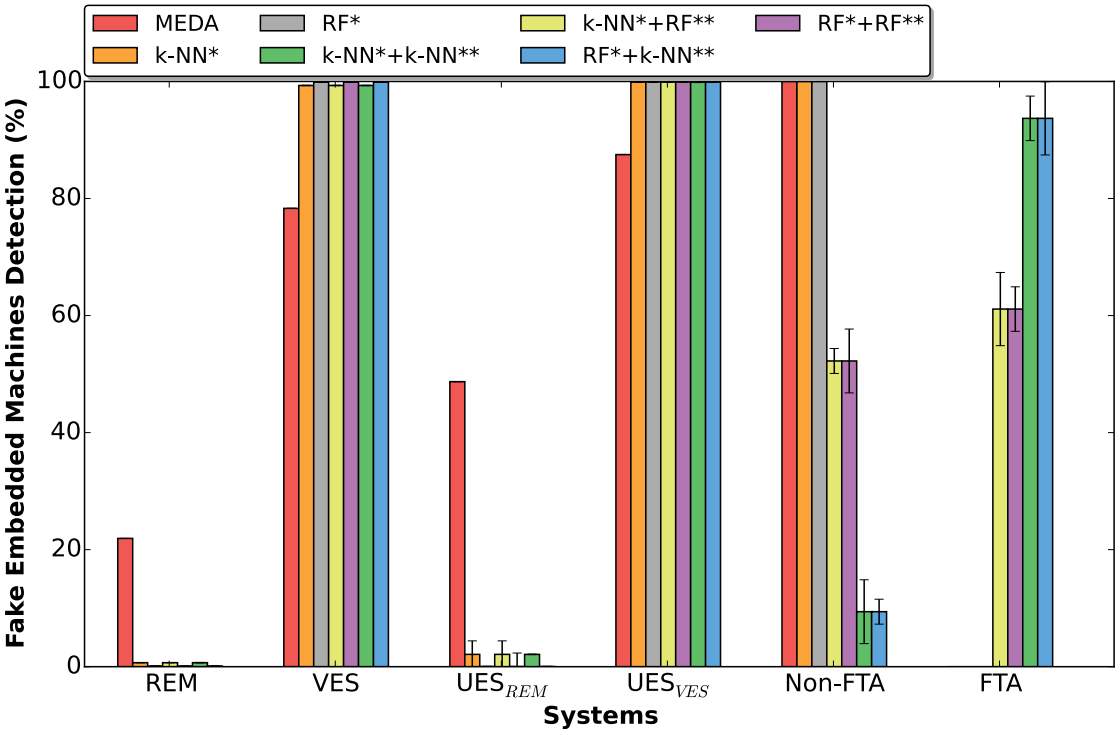


FIGURE 5.7: Summary of results for all behavioural tests for 200 pings. *Classification-based algorithm presented in Chapter 4; **RF and k-NN trained with P.mode and P.median features.

5.6 Applicability of the Proposed Solution

In the follow subsections, the application of this solution with some architectural reference models for M2M communications in the IoT system is presented. How this solution can protect M2M devices in IoT real-life scenarios is highlighted.

5.6.1 Architectural Reference Models

In the IoT there is necessity to incorporate trust management in all layers in the architectural reference model. The final proposed solution presented in this thesis can be applied in European Telecommunications Standards Institute (ETSI), oneM2M and ITU as well as emerging trends in IoT such as Social IoT (SIoT), Virtualisation Continuum (VC) and Fog Computing (FC).

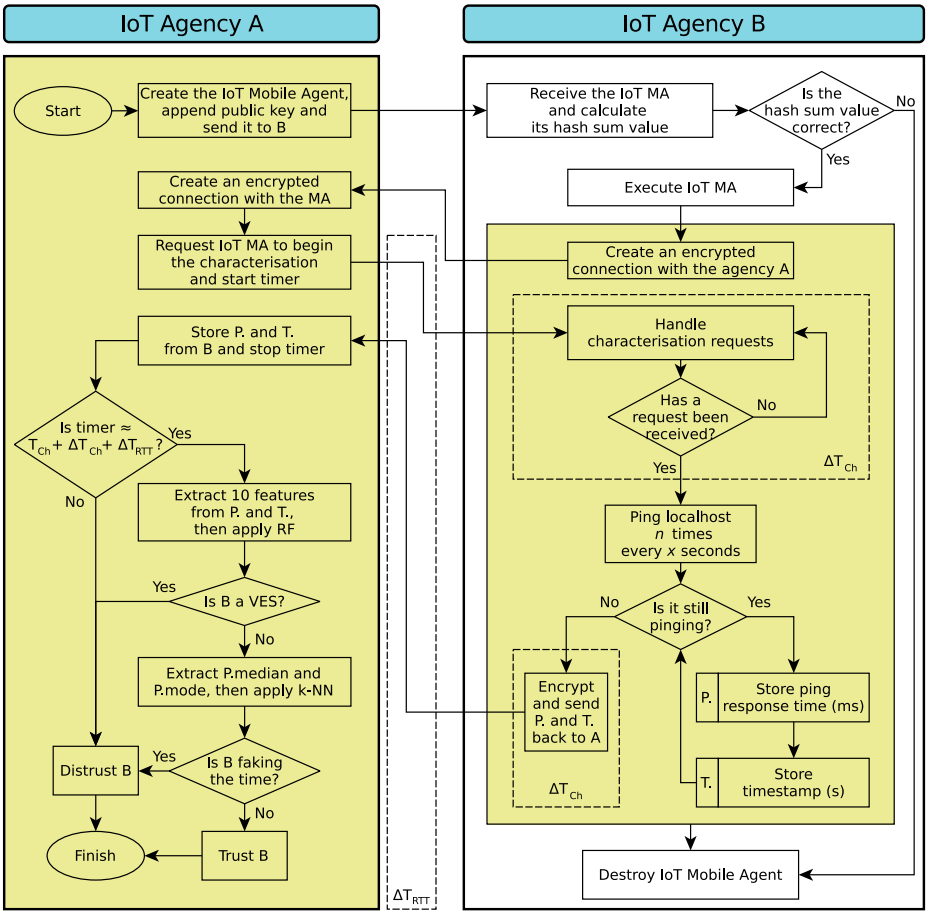


FIGURE 5.8: Final detection algorithm for detecting VESs, UESs and FTAs, in which n is fixed to 200, x is fixed to 0.2 and T_{Ch} is fixed to 40 seconds.

The final proposed solution can be easily included in the ETSI and oneM2M architectural reference models. This will require the use of IoT trust agencies in the M2M Device and Gateway Domain as part of M2M Applications. In the M2M/ITU architectural reference model, this solution can be incorporated as part of the Specific Security Capabilities targeting the Device layer. The functionality to request/receive the characterisation can also be included in both the M2M Network Domain (ETSI and oneM2M) and Network layer (ITU). This allows the IoT core to understand if the M2M-embedded machines that are communicating are real or forged.

The final proposed solution can also be applied to the SIoT as part of the Social agents which are already in the Gateway and Object. Moreover, it also can be incorporated into the Object interface solely for the purpose of requesting/receiving the characterisation. In the VC, the IoT trust agencies can be included in the Physical layer inside objects. Virtual objects in the Virtualisation layer can request/receive the characterisation to ensure that objects in the Physical layer are real. This will allow Virtual objects to represent only functionalities of real embedded machines. Lastly, the final proposed solution can also be part of the FC in the Device layer and Fogs can request/receive the characterisation for trusting objects in the Device layer.

An overview of the IoT trust agencies that can request/receive the characterisation in the IoT core and the IoT trust agencies and IoT mobile agents that can request/receive and perform the characterisation in IoT/M2M devices is shown in Figure 5.9.

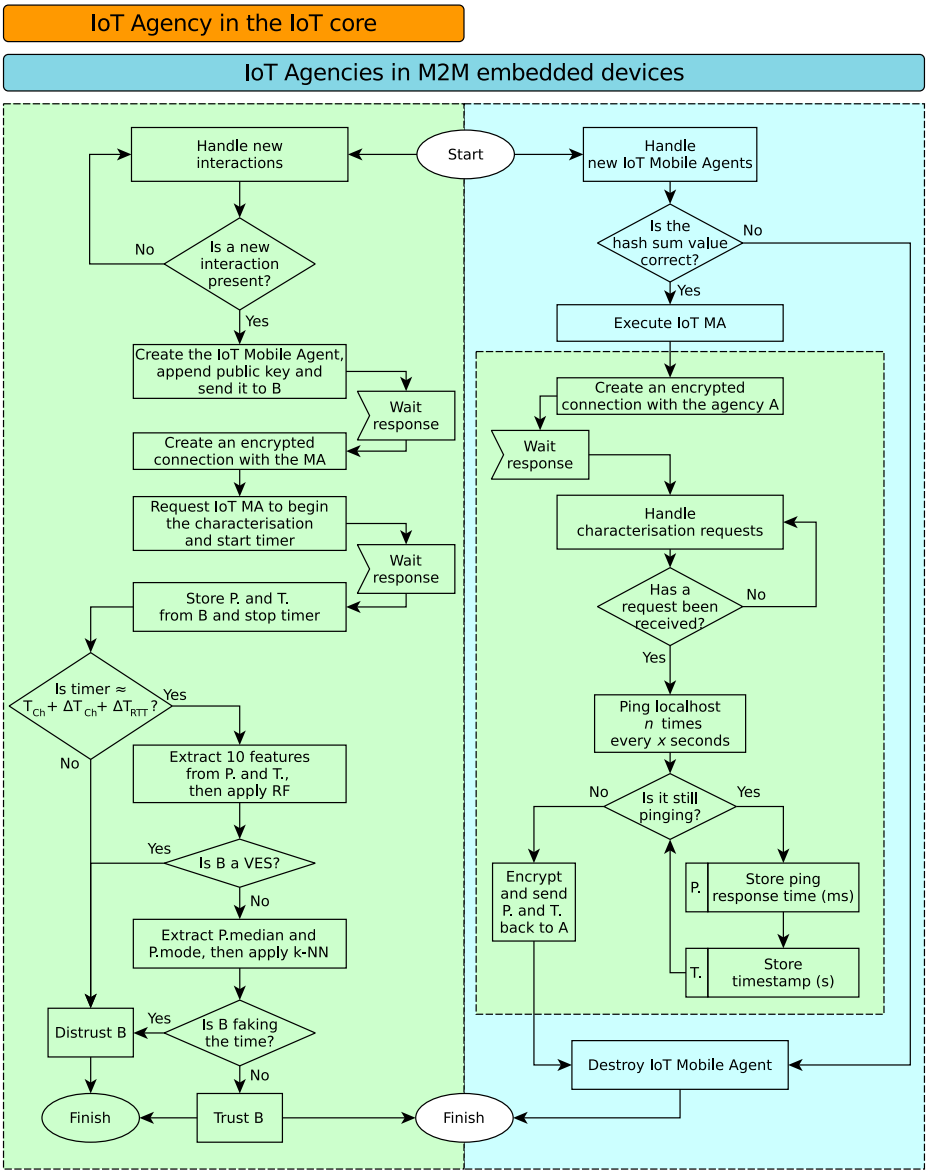


FIGURE 5.9: Final detection algorithm in IoT trust agencies for both the IoT core and IoT/M2M-embedded devices, in which n is fixed to 200, x is fixed to 0.2 and T_{Ch} is fixed to 40 seconds.

5.6.2 Implementation Feasibility

In this thesis, the proposed characterisation algorithms are based on the ping command. This command is available in all IoT-embedded machines with the Internet protocol suite implemented. These constitute the majority of embedded machines available in the IoT. An example of IoT-embedded machines that do not support the characterisation algorithm are Bluetooth Low Energy (BLE)-based sensors. However, these devices

are currently not smart enough to implement advanced security mechanisms or to communicate directly to the IoT core.

For the purpose of this thesis, the characterisation algorithm presented is a bash script running under Linux-based OSs (Appendix A.1), whereas MEDA, the classification-based algorithm and the FTA detection algorithm are Python-based scripts (Appendices A.3 and A.4). An example of output obtained by performing the characterisation algorithm is shown in Appendix A.2.

In order to incorporate these algorithms in an IoT trust mobile agent and to use them in real IoT scenarios, practical considerations are needed. Firstly, the proposed final solution must be part of the IoT standard. Secondly, trusted third party entities will be required to certify the agent. This will give IoT agencies the opportunity to validate IoT mobile agents before performing the characterisation process. This procedure can be implemented quite easily in embedded machines with an open-source OS, whereas for proprietary embedded machines and OSs, each company must provide this certification agent as part of their systems in order to be compatible with the standard.

The characterisation algorithm and MEDA can be easily used in almost all IoT-embedded devices with a very small amount of memory, processing capability and power. This is because these algorithms use basic mathematical functions and do not require any previous knowledge to detect forged embedded machines.

The classification-based algorithm and the FTA detection algorithm are based on pre-trained classifiers such as k-NN and RF. Authors in [253] proposed a k-NN algorithm for an embedded machine with very low computation capabilities and resources, like a microcontroller for real-time processing with a 32-bit ARM Cortex M3 processor at 72 MHz, with 512 KB of flash and 64 KB of RAM. This machine was also equipped with five sensors: accelerometer, magnetometer, altimeter, temperature and force. Results from their proposed solution show that the dimension of training and test sets is the key to reducing the time and memory required for the classification. The latency introduced by their proposed k-NN algorithm is 950 μsec with 9 KB of RAM, in which k is equal to 5. A training set with 50 instances and a test set with 6 instances were used as parameters for obtaining accuracy, which in this case was 92.7%. Authors in [254] showed an implementation of RF for ARM- and FPGA-based architectures used in IoT-embedded machines. Results for the embedded ARM show that for a pre-trained forest composed of 50 decision trees, with 1349 nodes and around 675 paths between root and leaf nodes, on 6000 training instances, 1 MHz CPU is needed to evaluate 300 samples per second (around 3.3 kHz per sample). The flash memory required for storing the pre-trained RF was in the order of a few megabytes. Therefore, it is clear that these classifiers may require somewhat greater computational capabilities and relatively larger memory resources compared to MEDA.

These aspects highlight that the proposed solution for detecting forged embedded machines based on the characterisation algorithms, RF and k-NN, can be implemented

in IoT-embedded machines with low computational capabilities and a small amount of memory.

5.6.3 Real-Life Scenarios: Applicability

The applicability of the final proposed solution to real-life scenarios allows Intelligent Transportation Systems (ITS), Intelligent Healthcare Systems (IHS) and Intelligent Building Systems (IBS) to establish trust relationships among IoT/M2M-embedded machines in both open and closed networks.

Intelligent Transportation Systems

Attackers that aim to disrupt the traffic by using multiple forged embedded machines can be detected by using the final proposed solution. IoT mobile trust agents in vehicles and IoT mobile trust agents in the ITS core will be able to detect these forged machines. This detection can, for example, block false traffic information sent by an attacker to other vehicles and the ITS. Therefore, by distrusting this information, the ITS will continue to work properly, suggesting the best route, preserving road safety and providing the best in-vehicle entertainment.

Intelligent Healthcare Systems

The IHS and IoT-embedded machines such as smart body sensors, emergency vehicles etc., can be protected by using the final proposed solution. With this, smart body sensors and their controllers could detect the forged embedded machines attackers use to send false body information. The IHS core could also distrust health information from forged embedded machines, and therefore could send alerts to hospitals, patients and emergency vehicles only during actual emergencies.

Intelligent Building Systems

The final proposed solution can also be used by the IBS. Smart building sensors and actuators, and the IBS core could be able to detect possible attackers and distrust their information. By detecting attacks, the IBS could, for example, provide the perfect room conditions and the perfect occupancy of specific areas in the building and it will only alert in the case of a real fire.

5.7 Summary

In this chapter, an attack against behavioural tests was presented. This was termed a Fake Timing Attack (FTA). This FTA involved using timing information from a real embedded machine (REM) from the initial dataset in order to modify the kernel of a powerful machine for the purpose of mimicking its timing behaviours.

A detection model for uncovering the FTA based on the theoretical analysis gathered by observing the modified kernel behaviour was then outlined. Metrics available for detecting virtual and/or emulated embedded systems (VESs) were further analysed to determine if the theoretical analysis was correct. The mode and median features from the ping response time were then used to detect the FTA. A comparison of two classifiers, k-NN and RF, trained by using these features for detecting an FTA was provided. Results show that k-NN gave the best detection, with an overall detection performance (ODP) of 93.70%. The final detection algorithm is able to recognise VESs and detect FTAs in around 40 seconds, making it easily applicable to IoT/M2M-embedded machines.

Furthermore, the final detection algorithm was tested with newer REMs and VESs with completely different capabilities from the REMs and VESs present in the initial dataset. This was done in order to determine if the algorithm was capable of detecting unknown embedded machines (UESs) without previous knowledge of them. Results show that the final detection algorithm can recognise unknown REMs and VESs, with an overall detection performance of 99.96% and 99.92% respectively. This proves that the final detection algorithm is capable not only of recognising VESs and detecting FTAs, but also of recognising UESs. This makes it easily applicable to the detection of embedded devices with black-box architectures and therefore to future IoT/M2M-embedded machines.

Finally, the applicability of the final proposed solution to proposed IoT/M2M architectural reference models, current trends in the IoT and real-life scenarios was shown. This confirmed that it can be easily applied to the detection of forged embedded machines in the IoT, and also underlines its benefits in terms of trusting information received by M2M-embedded machines.

In the next chapter, the conclusions of this thesis are provided. Contributions and findings are also presented. Moreover, future works required to improve the way IoT/M2M-embedded machines can successfully trust each other are suggested.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The Internet of Things (IoT) is considered one of the most prominent Information and Communication technologies to be used on a large scale. At its base there is the communication among objects for achieving specific and complex goals. These objects rely on Machine-to-Machine (M2M) communications in order to exchange information. The nature of IoT/M2M devices is mostly embedded with low computational capabilities, and this increases their vulnerability. Therefore, a lack of security can dramatically degrade the operation of an application system. This may have very negative effects, including putting peoples' lives at risk. Trust has been studied and has been identified as a mechanism for securing M2M communications in the IoT.

In this thesis, a new solution is proposed to increase IoT security that can be easily applied to the existing architectural reference models. Unlike current research on incorporating trust into the IoT system, this new solution is applicable to real IoT scenarios. Moreover, this new solution can address a new and emerging threat against M2M communications. This new threat was identified in this thesis and consists of using forged embedded machines. These machines can be used for launching several simultaneous misbehaviour attacks against trust management systems in order to disrupt the network. It has been shown that the new solution can be applied to existing and proposed standards for incorporating trust in the IoT.

Several research outputs for improving the security of M2M communications in the IoT have been presented in this work:

- A study of the IoT ecosystem, identifying embedded machines as its critical part;
- An investigation into previously unconsidered threats against M2M communications in the IoT;
- The development of a characterisation algorithm for collecting reliable embedded machine behaviours;

- A study of embedded machine behaviours in order to obtain useful metrics for creating trust relationships;
- The creation of two datasets with behaviours from real, virtual and emulated embedded systems, which is the starting point for achieving reliable results;
- Implementation of methods for detecting newly discovered threats;
- A study of an attack against the proposed detection methods, along with a protection mechanism for its detection.

The results presented in Section 5.6 show that the final solution can be used in all the proposed architectural reference models, making its applicability independent of them and easily integrated during the standardisation process.

6.2 Contributions and Findings

A key contribution is that the methods proposed in this thesis are completely agnostic to the machine type, its characteristics and the OS used. This is fundamental for their application in present and future IoT-embedded systems, because it is not feasible to know machine-related information a priori, as this can change and develop over time. Furthermore, the adoption of these methods can be incorporated in IoT mobile trust agents. This allows information to be secure directly as part of the smartness of the object. In summary, findings from this thesis show that it is possible to establish reliable communications among IoT/M2M-embedded machines in the presence of forged machines. Details of the main conclusions from this thesis are summarised in the following subsections.

6.2.1 Introduction, Background and Related Work

This research began by identifying the characteristics of objects in the IoT in terms of connectivity, computational capabilities etc. By analysing several network topologies used for creating M2M communications, it became clear that linear topologies give attackers the opportunity to easily intercept information.

The analysis of real-life scenarios with threats against "things" shows the importance of securing the system and protecting these objects. The study of architectural reference models shows that there is a lack of standardisation in relation to the IoT. This reduces adoption of the IoT in large-scale, real scenarios and makes implementing security measures challenging.

An investigation of attacks against M2M devices has highlighted why security and trust play an important role in M2M communications. The evaluation of current solutions for incorporating trust in the IoT shows that these cannot be applied in real IoT scenarios. There are several reasons why these solutions cannot be applied. Firstly, most employ assumptions that are unrealistic in real environments, such as pre-trusted and

invincible objects, objects able to overhear the traffic of its neighbours etc. Secondly, most of them can be applied only in static networks, which limit their applicability in many real IoT scenarios. Furthermore, most require objects with memory capabilities that are rarely present in real IoT-embedded machines. Finally, and most importantly, none of these solutions is able to distrust forged objects in the IoT network.

6.2.2 A New Threat and a Novel Solution: Machine Emulation Detection Algorithm

A new threat against M2M communications has been identified, showing a lack of protection in current TMFs in the IoT. This attack, which uses forged embedded machines, gives attackers the opportunity to create a new subverted type of communication, identified as Machine-to-Fake Machine (M2FM) communication. This finding shows how real-life scenarios can be attacked and how the IoT system can be compromised.

Therefore, a first detection mechanism against this attack has been identified by studying embedded machine behaviours. Lack of previous studies and available datasets show the urgency to protect the IoT system. The initial study of machine characteristics, capabilities and network connectivities, were used for identifying trust metrics related to the behaviours of machines rather than specific characteristics. This led to a new method for characterising these behaviours and then using them as threshold values for the detection. Results show that this mechanism is able to detect forged embedded machines within 3 minutes with a detection accuracy of 79.21% [6].

6.2.3 A Classification Approach to Detecting Illegitimate Embedded Machines

The long time required by the first detection method (MEDA) led to a search for an alternative approach to improving the characterisation of embedded machine behaviours. The moderate detection uncertainty and low recognition of real embedded machines also led to the identification of a new approach for detecting forged embedded machines.

The new method employs machine learning algorithms, which can use more than one behaviour metric per time, greatly improving detection. A new evaluation method for understanding which algorithm behaves correctly was therefore proposed. This shows that k-NN performs well when a few trust metrics are used and that RF performs well in almost all the other cases. Results show that by using RF with 10 metrics, it is possible to detect forged embedded machines with an overall detection performance (ODP) greater than 99.5% within only 5 seconds; this can rise to around 99.9% within 40 seconds [110].

6.2.4 Attack and Defence in Behavioural Tests

The two detection methods proposed in this thesis collect behaviours from real embedded machines and use these to detect forged embedded machines. This led to the study of

a possible attack in which real embedded machines are mimicked by replicating their behaviour inside a modified powerful machine. This study shows that previous detection methods can be defeated by perpetrating this attack, referred to as fake timing attack. A detailed analysis of the trust metrics retrieved by launching the attack revealed that two metrics, mode and median from ping response time, are critical in detecting this attack. These metrics were used by k-NN for the detection, giving an ODP value of 93.70% within 40 seconds.

Further evaluations demonstrate that RF is able to recognise real and forged embedded machines that were previously not present in the initial dataset. This aspect shows that the proposed solution can be easily applied to future IoT/M2M-embedded machines and that it is completely agnostic in relation to the machine's architecture and OS. Results show that the proposed solution can detect unknown forged embedded machines with an ODP value of around 99.9% within 40 seconds.

Finally, the combination of RF and k-NN for detecting known and unknown forged embedded machines, and the fake timing attack can be completed within 40 seconds with an ODP value of 99.89%, 99.92% and 93.70% respectively. These results show that this solution can be used to create reliable trust relationships among M2M devices in the IoT. By using this solution as a method of pre-trust evaluation of IoT/M2M-embedded machines, trustor machines will save energy and time [111].

6.3 Future Work

The findings in this thesis clearly indicate that the IoT is not yet ready to be deployed. However, it clearly is being deployed, therefore attackers can exploit its lack of security for malicious purposes. The trust management frameworks that are currently available are not capable of fully creating trust relationships among objects, which leads to the need for further investigations in this direction. For this purpose, a standardised architectural reference model for the IoT and M2M communications is needed. This void prevents researchers from creating a proper security level in the IoT architecture, nurturing the development of ad hoc solutions.

The solution proposed in this thesis for detecting forged embedded machines in the IoT can be used as a method of pre-trust evaluation. However, it must be included as part of a high-level trust management framework. This trust management framework can use the proposed solution and at the same time detect other attacks. The combination of multi-layer metrics to detect attacks against trust mechanisms has not been properly studied yet. Similarly, studies about propagating trust to other M2M devices depending on the pre-trust evaluation must be carried out.

The binary classification approach used in this thesis can be modified to a multiclass classification to recognise the specific system type used by REMs and VESs. In this scenario, a further trust assessment can be created by exchanging the embedded machine type during the initial interaction. The trustor will request details of the trustee's system

type and will be able to detect if the trustee is providing the timing information that corresponds to that system. Therefore, the trustor will be able to determine if the trustee is using a specific embedded machine, such as Raspberry Pi, Arduino etc., or a specific virtual or an emulated system to create forged embedded machines, such as VirtualBox, QEMU etc.

This solution can also be modified for application to other fields. For example, it can be used to detect the Android's Dalvik virtual machine more quickly than existing heuristic detection methods [219, 220], which require around 20 minutes. By contrast, the proposed solution requires only circa 40 seconds, and is therefore more than 20 times faster.

The application of IoT mobile trust agents could be further investigated by applying the proposed solution to different IoT scenarios. Its application can also be evaluated with both open and closed networks in order to properly appreciate the dangerousness of the attack and the effectiveness of the solution. The ratio between ODP and ODS used in Equation 4.4 (Section 4.2.6) should be changed in order to identify the best classifier for each level of security required (Table 4.8). Further tests with high priority traffic should be performed during the simulations of the proposed solution in order to check if this affects the kernel behaviour and therefore detection results. Finally, other solutions based on timing behaviours should be evaluated for M2M-embedded devices that do not support the ping command locally, such as devices based only on IEEE 802.15.4, LoRa, LTE etc.

6.4 Summary

In this thesis, an overview of the Internet of Things is provided. The focus is on issues which prevent the creation of trusted relationships among IoT-embedded machines in M2M communications. It has been shown that the lack of a standard architectural reference model for incorporating trust prevents researchers from providing an optimal Trust Management Framework for the IoT. A new threat consisting of forged embedded machines used by attackers to subvert M2M communication has been identified. Several new solutions were proposed as part of IoT mobile trust agents in order to allow M2M-embedded machines and the IoT core to identify this attack. This is achieved by performing a pre-trust evaluation in order to save energy and computational resources when creating trust relationships.

Evaluations and results of the final proposed solution show its efficiency in terms of overall detection speed, overall detection performance and resilience against attacks, independent of the machine architecture and its OS. The detection of unknown embedded machines demonstrates its easy applicability to future IoT-embedded machines and also in a final standardised architecture reference model for M2M communications in the IoT. Finally, as demonstrated, this trust evaluation can be used by IoT applications to

preserve their operations in real-life scenarios. These are very important aspects because IoT does not connect only machines, but also people's lives.

Appendix A

Appendices

A.1 Characterisation Algorithm

```
#!/bin/sh
#
# Copyright 2015 Valerio Selis
#
# This program is free software; you can redistribute it and/or
   modify
# it under the terms of the GNU General Public License as
   published by
# the Free Software Foundation; either version 2 of the License,
   or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public
   License
# along with this program; if not, see <http://www.gnu.org/
   licenses/>.
#
# Usage: characterisation.sh <sim_number> <ping_number> <
   ping_interval> <ping_pkt_size>
#

# Check if the OS is Android
check_android=$(ls /sdcard)
# Check if the OS is NetBSD
check_netbsd=$(uname -a | grep "evb")
# Check if WNDR4700 is the real embedded machine
```

```
check_WNDR4700=$(uname -a | grep "WNDR4700")

# Select the folder for saving the log file
if [ "$check_android" = "" ];then
    if [ "$check_WNDR4700" = "" ];then
        main_log="/tmp/simping.log"
        main_dir="/tmp/"
    else
        # WNDR4700
        main_log="/tmp/mnt/sda1/simping.log"
        main_dir="/tmp/mnt/sda1/"
    fi
else
    # Android OS
    main_log="/sdcard/simping.log"
    main_dir="/sdcard/"
fi

echo "Starting  $(date) - $(date +%s)..."
echo "Starting  $(date) - $(date +%s)..." >> $main_log

# Retrieve number of CPUs and cores
if [ "$check_netbsd" = "" ];then
    cpus=$(cat /proc/cpuinfo | grep processor | awk '{print $NF}')
else
    cpus=$(sysctl -a | egrep -i 'hw.machine|hw.model|hw.ncpu' |
   grep "hw.ncpu:" | awk {'print $NF'})
fi
max_cpus=$(echo $cpus | awk {'print $NF'})
if [ "$max_cpus" = "" ];then
    cpus=0
    max_cpus=0
fi

# Number of simulation loops, minimum 1 loop
number_start=0
number_end=1
if [ "$1" != "" ];then
    # Number of loops given in input
    number_end=$1
fi

# Number of pings, default 1000
number_pings=1000
if [ "$2" != "" ];then
    # Number of pings given in input
    number_pings=$2
```

```
fi

# Interval number between pings, default 0.2
interval_pings=0.2
if [ "$3" != "" ];then
    # Interval number between pings given in input
    interval_pings=$3
fi

# Ping packet size, default 56 bytes
pkt_size=56
if [ "$4" != "" ];then
    # Ping packet size given in input
    pkt_size=$4
fi

end_sim='n'
while [ $end_sim != 'y' ];do
    # Execute 2 simulations
    # - Sim 0: ping
    # - Sim 1: ping and CPU under stress
    echo "Simulation loop #$number_start $(date) - $(date +%s)" >>
    $main_log
    sim_count=0
    ready='n'
    while [ $ready != 'y' ];do
        ts=$(date +%s)
        sim=""
        sim_dir=""
        # Start the dd command if necessary for stressing the CPU
        if [ $sim_count -eq 1 ];then
            for each in $cpus;do
                dd if=/dev/urandom of=/dev/null &
            done
            sim="02_$ts"
            sim_dir="ping_dd_$sim"
            echo "Sim #$sim_count: ping -c $number_pings -i
    $interval_pings -s $pkt_size 127.0.0.1 with dd ($ts)" >>
    $main_log
        else
            sim="1000_$ts"
            sim_dir="ping_$sim"
            echo "Sim #$sim_count: ping -c $number_pings -i
    $interval_pings -s $pkt_size 127.0.0.1 ($ts)" >> $main_log
        fi

        # Start the characterisation algorithm depending on the OS
```

```
    if [ "$check_netbsd" = "" ];then
        # Start the characterisation algorithm
        echo "Start characterisation: $(date) - $(date +%s)"
        echo "Start characterisation: $(date) - $(date +%s)"
>> "/$main_dir/$sim_dir/log"
        ping -c $number_pings -i $interval_pings -s $pkt_size
127.0.0.1 | while read LINE
        do
            # Store ping response time
            echo "$LINE"
            # Store timestamp value
            echo "$(date +%s)" >> "/$main_dir/$sim_dir/lo_ts"
            # Store CPU usage
            cat /proc/stat | grep '^cpu ' >> "/$main_dir/
$sim_dir/lo_cpu"
        done > "/$main_dir/$sim_dir/lo_ping"
        # Stop the characterisation algorithm
        echo "Characterisation finished! $(date) - $(date +%s)
"
        echo "Characterisation finished! $(date) - $(date +%s)
" >> "/$main_dir/$sim_dir/log"
        sleep 1s
    else
        # Start iostat for retrieving the CPU usage
        iostat -C 1 > "/$main_dir/$dir/iostat.txt" &
        iostat_pid=$!
        # Make sure iostat.txt contains an up to date CPU
usage
        sleep 1s
        # Start the characterisation for NetBSD
        echo "Start characterisation: $(date) - $(date +%s)"
        echo "Start characterisation: $(date) - $(date +%s)"
>> "/$main_dir/$sim_dir/log"
        ping -c $number_pings -i $interval_pings -s $pkt_size
127.0.0.1 | while read LINE
        do
            # Store ping response time
            echo "$LINE"
            # Store timestamp value
            echo "$(date +%s)" >> "/$main_dir/$sim_dir/lo_ts"
            # Store CPU usage
            tail -1 "/$main_dir/$dir/iostat.txt" | tr "\n" " "
 >> "/$main_dir/$sim_dir/lo_cpu"
        done > "/$main_dir/$sim_dir/lo_ping"
        # Stop the characterisation algorithm for NetBSD
        echo "Characterisation finished! $(date) - $(date +%s)
"
```

```
            echo "Characterisation finished! $(date) - $(date +%s)
    " >> "/$main_dir/$sim_dir/log"
            # Kill iostat
            kill $iostat_pid
            sleep 1s
        fi

        # Kill all dd commands if it is Sim 1
        if [ $sim_count -eq 1 ];then
            killall dd
        fi

        # Increase the simulation number
        sim_count=$(($sim_count+1))
        # Check if two simulations were executed
        if [ $sim_count -eq 2 ];then
            echo "Finished two simulations for loop: $number_start
     of $number_end!"
            echo "Finished two simulations for loop: $number_start
     of $number_end!" >> $main_log
            # Restart simulation counter
            sim_count=0
            ready='y'
        fi
    done

    # Increase the simulation loop counter
    number_start=$(($number_start+1))
    # Check if all simulation loops were executed
    if [ $number_start -eq $number_end ];then
        echo "Finished! Goodbye :)"
        echo "Finished! Goodbye :)" >> $main_log
        end_sim='y'
    fi
done

exit
```

A.2 Characterisation Algorithm Outputs

File with ping response time (lo_ping)

```
Line# |             Data
-----------------------------------------------------------------------

1     | PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
```

```
2     | 64 bytes from 127.0.0.1: icmp_req=1 ttl=64 time=0.260 ms
3     | 64 bytes from 127.0.0.1: icmp_req=2 ttl=64 time=0.216 ms
4     | 64 bytes from 127.0.0.1: icmp_req=3 ttl=64 time=0.210 ms
5     | 64 bytes from 127.0.0.1: icmp_req=4 ttl=64 time=0.209 ms
6     | 64 bytes from 127.0.0.1: icmp_req=5 ttl=64 time=0.211 ms
7     | 64 bytes from 127.0.0.1: icmp_req=6 ttl=64 time=0.213 ms
8     | 64 bytes from 127.0.0.1: icmp_req=7 ttl=64 time=0.201 ms
9     | 64 bytes from 127.0.0.1: icmp_req=8 ttl=64 time=0.210 ms
10    | 64 bytes from 127.0.0.1: icmp_req=9 ttl=64 time=0.207 ms

....                 ....

996   | 64 bytes from 127.0.0.1: icmp_req=995 ttl=64 time=0.194 ms
997   | 64 bytes from 127.0.0.1: icmp_req=996 ttl=64 time=0.216 ms
998   | 64 bytes from 127.0.0.1: icmp_req=997 ttl=64 time=0.212 ms
999   | 64 bytes from 127.0.0.1: icmp_req=998 ttl=64 time=0.200 ms
1000  | 64 bytes from 127.0.0.1: icmp_req=999 ttl=64 time=0.213 ms
1001  | 64 bytes from 127.0.0.1: icmp_req=1000 ttl=64 time=0.216
   ms
1002  |
1003  | --- 127.0.0.1 ping statistics ---
1004  | 1000 packets transmitted, 1000 received, 0% packet loss,
   time 199796ms
1005  | rtt min/avg/max/mdev = 0.184/0.213/0.339/0.010 ms
```

File with timestamp values (lo_ts)

```
Line# |    Data
------------------
1     | 1433860111
2     | 1433860111
3     | 1433860111
4     | 1433860111
5     | 1433860111
6     | 1433860112
7     | 1433860112
8     | 1433860112
9     | 1433860112
10    | 1433860112
11    | 1433860113

....        ....

996   | 1433860310
997   | 1433860310
998   | 1433860310
999   | 1433860310
1000  | 1433860311
```

```
1001   | 1433860311
1002   | 1433860311
1003   | 1433860311
1004   | 1433860311
1005   | 1433860311
```

A.3 Machine Emulation Detection Algorithm

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
#
# Copyright 2015 Valerio Selis
#
# This program is free software; you can redistribute it and/or
    modify
# it under the terms of the GNU General Public License as
    published by
# the Free Software Foundation; either version 2 of the License,
    or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public
    License
# along with this program; if not, see <http://www.gnu.org/
    licenses/>.
#
# Usage: python MEDA.py <ping response time> <timestamp values>
#

# Import Python modules
import timeit, sys, string
import numpy as np

# Check inputs
if len(sys.argv) != 3:
     print "Usage: python " + str(sys.argv[0]) + " <lo_ping> <lo_ts
    >"
     exit()

# Initialise characterisation metrics for ping response time (P)
    and timestamp values (T)
```

```
## Min
PMinL = 0.067
PMinU = 0.193
TminL = 0.0
TminU = 0.0
## Max
PMaxL = 0.140
PMaxU = 2.060
TMaxL = 1.0
TMaxU = 2.0
## Range
PRangeL = 0.061
PRangeU = 1.993
## Sum
PSumL = 99.064
PSumU = 288.117
TSumL = 199.0
TSumU = 201.0
## Mean
PMeanL = 0.099
PMeanU = 0.288
TMeanL = 0.199
TMeanU = 0.201
## Variance
PVarL = 0.0
PVarU = 0.034
TVarL = 0.159
TVarU = 0.162
## Standard Deviation (SD)
PStdL = 0.002
PStdU = 0.183
TStdL = 0.399
TStdU = 0.402
## Mean-Standard Deviation (Mean-SD)
PMeanStdL = 0.060
PMeanStdU = 0.215
## Mean+Standard Deviation (Mean+SD)
PMeanStdpL = 0.110
PMeanStdpU = 0.452

# Store the time
start_time = timeit.default_timer()

# Load ping response time and timestamp values from the target
   machine in the dictionary
## Ping response time
TargetP = []
```

```
with open(str(sys.argv[1]), 'rb') as textfile:
    pr = textfile.readlines()
    for row in pr:
        ping = string.split(str(row), '=')
        if len(ping) == 4:
            TargetP.append(float(ping[3].replace('\r\n', '').
   replace('ping', '').replace('ms', '')))
## Timestamp values
TargetT = []
with open(str(sys.argv[2]), 'rb') as textfile:
    pr = textfile.readlines()
    ts = []
    i = 0
    for row in pr:
        ts.append(row.replace('\n', ''))
        if i == 1:
            TargetT.append(0)
        elif i <= 1001:
            TargetT.append(int(ts[i])-int(ts[i-1]))
        i += 1

# Extract target characterisation metrics from ping response time
   and timestamp values
## Min
TargetPMin = min(TargetP)
TargetTMin = min(TargetT)
## Max
TargetPMax = max(TargetP)
TargetTMax = max(TargetT)
## Range
TargetPRange = TargetPMax - TargetPMin
## Sum
TargetPSum = sum(TargetP)
TargetTSum = sum(TargetT)
## Mean
TargetPMean = np.mean(TargetP)
TargetTMean = np.mean(TargetT)
## Variance
TargetPVar = np.var(TargetP)
TargetTVar = np.var(TargetT)
## Standard Deviation (SD)
TargetPStd = np.std(TargetP)
TargetTStd = np.std(TargetT)
## Mean-Standard Deviation (Mean-SD)
TargetPMeanStd = TargetPMean - TargetPStd
## Mean+Standard Deviation (Mean+SD)
TargetPMeanStdp = TargetPMean + TargetPStd
```

```
# Detection thresholds
real = 0
virtual = 0
if PMinL <= TargetPMin <= PMinU:
    real = real + 1
else:
    virtual = virtual + 1
if PMaxL <= TargetPMax <= PMaxU:
    real = real + 1
else:
    virtual = virtual + 1
if PRangeL <= TargetPRange <= PRangeU:
    real = real + 1
else:
    virtual = virtual + 1
if PSumL <= TargetPSum <= PSumU:
    real = real + 1
else:
    virtual = virtual + 1
if PMeanL <= TargetPMean <= PMeanU:
    real = real + 1
else:
    virtual = virtual + 1
if PVarL <= TargetPVar <= PVarU:
    real = real + 1
else:
    virtual = virtual + 1
if PStdL <= TargetPStd <= PStdU:
    real = real + 1
else:
    virtual = virtual + 1
if PMeanStdL <= TargetPMeanStd <= PMeanStdU:
    real = real + 1
else:
    virtual = virtual + 1
if PMeanStdpL <= TargetPMeanStdp <= PMeanStdpU:
    real = real + 1
else:
    virtual = virtual + 1
if TminL <= TargetTMin <= TminU:
    real = real + 1
else:
    virtual = virtual + 1
if TMaxL <= TargetTMax <= TMaxU:
    real = real + 1
else:
```

```
    virtual = virtual + 1
if TSumL <= TargetTSum <= TSumU:
    real = real + 1
else:
    virtual = virtual + 1
if TMeanL <= TargetTMean <= TMeanU:
    real = real + 1
else:
    virtual = virtual + 1
if TVarL <= TargetTVar <= TVarU:
    real = real + 1
else:
    virtual = virtual + 1
if TStdL <= TargetTStd <= TStdU:
    real = real + 1
else:
    virtual = virtual + 1

if virtual == 0:
    print "The target embedded machine is a REM"
else:
    print "The target embedded machine is a VES"

# Calculate the time required to detect if the target embedded
   machine is a REMs or VESs
print "Time required by MEDA: " + str(timeit.default_timer() -
   start_time) + " (s)\n"

exit()
```

A.4 Classification-based Algorithm

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
#
# Copyright 2015 Valerio Selis
#
# This program is free software; you can redistribute it and/or
   modify
# it under the terms of the GNU General Public License as
   published by
# the Free Software Foundation; either version 2 of the License,
   or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
```

```
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public
   License
# along with this program; if not, see <http://www.gnu.org/
   licenses/>.
#
# Usage: python classification.py <dictionary.pkl> <FSM[ERT|L1FS]>
    <CM[kNN|RF]> <num_pings[25|200]>
#

# Import Python modules
import timeit, cPickle, sys
import sklearn
import sklearn.cross_validation as cv
import sklearn.neighbors as nb
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier

# Classification of REMs and VESs by using k-NN or RF, where:
# - X: target dataset (REMs and VESs) with features extracted by
   the features selection method (FSM) for specified number of
   pings
# - y: target values (0: REM, 1: VES)
# - FSM: features selection method (ERT or L1-FS)
# - CM: classification method (k-NN or RF)
# - num_pings: number of pings (25 or 200)
def Classification_algorithm(X, y, FSM, CM, num_pings):
    if FSM == "ERT":
        # ERT
        if num_pings == 25:
            # 25 pings
            # k-NN
            n_neighbors = 4
            weights = 'distance'
            # RF
            max_features = 'sqrt'
            n_estimators = 500
        else:
            # 200 pings
            # k-NN
            n_neighbors = 2
            weights = 'uniform'
            # RF
            max_features = 'sqrt'
```

```
            n_estimators = 1000
    else:
        # L1-FS
        if num_pings == 25:
            # 25 pings
            # k-NN
            n_neighbors = 1
            weights = 'uniform'
            # RF
            max_features = 'sqrt'
            n_estimators = 1000
        else:
            # 200 pings
            # k-NN
            n_neighbors = 1
            weights = 'uniform'
            # RF
            max_features = 'sqrt'
            n_estimators = 500

    start_time = timeit.default_timer()
    # Split the dataset in two part: trainset (75%) and testset
(25%)
    X_train, X_test, y_train, y_test = cv.train_test_split(X, y,
test_size=0.25, random_state=0)
    split_time = timeit.default_timer() - start_time

    if CM == "kNN":
        start_time = timeit.default_timer()
        knc = nb.KNeighborsClassifier(n_neighbors=n_neighbors,
weights=weights)
        knc.fit(X_train, y_train)
        y_pred = knc.predict(X_test)
        classification_time = timeit.default_timer() - start_time
        A = metrics.accuracy_score(y_test, y_pred)
        P, R, F1, support = metrics.
precision_recall_fscore_support(y_test, y_pred)
        ODP = float(A + P[0] + R[0] + F1[0] + P[1] + R[1] + F1[1])
/7.0)

        # Caclulate the time required to classify all machines (
REMs and VESs) in the dataset
        print "Time required by k-NN: " + str(classification_time)
    + " (s)\n"

        # Display Accuracy, Precision, Recoll and F1-score for
both REMs and VESs
```

```
        print "A: " + str(A)
        print "P_REM: " + str(P[0])
        print "R_REM: " + str(R[0])
        print "F1_REM: " + str(F1[0])
        print "P_VES: " + str(P[1])
        print "R_VES: " + str(R[1])
        print "F1_VES: " + str(F1[1])
        print "Support REM: " + str(support[0])
        print "Support VES: " + str(support[1])

        # Display ODP
        print "ODP: " + str(ODP)

    elif CM == "RF":
        start_time = timeit.default_timer()
        rfc = RandomForestClassifier(n_estimators=n_estimators,
    max_features=max_features, random_state=0)
        rfc.fit(X_train, y_train)
        y_pred = rfc.predict(X_test)
        classification_time = timeit.default_timer() - start_time
        A = metrics.accuracy_score(y_test, y_pred)
        P, R, F1, support = metrics.
    precision_recall_fscore_support(y_test, y_pred)
        ODP = float(A + P[0] + R[0] + F1[0] + P[1] + R[1] + F1[1])
    /7.0)

        # Caclulate the time required to classify all machines (
    REMs and VESs) in the dataset
        print "Time required by RF: " + str(classification_time) +
     " (s)\n"

        # Display Accuracy, Precision, Recoll and F1-score for
    both REMs and VESs
        print "A: " + str(A)
        print "P_REM: " + str(P[0])
        print "R_REM: " + str(R[0])
        print "F1_REM: " + str(F1[0])
        print "P_VES: " + str(P[1])
        print "R_VES: " + str(R[1])
        print "F1_VES: " + str(F1[1])
        print "Support REM: " + str(support[0])
        print "Support VES: " + str(support[1])

        # Display ODP
        print "ODP: " + str(ODP)

# Check inputs
```

```
if len(sys.argv) != 5:
    print "Usage: python " + str(sys.argv[0]) + " <dictionary.pkl>
    <FSM[ERT|L1FS]> <CM[kNN|RF]> <num_pings[25|200]>"
    exit()

# Dictionary initialisation with list of all machines (REMs and
   VESs) and their characterisations
with open(str(sys.argv[1]), 'rb') as pklfile:
    dictionary = cPickle.load(pklfile)

# Retrieve from the dictionary the normilised X, which contains
   features selected by FSM for specific number of pings
X = dictionary["X_" + str(sys.argv[2]) + str(sys.argv[4])]
# Retrieve from the dictionary y
y = dictionary["y_" + str(sys.argv[2]) + str(sys.argv[4])]
# Feature selection method used
FSM = str(sys.argv[2])
# Classification method to be used for classifying REMs and VESs
CM = str(sys.argv[3])
# Number of pings used by the characterisation algorithm
num_pings = str(sys.argv[4])
# Launch the classification algorithm
Classification_algorithm(X, y, FSM, CM, num_pings)

exit()
```

A.5 Architecture-based Timing Test on Raspberry Pi 2 model B

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

#define BCM2708_PERI_BASE        0x3F000000

// Read the cycle count register (CCNT)
static inline unsigned int get_cyclecount (void) {
  unsigned int value;
  asm volatile ("MRC p15, 0, %0, C9, C13, 0\t\n": "=r"(value));
  return value;
}

// Enable all counters (including cycle counter)
static inline void init_perfcounters (int32_t do_reset, int32_t
   enable_divider) {
```

```
  int32_t value = 1;

  // Peform reset:
  if (do_reset) {
      value |= 2;     // Reset all counters to zero
      value |= 4;     // Reset cycle counter to zero
  }

  if (enable_divider)
      value |= 8;     // Enable "by 64" divider for CCNT

  value |= 16;

  // Program the performance-counter control-register
  asm volatile ("MCR p15, 0, %0, c9, c12, 0\t\n" :: "r"(value));

  // Enable all counters
  asm volatile ("MCR p15, 0, %0, c9, c12, 1\t\n" :: "r"(0x8000000f
   ));

  // Clear overflows
  asm volatile ("MCR p15, 0, %0, c9, c12, 3\t\n" :: "r"(0x8000000f
   ));
}

// Perform the Architecture-based Timing Test
static int __init eacc_init(void) {
  long int time_nop = 0;
  long int time_access_sctlr;
  long int difference_nop_sctlr;

  // Disable counter overflow interrupts (just in case)
  asm ("MCR p15, 0, %0, C9, C14, 2\n\t" :: "r"(0x8000000f));

  printk(KERN_INFO "Architecture-based Timing Test kernel module
   loaded\n");

  init_perfcounters (1, 0);

  // Measure the time of a NOP operation
  time_nop = get_cyclecount();
  time_nop = get_cyclecount() - time_nop;

  // Measure the time to access system control register (SCTLR)
  time_access_sctlr = get_cyclecount();
  asm volatile ("MRC p15, 0, %0, c1, c0, 0\t\n" : "=r"(c1) ::);
  time_access_sctlr = get_cyclecount() - time_access_sctlr;
```

```
  printk(KERN_INFO "Time of a NOP operation: %ld\n", time_nop);
  printk(KERN_INFO "Time to access the SCTLR: %ld\n",
   time_access_sctlr);

  return 0;
}

static void __exit eacc_exit(void) {
  printk(KERN_INFO "Architecture-based Timing Test kernel module
   unloaded\n");
}

module_init(eacc_init);
module_exit(eacc_exit);
```

A.6 Architecture-based Timing Test: QEMU patch

```
// Patch for the 'qemu-2.12.0-rc0/target/arm/helper.c' file

...

uint64_t fake_counter = 0;

...

void pmccntr_sync(CPUARMState *env)
{
    uint64_t temp_ticks;

    temp_ticks = muldiv64(qemu_clock_get_ns(QEMU_CLOCK_VIRTUAL),
                          ARM_CPU_FREQ, NANOSECONDS_PER_SECOND);

    if (env->cp15.c9_pmcr & PMCRD) {
        /* Increment once every 64 processor clock cycles */
        temp_ticks /= 64;
    }

    if (arm_ccnt_enabled(env)) {
        env->cp15.c15_ccnt = temp_ticks - env->cp15.c15_ccnt;
    }

    fake_counter  = 0;
}
```

```
static uint64_t pmccntr_read(CPUARMState *env, const ARMCPRegInfo
    *ri)
{
    uint64_t total_ticks;

    if (!arm_ccnt_enabled(env)) {
        /* Counter is disabled, do not change value */
        return env->cp15.c15_ccnt;
    }

    total_ticks = muldiv64(qemu_clock_get_ns(QEMU_CLOCK_VIRTUAL),
                           ARM_CPU_FREQ, NANOSECONDS_PER_SECOND);

    if (env->cp15.c9_pmcr & PMCRD) {
        /* Increment once every 64 processor clock cycles */
        total_ticks /= 64;
    }

    if (fake_counter  == 0 || fake_counter  == 3)
        total_ticks = 2;
    else if  (fake_counter  == 1)
        total_ticks = 3;
    else if (fake_counter  == 2)
        total_ticks = 0;
    else if  (fake_counter  == 4)
        total_ticks = 4;

    fake_counter ++;
    return total_ticks - env->cp15.c15_ccnt;
}

...
```

Appendix B

Appendices

B.1 Performance Results from MEDA, *k*-NN and RF

TABLE B.1: MEDA performance results on recognising REMs and VESs for 1000 pings (dataset of REMs from Chapter 3, Section 3.4.2).

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	1.0000	0.4455	0.6164	0.7921	78.03%
VES	0.7504	1.0000	0.8574		
Average	0.8752	0.7228	0.7369		

TABLE B.2: MEDA performance results on recognising REMs and VESs for 25 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.5340	1.0000	0.6962	0.6728	71.78%
VES	1.0000	0.4764	0.6454		
Average	0.7670	0.7382	0.6708		

TABLE B.3: ERT+k-NN performance results on recognising REMs and VESs for 25 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9878	0.9810	0.9844	0.9846	98.46%
VES	0.9815	0.9881	0.9848		
Average	0.9847	0.9846	0.9846		

Table B.4: L1-FS+k-NN performance results on recognising REMs and VESs for 25 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9842	0.9814	0.9828	0.9830	98.30%
VES	0.9818	0.9845	0.9832		
Average	0.9830	0.9830	0.9830		

Table B.5: ERT+RF performance results on recognising REMs and VESs for 25 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9968	0.9952	0.9960	0.9960	99.60%
VES	0.9953	0.9968	0.9960		
Average	0.9960	0.9960	0.9960		

Table B.6: L1-FS+RF performance results on recognising REMs and VESs for 25 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9968	0.9952	0.9960	0.9960	99.60%
VES	0.9953	0.9968	0.9960		
Average	0.9960	0.9960	0.9960		

Table B.7: MEDA performance results on recognising REMs and VESs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.6083	1.0000	0.7564	0.7585	78.53%
VES	1.0000	0.6136	0.7605		
Average	0.8041	0.8068	0.7585		

Table B.8: ERT+k-NN performance results on recognising REMs and VESs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9900	0.9964	0.9932	0.9932	99.32%
VES	0.9964	0.9901	0.9932		
Average	0.9932	0.9932	0.9932		

Table B.9: L1-FS+k-NN performance results on recognising REMs and VESs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9955	0.9899	0.9927	0.9928	99.28%
VES	0.9901	0.9956	0.9929		
Average	0.9928	0.9928	0.9928		

Table B.10: ERT+RF performance results on recognising REMs and VESs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9996	0.9976	0.9986	0.9986	99.86%
VES	0.9976	0.9996	0.9986		
Average	0.9986	0.9986	0.9986		

Table B.11: L1-FS+RF performance results on recognising REMs and VESs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9992	0.9976	0.9984	0.9984	99.84%
VES	0.9976	0.9992	0.9984		
Average	0.9984	0.9984	0.9984		

Table B.12: MEDA performance results on recognising unknown REMs and VESs for 25 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.0000	0.5000	0.0000	0.5000	57.14%
VES	1.0000	1.0000	1.0000		
Average	0.5000	0.7500	0.5000		

Table B.13: ERT+k-NN performance results on recognising unknown REMs and VESs for 25 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	1.0000	0.8075	0.8821	0.9005	94.00%
VES	1.0000	0.9935	0.9967		
Average	1.0000	0.9005	0.9394		

Table B.14: L1-FS+k-NN performance results on recognising unknown REMs and VESs for 25 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	1.0000	0.8960	0.9420	0.9360	96.26%
VES	1.0000	0.9760	0.9879		
Average	1.0000	0.9360	0.9649		

Table B.15: ERT+RF performance results on recognising unknown REMs and VESs for 25 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	1.0000	0.9585	0.9786	0.9750	98.56%
VES	1.0000	0.9915	0.9957		
Average	1.0000	0.9750	0.9872		

Table B.16: L1-FS+RF performance results on recognising unknown REMs and VESs for 25 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	1.0000	0.4008	0.5723	0.9850	99.14%
VES	1.0000	0.9935	0.9967		
Average	1.0000	0.6972	0.7845		

Table B.17: MEDA performance results on recognising unknown REMs and VESs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.7750	0.5005	0.2760	0.5003	72.17%
VES	1.0000	1.0000	1.0000		
Average	0.8875	0.7503	0.6380		

Table B.18: ERT+k-NN performance results on recognising unknown REMs and VESs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	1.0000	0.9775	0.9885	0.9783	98.75%
VES	1.0000	0.9790	0.9893		
Average	1.0000	0.9783	0.9889		

Table B.19: L1-FS+k-NN performance results on recognising unknown REMs and VESs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	1.0000	0.9635	0.9811	0.9793	98.80%
VES	1.0000	0.9950	0.9975		
Average	1.0000	0.9793	0.9893		

Table B.20: ERT+RF performance results on recognising unknown REMs and VESs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	1.0000	0.9995	0.9997	0.9990	99.94%
VES	1.0000	0.9985	0.9992		
Average	1.0000	0.9990	0.9995		

Table B.21: L1-FS+RF performance results on recognising unknown REMs and VESs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	1.0000	0.9410	0.9687	0.9700	98.26%
VES	1.0000	0.9990	0.9995		
Average	1.0000	0.9700	0.9841		

Table B.22: k-NN performance results on recognising REMs and FTAs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.0000	0.0600	0.0011	0.9410	95.44%
FTA	1.0000	0.9400	0.9700		
Average	1.0000	0.0600	0.1100		

Table B.23: RF performance results on recognising REMs and FTAs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.0000	0.6500	0.2100	0.3480	53.97%
FTA	1.0000	0.3500	0.5200		
Average	1.0000	0.6500	0.7900		

B.2 Performance Results from *k*-NN and RF without the Normalisation Step

Table B.24: ERT+k-NN performance results on recognising REMs and VESs for 25 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9818	0.9810	0.9814	0.9816	98.16%
VES	0.9814	0.9822	0.9818		
Average	0.9816	0.9816	0.9816		

Table B.25: L1-FS+k-NN performance results on recognising REMs and VESs for 25 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9471	0.9608	0.9539	0.9540	95.40%
VES	0.9610	0.9473	0.9541		
Average	0.9540	0.9541	0.9540		

Table B.26: ERT+RF performance results on recognising REMs and VESs for 25 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9956	0.9943	0.9950	0.9846	99.35%
VES	0.9945	0.9956	0.9950		
Average	0.9950	0.9950	0.9950		

Table B.27: L1-FS+RF performance results on recognising REMs and VESs for 25 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9919	0.9943	0.9931	0.9846	99.20%
VES	0.9944	0.9921	0.9933		
Average	0.9932	0.9932	0.9932		

TABLE B.28: ERT+k-NN performance results on recognising REMs and VESs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9830	0.9802	0.9816	0.9818	98.18%
VES	0.9806	0.9834	0.9820		
Average	0.9818	0.9818	0.9818		

TABLE B.29: L1-FS+k-NN performance results on recognising REMs and VESs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9783	0.9806	0.9794	0.9796	97.96%
VES	0.9809	0.9786	0.9798		
Average	0.9796	0.9796	0.9796		

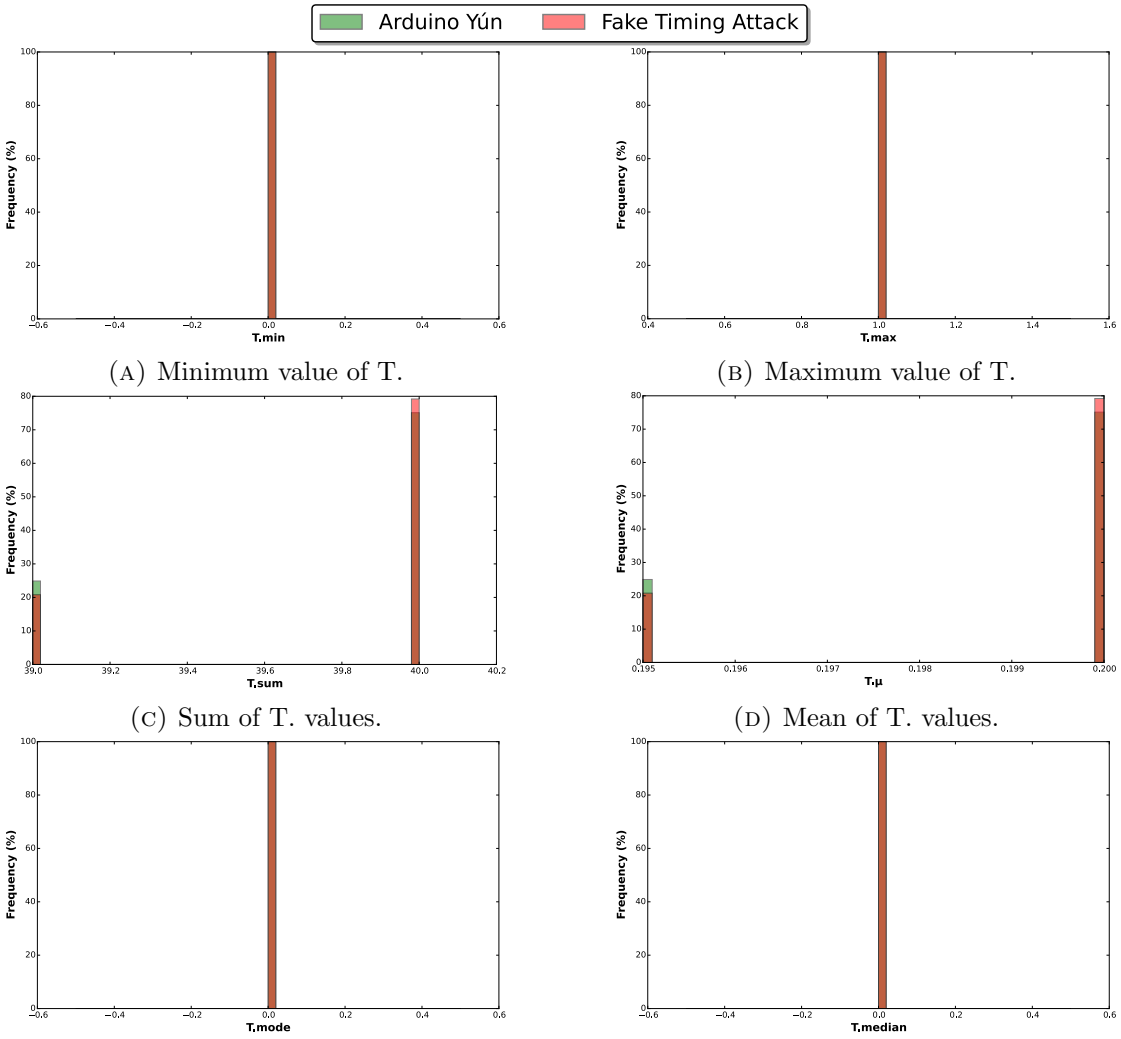
TABLE B.30: ERT+RF performance results on recognising REMs and VESs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9996	0.9984	0.9990	0.9846	99.69%
VES	0.9984	0.9996	0.9990		
Average	0.9990	0.9990	0.9990		

TABLE B.31: L1-FS+RF performance results on recognising REMs and VESs for 200 pings.

	Precision	**Recall**	**F1-score**	**Accuracy**	**ODP**
REM	0.9984	0.9976	0.9980	0.9846	99.61%
VES	0.9976	0.9984	0.9980		
Average	0.9980	0.9980	0.9980		

B.3 Cumulative Frequency Histograms for Timestamp Features used to Detect FTA



(A) Minimum value of T.

(B) Maximum value of T.

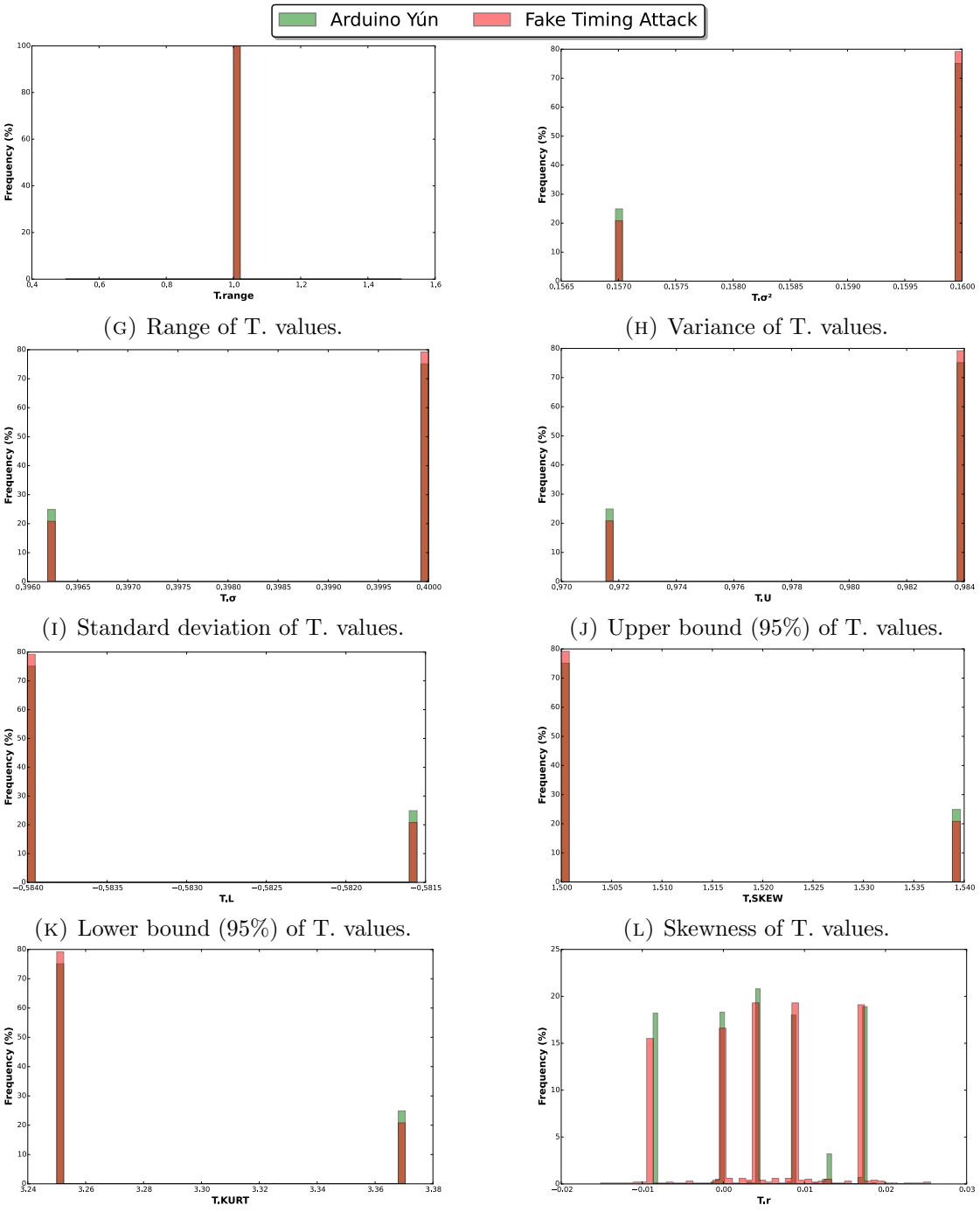
(C) Sum of T. values.

(D) Mean of T. values.

(E) Mode of T. values.

(F) Median of T. values.

FIGURE B.1: Cumulative frequency histograms for T. features obtained from the Arduino Yún and FTA for 200 pings and 1000 characterisation tests.



(G) Range of T. values.

(H) Variance of T. values.

(I) Standard deviation of T. values.

(J) Upper bound (95%) of T. values.

(K) Lower bound (95%) of T. values.

(L) Skewness of T. values.

(M) Kurtosis of T. values.

(N) Pearson correlation coefficient of T. values.

FIGURE B.1: (continued) Cumulative frequency histograms for T. features obtained from the Arduino Yún and FTA for 200 pings and 1000 characterisation tests.

Bibliography

[1] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. URL http://dblp.uni-trier.de/db/journals/jmlr/jmlr12.html{#}PedregosaVGMTGBPWDVPCBPD11.

[2] The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, October 2010. ISSN 13891286. doi: 10.1016/j.comnet.2010.05.010. URL http://www.sciencedirect.com/science/article/pii/S1389128610001568.

[3] Min Chen, Jiafu Wan, and Fang Li. Machine-to-Machine Communications. *KSII Transactions on Internet and Information Systems (TIIS)*, 6(2):480–497, 2012. ISSN 1976-7277. URL http://www.dbpia.co.kr/Article/1628958.

[4] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, September 2013. ISSN 0167739X. doi: 10.1016/j.future.2013.01.010. URL http://www.sciencedirect.com/science/article/pii/S0167739X13000241.

[5] Mahbubul Alam, Rasmus H Nielsen, and Neeli R Prasad. The evolution of M2M into IoT. In *2013 1st International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pages 112–115. IEEE, July 2013.

[6] Valerio Selis and Alan Marshall. MEDA: A Machine Emulation Detection Algorithm. In *Proceedings of the 12th International Conference on Security and Cryptography*, pages 228–235. SCITEPRESS - Science and and Technology Publications, 2015. ISBN 978-989-758-117-5. doi: 10.5220/0005535202280235. URL http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0005535202280235.

[7] Anas M Mzahm, Mohd Sharifuddin Ahmad, and Alicia YC Tang. Agents of Things (AoT): An intelligent operational concept of the Internet of Things (IoT). In

2013 13th International Conference on Intellient Systems Design and Applications, pages 159–164. IEEE, December 2013.

[8] Andrea Giordano, Giandomenico Spezzano, and Andrea Vinci. Smart Agents and Fog Computing for Smart City Applications. In *International Conference on Smart Cities*, pages 137–146. Springer, 2016.

[9] Giancarlo Fortino. Agents Meet the IoT: Toward Ecosystems of Networked Smart Objects. *IEEE Systems, Man, and Cybernetics Magazine*, 2(2):43–47, April 2016.

[10] George Lawton. Machine-to-machine technology gears up for growth. *Computer*, 37(9):12–15, September 2004.

[11] Gyu Myoung Lee, Noel Crespi, Jun Kyun Choi, and Matthieu Boussard. Internet of Things. *Evolution of Telecommunication Services*, 7768:257–282, 2013. doi: 10.1007/978-3-642-41569-2_13. URL http://link.springer.com/chapter/10.1007/978-3-642-41569-2_13.

[12] Juan Antonio Guerrero-ibanez, Sherali Zeadally, and Juan Contreras-Castillo. Integration challenges of intelligent transportation systems with connected vehicle, cloud computing, and internet of things technologies. *IEEE Wireless Communications*, 22(6):122–128, December 2015.

[13] Intel Corporation. Building an Intelligent Transportation System with the Internet of Things (IoT), 2014. URL http://www.intel.co.uk/content/dam/www/program/embedded/internet-of-things/blueprints/iot-building-intelligent-transport-system-blueprint.pdf. Accessed: 14-10-2016.

[14] Rifat Shahriyar, Md Faizul Bari, Gourab Kundu, Sheikh Iqbal Ahamed, and Md Mostofa Akbar. Intelligent mobile health monitoring system (IMHMS). In *International Conference on Electronic Healthcare*, pages 5–12. Springer, September 2009.

[15] M Brian Blake. An Internet of Things for Healthcare. *IEEE Internet Computing*, 19(4):4–6, July 2015.

[16] SM Riazul Islam, Daehan Kwak, MD Humaun Kabir, Mahmud Hossain, and Kyung-Sup Kwak. The Internet of Things for Health Care: A Comprehensive Survey. *IEEE Access*, 3:678–708, 2015.

[17] Chia-Fen Chi and Ratna Sari Dewi. Visual and auditory icons for intelligent building. In *2014 International Conference on Intelligent Green Building and Smart Grid (IGBSG)*, pages 1–5. IEEE, April 2014.

[18] Ahmad Shahi, Md Nasir Sulaiman, Norwati Mustapha, and Thinagaran Perumal. Naive Bayesian decision model for the interoperability of heterogeneous systems in an intelligent building environment. *Automation in Construction*, 54:83–92, 2015.

[19] Rawlson O'Neil King. Cyber security for intelligent buildings. *Engineering & Technology Reference*, 2016.

[20] BBC News Technology. Malware is making ATMs "spit cash", 22 November 2016. URL http://www.bbc.co.uk/news/technology-38063142. Accessed: 14-10-2016.

[21] BBC News Technology. $5 "Poison Tap" hacks locked computers, 17 November 2016. URL http://www.bbc.co.uk/news/technology-38012699. Accessed: 14-10-2016.

[22] BBC News Technology. Def Con: Do smart devices mean dumb security?, 6 August 2016. URL http://www.bbc.co.uk/news/technology-36995288. Accessed: 14-10-2016.

[23] BBC News Asia. South China Sea: Vietnam airport screens hacked, 29 July 2016. URL http://www.bbc.co.uk/news/world-asia-36927674. Accessed: 14-10-2016.

[24] BBC News Technology. Osram Lightify light bulbs "vulnerable to hack", 27 July 2016. URL http://www.bbc.co.uk/news/technology-36903274. Accessed: 14-10-2016.

[25] BBC News Technology. Web baby-monitoring cameras open to hacking, study warns, 3 September 2015. URL http://www.bbc.co.uk/news/technology-34138480. Accessed: 14-10-2016.

[26] BBC News Technology. Breached webcam and baby monitor site flagged by watchdogs, 21 November 2014. URL http://www.bbc.co.uk/news/technology-30121159. Accessed: 14-10-2016.

[27] BBC News Technology. Smart TVs subverted by radio attack, 9 June 2014. URL http://www.bbc.co.uk/news/technology-27761756. Accessed: 14-10-2016.

[28] BBC News Technology. "Contagious" wi-fi virus created by Liverpool researchers, 26 February 2014. URL http://www.bbc.co.uk/news/technology-26352439. Accessed: 14-10-2016.

[29] BBC News Technology. Fridge sends spam emails as attack hits smart gadgets, 17 February 2014. URL http://www.bbc.co.uk/news/technology-25780908. Accessed: 14-10-2016.

[30] Dave Lee. Global internet slows after "biggest attack in history", 27 March 2013. URL http://www.bbc.co.uk/news/technology-21954636. Accessed: 14-10-2016.

[31] BBC News Technology. Iranian oil terminal "offline" after "malware attack", 23 April 2012. URL http://www.bbc.co.uk/news/technology-17811565. Accessed: 14-10-2016.

[32] Jonathan Fildes. Stuxnet worm "targeted high-value Iranian assets", 23 September 2010. URL http://www.bbc.co.uk/news/technology-11388018. Accessed: 14-10-2016.

[33] BBC News Technology. First human "infected with computer virus", 27 May 2010. URL http://www.bbc.co.uk/news/10158517. Accessed: 14-10-2016.

[34] Jose Pagliery. Trump Hotels attacked by hackers – again, 5 April 2016. URL http://money.cnn.com/2016/04/05/technology/trump-hotels-hacked/. Accessed: 14-10-2016.

[35] Ahiza Garcia. Pokemon Go crashes and hackers claim responsibility, 18 July 2016. URL http://money.cnn.com/2016/07/16/technology/pokemon-go-crash-game/. Accessed: 13-02-2017.

[36] Jose Pagliery. Trump hotels hacked, credit card data at risk, 30 September 2015. URL http://money.cnn.com/2015/09/30/technology/trump-hotels-hack/. Accessed: 14-10-2016.

[37] Sam Thielman and Elle Hunt. Cyber attack: hackers "weaponised" everyday devices with malware, 22 October 2016. URL https://www.theguardian.com/technology/2016/oct/22/cyber-attack-hackers-weaponised-everyday-devices-with-malware-to-mount-assault. Accessed: 14-10-2016.

[38] Charles Arthur. What the New York Times Chinese hack tells us about the layer cake of hacking, 1 February 2013. URL https://www.theguardian.com/technology/2013/jan/31/new-york-times-hacking-china-lessons. Accessed: 14-10-2016.

[39] Carly Page. BlackBerry hacks a kettle to demonstrate IoT security strain, 20 July 2016. URL http://www.theinquirer.net/inquirer/news/2465342/blackberry-hacks-a-kettle-to-demonstrate-iot-security-strain. Accessed: 14-10-2016.

[40] Lee Bell. Apple advises users on iCloud security in response to China cyber attack reports, 22 October 2014. URL http://www.theinquirer.net/inquirer/news/2376697/china-targets-apples-icloud-with-man-in-the-middle-cyber-attacks. Accessed: 14-10-2016.

[41] Lee Bell. Belkin fixes home fire and blackout vulnerabilities in its Wemo systems, 19 February 2014. URL http://www.theinquirer.net/inquirer/news/2329796/belkin-fixes-home-fire-and-blackout-vulnerabilities-in-its-wemo-systems. Accessed: 14-10-2016.

[42] The Register Security. Samsung smart fridge leaves Gmail logins open to attack, 24 August 2015. URL http://www.theregister.co.uk/2015/08/24/smart_fridge_security_fubar/. Accessed: 14-10-2016.

[43] The Register Security. Infosec geniuses hack a Canon PRINTER and install DOOM, 15 September 2014. URL http://www.theregister.co.uk/2014/09/15/hacking_printers_to_pla_doom/. Accessed: 14-10-2016.

[44] Katie Mettler. Somebody keeps hacking these Dallas road signs with messages about Donald Trump, Bernie Sanders and Harambe the gorilla, 6 June 2016. URL https://www.washingtonpost.com/news/morning-mix/wp/2016/06/06/somebody-keeps-hacking-these-dallas-road-signs-with-messages-about-donald-trump-bernie-sanders-and-harambe-the-gorilla/. Accessed: 14-10-2016.

[45] Jamie Condliffe. Ransomware Took San Francisco's Public Transit for a Ride, 28 November 2016. URL https://www.technologyreview.com/s/602979/ransomware-took-san-franciscos-public-transit-for-a-ride/. Accessed: 14-10-2016.

[46] Jamie Condliffe. The Internet of Things Goes Rogue, 30 September 2016. URL https://www.technologyreview.com/s/602519/the-internet-of-things-goes-rogue/. Accessed: 14-10-2016.

[47] Tom Simonite. The Hackers' New Weapons: Routers and Printers, 28 April 2015. URL https://www.technologyreview.com/s/537031/the-hackers-new-weapons-routers-and-printers/. Accessed: 14-10-2016.

[48] Erica Naone. Taking Control of Cars From Afar, 14 March 2011. URL https://www.technologyreview.com/s/423292/taking-control-of-cars-from-afar/. Accessed: 14-10-2016.

[49] Dina Fine Maron. A New Cyber Concern: Hack Attacks on Medical Devices, 25 June 2013. URL https://www.scientificamerican.com/article/a-new-cyber-concern-hack/. Accessed: 14-10-2016.

[50] Kim Zetter. Hacker Can Send Fatal Dose to Hospital Drug Pumps, 8 May 2015. URL https://www.wired.com/2015/06/hackers-can-send-fatal-doses-hospital-drug-pumps/. Accessed: 14-10-2016.

[51] Kim Zetter. A Cyberattack Has Caused Confirmed Physical Damage for the Second Time Ever, 8 January 2015. URL https://www.wired.com/2015/01/german-steel-mill-hack-destruction/. Accessed: 14-10-2016.

[52] Andy Greenberg. Hackers Can Disable a Sniper RifleOr Change Its Target, 26 July 2015. URL https://www.wired.com/2015/07/hackers-can-disable-sniper-rifleor-change-target/. Accessed: 14-10-2016.

[53] Andy Greenberg. Hackers Remotely Kill a Jeep on the HighwayWith Me in It, 21 July 2015. URL https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/. Accessed: 14-10-2016.

[54] Andy Greenberg. The Unpatchable Malware That Infects USBs Is Now on the Loose, 2 October 2014. URL https://www.wired.com/2014/10/code-published-for-unfixable-usb-attack/. Accessed: 14-10-2016.

[55] Dave Thier. Blizzard And "World of Warcraft" Taken Down By DDoS Attacks, 14 April 2016. URL www.forbes.com/sites/davidthier/2016/04/14/lizard-squad-blizzard-and-world-of-warcraft-hit-my-multiple-attacks/. Accessed: 14-10-2016.

[56] John Archer. Sony Pictures Hack Could Also Impact Sony's PS4, Phone And TV Business, 18 December 2014. URL http://www.forbes.com/sites/johnarcher/2014/12/18/why-the-sony-pictures-hack-could-wreak-havoc-on-sonys-ps4-phone-and-tv-sales/. Accessed: 14-10-2016.

[57] Parmy Olson. The Largest Cyber Attack In History Has Been Hitting Hong Kong Sites, 20 November 2014. URL http://www.forbes.com/sites/parmyolson/2014/11/20/the-largest-cyber-attack-in-history-has-been-hitting-hong-kong-sites/. Accessed: 14-10-2016.

[58] Elise Ackerman. The U.S. Department Of Homeland Security Warns iPhone, iPad Owners To Beware Of "Masque Attack", 13 November 2014. URL http://www.forbes.com/sites/eliseackerman/2014/11/13/the-u-s-department-of-homeland-security-warns-iphone-ipad-owners-to-beware-of-masque-attack/. Accessed: 14-10-2016.

[59] Leo King. Smart Home? These Connected LED Light Bulbs Could Leak Your Wi-Fi Password, 9 July 2014. URL http://www.forbes.com/sites/leoking/2014/07/09/smart-home-these-connected-led-light-bulbs-could-leak-your-wi-fi-password/. Accessed: 14-10-2016.

[60] Tamlin Magee. Trustwave Demonstrates Malware That Logs Touchscreen Swipes To Record Your PIN, 27 January 2014. URL http://www.forbes.com/sites/tamlinmagee/2014/01/27/trustwave-demonstrates-malware-that-logs-touchscreen-swipes-to-record-your-pin/. Accessed: 14-10-2016.

[61] Andy Greenberg. Hackers Reveal Nasty New Car Attacks–With Me Behind The Wheel, 24 July 2013. URL http://www.forbes.com/sites/andygreenberg/2013/07/24/hackers-reveal-nasty-new-car-attacks-with-me-behind-the-wheel-video/. Accessed: 14-10-2016.

[62] Lucian Constantin. Attackers hijack CCTV cameras to launch DDoS attacks, 22 October 2015. URL http://www.computerworld.com/article/2996079/

internet-of-things/attackers-hijack-cctv-cameras-to-launch-ddos-attacks.html. Accessed: 14-10-2016.

[63] Darlene Storm. Researchers hack a pacemaker, kill a man(nequin), 8 September 2015. URL https://www.computerworld.com/article/2981527/cybercrime-hacking/researchers-hack-a-pacemaker-kill-a-man-nequin.html. Accessed: 14-10-2016.

[64] Darlene Storm. MEDJACK: Hackers hijacking medical devices to create backdoors in hospital networks, 8 June 2015. URL https://www.computerworld.com/article/2932371/cybercrime-hacking/medjack-hackers-hijacking-medical-devices-to-create-backdoors-in-hospital-networks.html. Accessed: 14-10-2016.

[65] Gregg Keizer. Garden-variety DDoS attack knocks North Korea off the Internet, 23 December 2015. URL https://www.computerworld.com/article/2862652/garden-variety-ddos-attack-knocks-north-korea-off-the-internet.html. Accessed: 14-10-2016.

[66] Jaikumar Vijayan. Target attack shows danger of remotely accessible HVAC systems, 7 February 2014. URL https://www.computerworld.com/article/2487452/cybercrime-hacking/target-attack-shows-danger-of-remotely-accessible-hvac-systems.html. Accessed: 14-10-2016.

[67] Darlene Storm. Hackers exploit SCADA holes to take full control of critical infrastructure, 15 January 2014. URL https://www.computerworld.com/article/2475789/cybercrime-hacking/hackers-exploit-scada-holes-to-take-full-control-of-critical-infrastructure.html. Accessed: 14-10-2016.

[68] Lucian Constantin. Researcher hijacks unsecure embedded devices en masse for Internet scanning project, 19 March 2013. URL https://www.computerworld.com/article/2495541/security0/researcher-hijacks-unsecure-embedded-devices-en-masse-for-internet-scanning-project.html. Accessed: 14-10-2016.

[69] Michael Kan. Upgraded Mirai botnet disrupts Deutsche Telekom by infecting routers, 28 November 2016. URL http://www.pcworld.com/article/3145449/security/upgraded-mirai-botnet-disrupts-deutsche-telekom-by-infecting-routers.html. Accessed: 14-10-2016.

[70] Jared Newman. The Internet of Things is all fun and games until a racist takes over your printer, 28 March 2016. URL http://www.pcworld.com/article/3048794/security/the-internet-of-things-is-all-fun-and-games-until-a-racist-takes-over-your-printer.html. Accessed: 14-10-2016.

[71] Jared Newman. Internet-connected Hello Barbie doll can be hacked, 7 December 2015. URL http://www.pcworld.com/article/3012220/security/internet-connected-hello-barbie-doll-can-be-hacked.html. Accessed: 14-10-2016.

[72] Lucian Constantin. Over 100 DDoS botnets built using Linux malware for embedded devices, 30 June 2015. URL http://www.pcworld.com/article/3090430/over-100-ddos-botnets-built-using-linux-malware-for-embedded-devices.html. Accessed: 14-10-2016.

[73] Lucian Constantin. Thousands of hacked CCTV devices used in DDoS attacks, 28 June 2015. URL http://www.pcworld.com/article/3089346/security/thousands-of-hacked-cctv-devices-used-in-ddos-attacks.html. Accessed: 14-10-2016.

[74] Lucian Constantin. Authentication bypass bug exposes Foscam webcams to unauthorized access, 24 January 2014. URL http://www.pcworld.com/article/2091180/authentication-bypass-bug-exposes-foscam-webcams-to-unauthorized-access.html. Accessed: 14-10-2016.

[75] Kaoru Hayashi. Linux Worm Targeting Hidden Devices, 27 November 2013. URL https://www.symantec.com/connect/blogs/linux-worm-targeting-hidden-devices. Accessed: 14-10-2016.

[76] Trend Micro Security News. Security Team Exposes Vulnerabilities in Drones, 14 June 2016. URL http://www.trendmicro.com/vinfo/us/security/news/internet-of-things/security-team-exposes-vulnerabilities-in-drones. Accessed: 14-10-2016.

[77] Trend Micro Security News. Surveillance Cameras Found Embedded with Malware, 12 April 2016. URL http://www.trendmicro.com/vinfo/us/security/news/internet-of-things/surveillance-cameras-found-with-malware. Accessed: 14-10-2016.

[78] Trend Micro Security News. Nissan Leaf Can be Hacked via Mobile App and Web Browser, 26 February 2016. URL http://www.trendmicro.com/vinfo/us/security/news/internet-of-things/nissan-leaf-can-be-hacked-via-mobile-app-and-web-browser. Accessed: 14-10-2016.

[79] Trend Micro Security News. Researchers Discover a Not-So-Smart Flaw In Smart Toy Bear, 4 February 2016. URL http://www.trendmicro.com/vinfo/us/security/news/internet-of-things/researchers-discover-flaw-in-smart-toy-bear. Accessed: 14-10-2016.

[80] Trend Micro Security News. Carjacking by CD? Researcher Shows How a Spiked Song Can Be Used to Hack a Car, 2 February 2016. URL http://www.trendmicro.com/vinfo/us/security/news/internet-of-

things/carjacking-by-cd-research-shows-how-spiked-song-can-hack-a-car. Accessed: 14-10-2016.

[81] Trend Micro Security News. Israel's Electric Authority "Hack" Caused by Ransomware, 29 January 2016. URL http://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/israels-electric-authority-hack-caused-by-ransomware. Accessed: 14-10-2016.

[82] Trend Micro Security News. First Malware-Driven Power Outage Reported in Ukraine, 6 January 2016. URL http://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/first-malware-driven-power-outage-reported-in-ukraine. Accessed: 14-10-2016.

[83] Trend Micro Security News. Car Hacking: The Very Real Possibility of Hackers Driving Your Car, 29 August 2015. URL http://www.trendmicro.com/vinfo/us/security/news/vulnerabilities-and-exploits/car-hacking-the-very-real-possibility-of-hackers-driving-your-car. Accessed: 14-10-2016.

[84] Trend Micro Security News. The Gaspot Experiment: How Gas-Tank-Monitoring Systems Could Make Perfect Targets for Attackers, 6 August 2015. URL http://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/the-gaspot-experiment. Accessed: 14-10-2016.

[85] Trend Micro Security News. Seagate NAS Unpatched Vulnerabilities Put Thousands of Users at Risk, 2 March 2015. URL http://www.trendmicro.com/vinfo/us/security/news/vulnerabilities-and-exploits/seagate-nas-unpatched-vulnerability-puts-thousands-of-users-at-risk. Accessed: 14-10-2016.

[86] Trend Micro Security News. Researchers Discover Weak Security in Many Home Security Systems, 17 February 2015. URL http://www.trendmicro.com/vinfo/us/security/news/internet-of-things/researchers-discover-weak-security-in-home-security-systems. Accessed: 14-10-2016.

[87] Trend Micro Security News. Tampered US Gas Pumps Point to Anonymous Group, 13 February 2015. URL http://www.trendmicro.com/vinfo/us/security/news/internet-of-things/tampered-us-gas-pumps-point-to-anonymous-group. Accessed: 14-10-2016.

[88] Trend Micro Security News. Threats at Sea: A Security Evaluation of AIS, 16 December 2014. URL http://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/a-security-evaluation-of-ais. Accessed: 14-10-2016.

[89] Trend Micro Security News. About the Shellshock Vulnerability: The Basics of the "Bash Bug", 26 September 2014. URL http://www.trendmicro.com/vinfo/us/security/news/vulnerabilities-and-exploits/the-shellshock-vulnerability-bash-bug. Accessed: 14-10-2016.

[90] Kyle Wilhoit and Marco Balduzzi. Vulnerabilities Discovered in Global Vessel Tracking Systems, 15 October 2013. URL http://blog.trendmicro.com/trendlabs-security-intelligence/vulnerabilities-discovered-in-global-vessel-tracking-systems/. Accessed: 14-10-2016.

[91] Daniel Cid. IoT Home Router Botnet Leveraged in Large DDoS Attack, 1 September 2016. URL https://blog.sucuri.net/2016/09/iot-home-router-botnet-leveraged-in-large-ddos-attack.html. Accessed: 14-10-2016.

[92] Daniel Cid. Large CCTV Botnet Leveraged in DDoS Attacks, 27 June 2016. URL https://blog.sucuri.net/2016/06/large-cctv-botnet-leveraged-ddos-attacks.html. Accessed: 14-10-2016.

[93] Selena Larson. A smart fish tank left a casino vulnerable to hackers , 19 July 2017. URL http://money.cnn.com/2017/07/19/technology/fish-tank-hack-darktrace/index.html. Accessed: 26-07-2017.

[94] Andy Greenberg. Hack Brief: 'Devil's Ivy' Vulnerability Could Afflict Millions of IoT Devices, 18 July 2017. URL https://www.wired.com/story/devils-ivy-iot-vulnerability/. Accessed: 26-07-2017.

[95] Robert Graham. MASSCAN: Mass IP port scanner. URL http://tools.kali.org/information-gathering/masscan. Accessed: 24-01-2017.

[96] Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009. URL https://nmap.org/.

[97] John Matherly. *Complete Guide to Shodan*. Shodan, LLC, 2016. URL https://www.shodan.io/.

[98] Robert McMillan. Tour the World's Webcams With the Search Engine for the Internet of Things, 7 August 2013. URL https://www.wired.com/2013/07/shodan-search-engine/. Accessed: 24-01-2017.

[99] Levente Buttyan and Jean-Pierre Hubaux. *Security and cooperation in wireless networks: thwarting malicious and selfish behavior in the age of ubiquitous computing*. Cambridge University Press, 2007.

[100] Microsoft Corporation. Microsoft security intelligence report. *Microsoft Secur. Intell. Rep*, 22:1–74, 2017.

[101] Bill Nelson, Amelia Phillips, and Christopher Steuart. *Guide to Computer Forensics and Investigations*. Cengage Learning, 2014. ISBN 9781305176089.

[102] Erwin Adi, Zubair A Baig, Philip Hingston, and Chiou-Peng Lam. Distributed denial-of-service attacks against HTTP/2 services. *Cluster Computing*, 19(1):79–86, 2016.

[103] Yasir Mehmood, Muhammad Awais Shibli, Ayesha Kanwal, and Rahat Masood. Distributed intrusion detection system using mobile agents in cloud computing environment. In *2015 Conference on Information Assurance and Cyber Security (CIACS)*, pages 1–8. IEEE, 2015.

[104] Primož Cigoj and Borka Jerman Blažič. An Innovative Approach in Digital Forensic Education and Training. In *IFIP World Conference on Information Security Education*, pages 101–110. Springer, 2015.

[105] Marwan Darwish, Abdelkader Ouda, and Luiz Fernando Capretz. Cloud-based DDoS attacks and defenses. In *2013 International Conference on Information Society (i-Society)*, pages 67–71. IEEE, 2013.

[106] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip feng shui: Hammering a needle in the software stack. In *Proceedings of the 25th USENIX Security Symposium*, 2016.

[107] Prachi Deshpande, Aditi Aggarwal, SC Sharma, P Sateesh Kumar, and Ajith Abraham. Distributed port-scan attack in cloud environment. In *2013 5th International Conference on Computational Aspects of Social Networks (CASoN)*, pages 27–31. IEEE, 2013.

[108] Zahra Jadidi, Vallipuram Muthukkumarasamy, Elankayer Sithirasenan, and Kalvinder Singh. Flow-based anomaly detection using semisupervised learning. In *2015 9th International Conference on Signal Processing and Communication Systems (ICSPCS)*, pages 1–5. IEEE, 2015.

[109] Kai Axford. Security in a Virtual World, 2016. URL https://technet.microsoft.com/en-gb/library/cc974514.aspx. Accessed: 20-10-2016.

[110] Valerio Selis and Alan Marshall. A classification-based algorithm to detect forged embedded machines in IoT environments. *IEEE Systems Journal*, 2017. (Accepted with minor revisions).

[111] Valerio Selis and Alan Marshall. A Fake Timing Attack Against Behavioural Tests Used in Embedded IoT M2M Communications. In *2017 1st Cyber Security in Networking Conference (CSNet'17)*, Rio de Janeiro, Brazil, October 2017. IEEE.

[112] Hans Vestburg. CEO to shareholders: 50 billion connections 2020, 13 April 2010. URL https://www.ericsson.com/thecompany/press/releases/2010/04/1403231. Accessed: 16-10-2016.

[113] Dave Evans. The Internet of Things, April 2011. URL https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf. Accessed: 16-10-2016.

[114] Jon Iwata. Making Markets: Smarter Planet, May 2012. URL http://www.ibm.com/investor/events/investor0512/presentation/05_Smarter_Planet.pdf. Accessed: 16-10-2016.

[115] IHS Markit. Making Markets: Smarter Planet, 11 April 2016. URL http://news.ihsmarkit.com/press-release/technology/tech-companies-creating-strategic-platforms-support-internet-things-ihs-say. Accessed: 16-10-2016.

[116] Buckland Emma, Margaret Ranken, Matt Arnott, and Pierce Owen. IoT Global Forecast & Analysis 2015-25, 5 August 2016. URL https://machinaresearch.com/news/press-release-global-internet-of-things-market-to-grow-to-27-billion-devices-generating-usd3-trillion-revenue-in-2025/. Accessed: 21-10-2016.

[117] Ericsson. Ericsson Mobility Report, November 2016. URL https://www.ericsson.com/assets/local/mobility-report/documents/2016/ericsson-mobility-report-november-2016.pdf. Accessed: 16-10-2016.

[118] Jim Morrish and Matt Hatton. The Connected Life, 11 October 2011. URL https://machinaresearch.com/static/media/uploads/machina_research_press_release_gsma_2010_20.pdf. Accessed: 21-10-2016.

[119] Cisco Systems. Connections Counter: The Internet of Everything in Motion, 29 July 2013. URL https://newsroom.cisco.com/feature-content?articleId=1208342. Accessed: 16-10-2016.

[120] Jim Morrish. The Connected Life, 24 June 2013. URL http://www.gsma.com/connectedliving/wp-content/uploads/2013/03/JimMorrish_GSMA-Connected-Life-20130624-v4.pdf. Accessed: 21-10-2016.

[121] Vernon Turner Denise Lund, Carrie MacGillivray and Mario Morales. Worldwide and Regional Internet of Things (IoT) 2014-2020. Forecast: A Virtuous Circle of Proven Value and Demand, May 2014. URL https://www.business.att.com/content/article/IoT-worldwide_regional_2014-2020-forecast.pdf. Accessed: 16-10-2016.

[122] J. Rivera and R. van der Meulen. Gartner Says 4.9 Billion Connected "Things" Will Be in Use in 2015, 11 November 2014. URL http://www.gartner.com/newsroom/id/2905717. Accessed: 16-10-2016.

[123] NCTA The Internet & Television Associatio. Behind The Numbers: Growth in the Internet of Things., 20 March 2015. URL https://www.ncta.com/platform/broadband-internet/behind-the-numbers-growth-in-the-internet-of-things/. Accessed: 22-10-2016.

[124] BI Intelligence. The Internet of Everything: 2015, 8 April 2015. URL http://uk.businessinsider.com/internet-of-everything-2015-bi-2014-12. Accessed: 19-10-2016.

[125] Rob van der Meulen. Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015, 10 November 2015. URL http://www.gartner.com/newsroom/id/3165317. Accessed: 16-10-2016.

[126] Ericsson. Ericsson Mobility Report, November 2015. URL http://www.ericsson.com/res/docs/2015/mobility-report/ericsson-mobility-report-nov-2015.pdf. Accessed: 16-10-2016.

[127] BI Intelligence. Here's how the Internet of Things will explode by 2020, 1 September 2016. URL http://uk.businessinsider.com/iot-ecosystem-internet-of-things-forecasts-and-business-opportunities-2016-2. Accessed: 19-10-2016.

[128] Chetan Sharma. Correcting the IoT History. URL http://www.chetansharma.com/IoT_History.htm. Accessed: 25-01-2017.

[129] Sabina Jeschke, Christian Brecher, Houbing Song, and Danda B Rawat. Industrial Internet of Things.

[130] Luigi Atzori, Antonio Iera, and Giacomo Morabito. Understanding the Internet of Things: definition, potentials, and societal role of a fast evolving paradigm. *Ad Hoc Networks*, 2016.

[131] Faisal Karim Shaikh, Sherali Zeadally, and Ernesto Exposito. Enabling technologies for green internet of things. *IEEE Systems Journal*, 2015.

[132] Ivan Stojmenovic. Machine-to-machine communications with in-network data aggregation, processing, and actuation for large-scale cyber-physical systems. *IEEE Internet of Things Journal*, 1(2):122–128, 2014.

[133] John B. Kennedy. When Woman Is Boss: An interview with Nikola Tesla, 1926. URL http://www.tfcbooks.com/tesla/1926-01-30.htm. Accessed: 17-02-2017.

[134] Internet of Things. URL https://en.oxforddictionaries.com/definition/internet_of_things. Accessed: 25-01-2017.

[135] Lu Tan and Neng Wang. Future internet: The internet of things. In *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, volume 5, pages V5–376. IEEE, 2010.

[136] Daniel Giusto, Antonio Iera, Giacomo Morabito, and Luigi Atzori. *The internet of things: 20th Tyrrhenian workshop on digital communications*. Springer Science & Business Media, 2010.

[137] ITU-T Y.4000/Y.2060. Overview of the Internet of things, 2012. URL http://www.itu.int/ITU-T/recommendations/rec.aspx?rec=11559. Accessed: 27-01-2017.

[138] Roberto Minerva, Abyi Biru, and Domenico Rotondi. Towards a definition of the Internet of Things (IoT). *IEEE Internet Initiative*, 2015. URL http://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf. Accessed: 13-02-2017.

[139] Luigi Atzori, Antonio Iera, Giacomo Morabito, and Michele Nitti. The social internet of things (siot)–when social networks meet the internet of things: Concept, architecture and network characterization. *Computer networks*, 56(16):3594–3608, 2012.

[140] Luigi Atzori, Antonio Iera, and Giacomo Morabito. Siot: Giving a social structure to the internet of things. *IEEE communications letters*, 15(11):1193–1195, 2011.

[141] Sarfraz Alam, Mohammad MR Chowdhury, and Josef Noll. Interoperability of security-enabled internet of things. *Wireless Personal Communications*, 61(3): 567–586, 2011.

[142] iCore Empowering IoT through Cognitive Technologies. Internet Connected Objects for Reconfigurable Ecosystem D2.5 – Final architecture reference Model. 2014. URL http://www.iot-icore.eu/attachments/article/89/20141031_final_architecture.pdf. Accessed: 08-03-2017.

[143] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the 1st edition of the MCC workshop on Mobile Cloud Computing*, pages 13–16. ACM, 2012.

[144] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. Fog computing: A platform for internet of things and analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*, pages 169–186. Springer, 2014.

[145] Gert De Laet and Gert Schauwers. *Network Security Fundamentals*. Cisco Press, 2005.

[146] Aikaterini Mitrokotsa, Melanie R Rieback, and Andrew S Tanenbaum. Classifying RFID attacks and defenses. *Information Systems Frontiers*, 12(5):491–505, 2010.

[147] Jonny Milliken, Valerio Selis, and Alan Marshall. Detection and analysis of the Chameleon WiFi access point virus. *EURASIP Journal on Information Security*, 2013(1):2, 2013.

[148] Changhua He John C Mitchell. Security Analysis and Improvements for IEEE 802.11i. In *The 12th Annual Network and Distributed System Security Symposium (NDSS'05), Stanford University, Stanford*, pages 90–110. Citeseer, 2005.

[149] Kemal Bicakci and Bulent Tavli. Denial-of-Service attacks and countermeasures in IEEE 802.11 wireless networks. *Computer Standards & Interfaces*, 31(5):931–941, 2009.

[150] Erik Tews and Martin Beck. Practical attacks against WEP and WPA. In *Proceedings of the 2nd ACM conference on Wireless Network Security*, pages 79–86. ACM, 2009.

[151] Thomas dOtreppe. Aircrack-ng, 2013.

[152] Kristopher Kendall. A database of computer attacks for the evaluation of intrusion detection systems. Technical report, DTIC Document, 1999.

[153] David R Raymond and Scott F Midkiff. Denial-of-service in wireless sensor networks: Attacks and defenses. *IEEE Pervasive Computing*, 7(1), 2008.

[154] Bjorn Stelte and Gabi Dreo Rodosek. Thwarting attacks on ZigBee-Removal of the KillerBee stinger. In *2013 9th International Conference on Network and Service Management (CNSM)*, pages 219–226. IEEE, 2013.

[155] Ulrike Meyer and Susanne Wetzel. A man-in-the-middle attack on UMTS. In *Proceedings of the 3rd ACM workshop on Wireless security*, pages 90–97. ACM, 2004.

[156] Radmilo Racic, Denys Ma, Hao Chen, and Xin Liu. Exploiting Opportunistic Scheduling in Cellular Data Networks. In *NDSS*. Citeseer, 2008.

[157] Soshant Bali, Sridhar Machiraju, Hui Zang, and Victor Frost. A measurement study of scheduler-based attacks in 3G wireless networks. In *International Conference on Passive and Active Network Measurement*, pages 105–114. Springer, 2007.

[158] Radmilo Racic, Denys Ma, and Hao Chen. Exploiting MMS vulnerabilities to stealthily exhaust mobile phone's battery. In *2006 Securecomm and Workshops*, pages 1–10. IEEE, 2006.

[159] Andreas Berger, Ivan Gojmerac, and Oliver Jung. Internet security meets the IP multimedia subsystem: an overview. *Security and Communication Networks*, 3 (2-3):185–206, 2010.

[160] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[161] Khaled Abdulla Al Rabaiei and Saad Harous. Internet of things: Applications and challenges. In *2016 12th International Conference on Innovations in Information Technology (IIT)*, pages 1–6. IEEE, 2016.

[162] Zhihua Hu. The research of several key question of internet of things. In *2011 International Conference on Intelligence Science and Information Engineering (ISIE)*, pages 362–365. IEEE, 2011.

[163] Gang Gan, Zeyong Lu, and Jun Jiang. Internet of things security analysis. In *2011 International Conference on Internet Technology and Applications (iTAP)*, pages 1–4. IEEE, 2011.

[164] G.A. Duckett. *Encryption: Questions and Answers*. CreateSpace Independent Publishing Platform, 2016. ISBN 9781533418395. URL https://books.google.co.uk/books?id=UrX5jwEACAAJ.

[165] William Stallings. *Cryptography and network security: principles and practice*. Pearson Education India, 2003.

[166] Miodrag Potkonjak, Saro Meguerdichian, Ani Nahapetian, and Sheng Wei. Differential public physically unclonable functions: architecture and applications. In *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 242–247. IEEE, 2011.

[167] Jorge Guajardo, Sandeep S Kumar, Geert-Jan Schrijen, and Pim Tuyls. Physical unclonable functions and public-key crypto for FPGA IP protection. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 189–195. IEEE, 2007.

[168] G Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual design automation conference*, pages 9–14. ACM, 2007.

[169] Nathan Beckmann and Miodrag Potkonjak. Hardware-based public-key cryptography with public physically unclonable functions. In *International Workshop on Information Hiding*, pages 206–220. Springer, 2009.

[170] Jian Yin and Sanjay Kumar Madria. Sybil attack detection in a hierarchical sensor network. In *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops (SecureComm 2007)*, pages 494–503. IEEE, 2007.

[171] Gustaf Ouvrier, Michel Laterman, Martin Arlitt, and Niklas Carlsson. Characterizing the HTTPS trust landscape: a passive view from the edge. *IEEE Communications Magazine*, 55(7):36–42, 2017.

[172] Trust. URL http://www.dictionary.com/browse/trust. Accessed: 02-04-2017.

[173] Trust. URL https://www.merriam-webster.com/dictionary/trust. Accessed: 02-04-2017.

[174] Audun Jøsang. Artificial reasoning with subjective logic. In *Proceedings of the 2nd Australian workshop on commonsense reasoning*, volume 48, page 34. Citeseer, 1997.

[175] Tyrone Grandison and Morris Sloman. A survey of trust in internet applications. *IEEE Communications Surveys & Tutorials*, 3(4):2–16, 2000.

[176] Jin-Hee Cho, Ananthram Swami, and Ray Chen. A survey on trust management for mobile ad hoc networks. *IEEE Communications Surveys & Tutorials*, 13(4): 562–583, 2011.

[177] Wanita Sherchan, Surya Nepal, and Cecile Paris. A survey of trust in social networks. *ACM Computing Surveys (CSUR)*, 45(4):47, 2013.

[178] Guo Ya-Jun, Hong Fan, Zhang Qing-Guo, and Li Rong. An access control model for ubiquitous computing application. 2005.

[179] Ji Guo, Alan Marshall, and Bosheng Zhou. A Multi-Parameter Trust Framework for Mobile Ad Hoc Networks. In *Security, Privacy, Trust, and Resource Management in Mobile and Wireless Communications*, pages 245–277. IGI Global, 2014.

[180] D Harrison McKnight and Norman L Chervany. The meanings of trust. 1996.

[181] Florina Almenárez, Andrés Marín, Celeste Campo, and Carlos Garcia. PTM: A pervasive trust management model for dynamic open environments. In *First Workshop on Pervasive Security, Privacy and Trust (PSPT)*, volume 4, pages 1–8, 2004.

[182] Yan Lindsay Sun, Wei Yu, Zhu Han, and KJ Ray Liu. Information theoretic framework of trust modeling and evaluation for ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 24(2):305–317, 2006.

[183] Munirul M Haque and Sheikh I Ahamed. An omnipresent formal trust model (FTM) for pervasive computing environment. In *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, volume 1, pages 49–56. IEEE, 2007.

[184] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE, 1996.

[185] Audun Jøsang, Claudia Keser, and Theo Dimitrakos. Can we manage trust? In *International Conference on Trust Management*, pages 93–107. Springer, 2005.

[186] Yosra Ben Saied, Alexis Olivereau, Djamal Zeghlache, and Maryline Laurent. Trust management system design for the Internet of Things: A context-aware and multi-service approach. *Computers & Security*, 39:351–365, November 2013. ISSN 01674048. doi: 10.1016/j.cose.2013.09.001. URL http://www.sciencedirect.com/science/article/pii/S0167404813001302.

[187] Dong Chen, Guiran Chang, Dawei Sun, Jiajia Li, Jie Jia, and Xingwei Wang. TRM-IoT: A trust management model based on fuzzy reputation for internet of things, 2011. ISSN 1820-0214.

[188] Zhikui Chen, Ruochuan Ling, Chung-Ming Huang, and Xu Zhu. A scheme of access service recommendation for the Social Internet of Things. *International Journal of Communication Systems*, February 2015. ISSN 10745351. doi: 10.1002/dac.2930. URL http://doi.wiley.com/10.1002/dac.2930.

[189] Fenye Bao and Ing-Ray Chen. Trust management for the internet of things and its application to service composition. In *2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6. IEEE, June 2012. ISBN 978-1-4673-1239-4. doi: 10.1109/WoWMoM.2012.6263792. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6263792.

[190] Fenye Bao and Ing-Ray Chen. Dynamic trust management for internet of things applications. In *Proceedings of the 2012 international workshop on Self-aware internet of things (Self-IoT '12)*, page 1, New York, New York, USA, September 2012. ACM Press. ISBN 9781450317535. doi: 10.1145/2378023.2378025. URL http://dl.acm.org/citation.cfm?id=2378023.2378025.

[191] Fenye Bao, Ing-Ray Chen, and Jia Guo. Scalable, adaptive and survivable trust management for community of interest based Internet of Things systems. In *2013 IEEE 11th International Symposium on Autonomous Decentralized Systems (ISADS)*, pages 1–7. IEEE, March 2013. ISBN 978-1-4673-5070-9. doi: 10.1109/ISADS.2013.6513398. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6513398.

[192] Ing-Ray Chen, Jia Guo, and Fenye Bao. Trust Management for SOA-based IoT and Its Application to Service Composition. *IEEE Transactions on Services Computing*, PP(99):1–1, 2014. ISSN 1939-1374. doi: 10.1109/TSC.2014.2365797. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6940301.

[193] Michele Nitti, Roberto Girau, and Luigi Atzori. Trustworthiness Management in the Social Internet of Things. *IEEE Transactions on Knowledge and Data Engineering*, 26(5):1253–1266, May 2014. ISSN 1041-4347. doi: 10.1109/TKDE.

2013.105. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6547148.

[194] Ruidong Li, Jie Li, Peng Liu, and Hsiao-Hwa Chen. An objective trust management framework for mobile ad hoc networks. In *IEEE 65th Vehicular Technology Conference (VTC2007)*, pages 56–60. IEEE, 2007.

[195] Girish Suryanarayana, Mamadou H Diallo, Justin R Erenkrantz, and Richard N Taylor. Architectural support for trust models in decentralized applications. In *Proceedings of the 28th International Conference on Software Engineering*, pages 52–61. ACM, 2006.

[196] Ji Guo, Alan Marshall, and Bosheng Zhou. A New Trust Management Framework for Detecting Malicious and Selfish Behaviour for Mobile Ad Hoc Networks. In *10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 142–149. IEEE, November 2011. ISBN 978-1-4577-2135-9. doi: 10.1109/TrustCom.2011.21. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6120813.

[197] Mohammad Karami and Mohammad Fathian. A robust trust establishment framework using Dempster-Shafer theory for MANETs. In *International Conference for Internet Technology and Secured Transactions (ICITST 2009)*, pages 1–7. IEEE, 2009.

[198] Laurent Eschenauer, Virgil D Gligor, and John Baras. On trust establishment in mobile ad-hoc networks. In *International Workshop on Security Protocols*, pages 47–66. Springer, 2002.

[199] Jia Guo, Ray Chen, and Jeffrey JP Tsai. A survey of trust computation models for service management in internet of things systems. *Computer Communications*, 97:1–14, 2017.

[200] Yan Sun, Zhu Han, and KJ Ray Liu. Defense of trust management vulnerabilities in distributed networks. *IEEE Communications Magazine*, 46(2):112–119, 2008.

[201] Fenye Bao, Ing-Ray Chen, MoonJeong Chang, and Jin-Hee Cho. Hierarchical Trust Management for Wireless Sensor Networks and its Applications to Trust-Based Routing and Intrusion Detection, 2012. ISSN 1932-4537.

[202] S Prasanna and V Vetriselvi. An improved intrusion detection technique for mobile adhoc networks. In *International Conference on Distributed Computing and Internet Technology*, pages 364–376. Springer, 2005.

[203] Jia Guo and Ing-Ray Chen. A Classification of Trust Computation Models for Service-Oriented Internet of Things Systems, 2015. URL http://people.cs.vt.edu/{~}irchen/ps/Guo-scc15.pdf.

[204] Michele Nitti, Roberto Girau, Luigi Atzori, Antonio Iera, and Giacomo Morabito. A subjective model for trustworthiness evaluation in the social Internet of Things. In *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 18–23. IEEE, September 2012. ISBN 978-1-4673-2569-1. doi: 10.1109/PIMRC.2012.6362662. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6362662.

[205] VMware, Inc. Configuration Maximums - vSphere 6.0, 2017. URL https://www.vmware.com/pdf/vsphere6/r60/vsphere-60-configuration-maximums.pdf.

[206] Joanna Rutkowska. Red Pill: Detect VMM using (almost) One CPU Instruction, 2004. URL http://web.archive.org/web/20041130172213/http://invisiblethings.org/papers/redpill.html.

[207] Lorenzo Martignoni, Roberto Paleari, Giampaolo Fresi Roglia, and Danilo Bruschi. Testing CPU Emulators. *Proceedings of the 18th International Symposium on Software Testing and Analysis*, pages 261–272, 2009. doi: 10.1145/1572272.1572303. URL http://doi.acm.org/10.1145/1572272.1572303.

[208] Hao Shi, Abdulla Alwabel, and Jelena Mirkovic. Cardinal Pill Testing of System Virtual Machines. In *USENIX Security*, pages 271–285, 2014. URL http://dblp.uni-trier.de/db/conf/uss/uss2014.html{#}ShiAM14.

[209] Thomas Raffetseder, Christopher Kruegel, and Engin Kirda. Detecting System Emulators. *Information Security*, pages 1–18, 2007. ISSN 0302-9743. doi: http://dx.doi.org/10.1109/SAINT.2010.108.

[210] Jia-Bin Wang, Yi-Feng Lian, and Kai Chen. Virtualization detection based on data fusion. In *2012 International Conference on Computer Science and Information Processing (CSIP)*, pages 393–396. IEEE, August 2012. ISBN 978-1-4673-1411-4. doi: 10.1109/CSIP.2012.6308876. URL http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6308876.

[211] Xu Chen, Jon Andersen, Z. Morley Mao, Michael Bailey, and Jose Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. *Proceedings of the International Conference on Dependable Systems and Networks*, pages 177–186, 2008. ISSN 1530-0889. doi: 10.1109/DSN.2008.4630086.

[212] Tadayoshi Kohno, Andre Broido, and K. C. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005. ISSN 15455971. doi: 10.1109/TDSC.2005.26.

[213] Van Jacobson, Robert Braden, and David Borman. TCP extensions for high performance. 1992.

[214] Libor Polčák and Barbora Franková. On Reliability of Clock-skew-based Remote Computer Identification. In *Proceedings of the 11th International Conference on Security and Cryptography*, pages 291–298. SCITEPRESS - Science and and Technology Publications, 2014. ISBN 978-989-758-045-1. doi: 10.5220/0005048502910298. URL http://dblp.uni-trier.de/db/conf/secrypt/secrypt2014.html{#}PolcakF14.

[215] Libor Polčák, Jakub Jirasek, and Petr Matousek. Comment on Remote Physical Device Fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 11(5):494–496, September 2014. ISSN 1545-5971. doi: 10.1109/TDSC.2013.26. URL http://www.computer.org/csdl/trans/tq/preprint/06547150-abs.html.

[216] Libor Polčák and Barbora Franková. Clock-Skew-Based Computer Identification: Traps and Pitfalls. *Journal of Universal Computer Science*, 21(9):1210–1233, 2015. URL http://dblp.uni-trier.de/db/journals/jucs/jucs21.html{#}PolcakF15.

[217] A. L. Ortega. *MAC Changer*, 2013. URL http://www.gnu.org/software/macchanger.

[218] Danny Quist and Val Smith. Further down the VM spiral. *Offensive Computing*, 2006. URL http://dl.packetstormsecurity.net/papers/general/dquist_valsmith_further_down_the_vm_spiral.pdf.

[219] Yiming Jing, Ziming Zhao, Gail-Joon Ahn, and Hongxin Hu. Morpheus. In *Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC'14)*, pages 216–225, New York, New York, USA, December 2014. ACM Press. ISBN 9781450330053. doi: 10.1145/2664243.2664250. URL http://dl.acm.org/citation.cfm?id=2664243.2664250.

[220] Timothy Vidas and Nicolas Christin. Evading android runtime analysis via sandbox detection. In *Proceedings of the 9th ACM symposium on Information, computer and communications security (ASIA CCS'14)*, pages 447–458, 2014. ISBN 9781450328005. doi: 10.1145/2590296.2590325. URL http://dl.acm.org/citation.cfm?doid=2590296.2590325.

[221] Cynthia E Irvine, John Scott Robin, et al. Analysis of the Intel Pentium's ability to support a secure virtual machine monitor. Proceedings of the 9th USENIX Security Symposium, Denver, CO., 2000.

[222] Samuel T. King, Peter M. Chen, Yi Min Wang, Chad Verbowski, Helen J. Wang, and Jacob R. Lorch. SubVirt: Implementing malware with virtual machines. In *Proceedings of the IEEE Symposium on Security and Privacy*, volume 2006, pages 314–327, 2006. ISBN 0769525741. doi: 10.1109/SP.2006.38.

[223] Danny B Lange and Mitsuru Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88–89, 1999.

[224] Bo Chen, Harry H Cheng, and Joe Palen. Mobile-C: a mobile agent platform for mobile C/C++ agents. *Software: Practice and Experience*, 36(15):1711–1733, 2006.

[225] IEEE Foundation for Intelligent Physical Agents. The Foundation for Intelligent Physical Agents. URL http://www.fipa.org/. Accessed: 07-04-2018.

[226] Alexandru Suna and Amal El Fallah-Seghrouchni. A mobile agents platform: architecture, mobility and security elements. In *International Workshop on Programming Multi-Agent Systems*, pages 126–146. Springer, 2004.

[227] PC Engines GmbH. ALIX 6F2 System Board, 2007. URL http://www.pcengines.ch/alix6f2.htm. Accessed: 21-04-2017.

[228] Google and LG Electronics. Nexus 5 Tech Specs, 2013. URL https://support.google.com/nexus/answer/6102470?hl=en. Accessed: 21-04-2017.

[229] Google and Asus. Nexus 7 (2012) Tech Specs (32GB + Mobile Data), 2012. URL https://support.google.com/nexus/answer/6102470?hl=en. Accessed: 21-04-2017.

[230] 8devices. Carambola, 2012. URL http://www.8devices.com/products/carambola. Accessed: 21-04-2017.

[231] Arduino. Arduino Board Yún, 2013. URL http://arduino.cc/en/Main/ArduinoBoardYun. Accessed: 21-04-2017.

[232] Raspberry Pi Foundation. Early versions of the Raspberry Pi Model B, 2012. URL https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2836/README.md. Accessed: 21-04-2017.

[233] Android Developers. SDK Tools - Android Emulator, 2014. URL https://developer.android.com/studio/run/emulator.html. Accessed: 23-04-2017.

[234] Genymobile. Genymotion, 2014. URL https://docs.genymotion.com/Content/Home.htm. Accessed: 23-04-2017.

[235] A. Gavare. GXemul, 2014. URL http://gxemul.sourceforge.net/gxemul-stable/doc/index.html. Accessed: 23-04-2017.

[236] Open Virtual Platform. OVPsim, 2014. URL http://www.ovpworld.org/technology_ovpsim.php. Accessed: 23-04-2017.

[237] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[238] Oracle Corporation. VirtualBox, 2014. URL https://www.virtualbox.org/manual/UserManual.html. Accessed: 23-04-2017.

[239] VMware Inc. VMware Player, 2015. URL http://www.vmware.com/uk/products/player/faqs.html. Accessed: 23-04-2017.

[240] Free Software Foundation. GNU Compiler Collection - ARM Options. URL https://gcc.gnu.org/onlinedocs/gcc-3.3.6/gcc/ARM-Options.html. Accessed: 20-08-2017.

[241] Arm Holdings. ARM7 processors. URL http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.arm7/index.html. Accessed: 07-04-2018.

[242] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006. ISSN 08856125. doi: 10.1007/s10994-006-6226-1.

[243] Re Fan, Kw Chang, and Cj Hsieh. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning*, 9(2008):1871–1874, 2008. ISSN 15324435. doi: 10.1038/oby.2011.351. URL http://dl.acm.org/citation.cfm?id=1442794.

[244] Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning*, 46:389–422, 2002. ISSN 08856125. doi: 10.1023/A:1012487302797.

[245] Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen. *Classification and Regression Trees*, volume 19. 1984. ISBN 0412048418.

[246] J. Ross Quinlan. *C4.5: Programs for Machine Learning*, volume 1. 1993. ISBN 1558602380. doi: 10.1016/S0019-9958(62)90649-6. URL http://portal.acm.org/citation.cfm?id=152181.

[247] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Number 2. 2009. ISBN 9780387848570. doi: 10.1007/b94608. URL http://www.springerlink.com/index/D7X7KX6772HQ2135.pdf.

[248] Harry Zhang. The Optimality of Naive Bayes. *Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2004)*, 1(2): 1 – 6, 2004. ISSN 01678655. doi: 10.1016/j.patrec.2005.12.001. URL http://www.aaai.org/Papers/FLAIRS/2004/Flairs04-097.pdf.

[249] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001. ISSN 08856125. doi: 10.1023/A:1010933404324. URL http://portal.acm.org/citation.cfm?id=570182{%}5Cnhttp://www.springerlink.com/content/u0p06167n6173512.

[250] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class AdaBoost. *Statistics and Its Interface*, 2(3):349–360, 2009. ISSN 19387989. doi: 10.4310/SII.2009.v2.n3.a8. URL http://www.stat.lsa.umich.edu/{~}jizhu/pubs/Zhu-SII09.pdfhttp://www.intlpress.com/site/pub/pages/journals/items/sii/content/vols/0002/0003/00024013/http://www.intlpress.com/site/pub/pages/journals/items/sii/content/vols/0002/0003/a008/.

[251] I Guyon, B Boser, and V Vapnik. Automatic Capacity Tuning of Very Large VC-Dimension Classifiers. *Advances in Neural Information Processing Systems*, 5:147–155, 1993. URL http://www.clopinet.com/isabelle/Papers/autocapa.ps.

[252] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. ISSN 08856125. doi: 10.1007/BF00994018.

[253] Marco Benocci, Elisabetta Farella, and Luca Benini. A context-aware smart seat. In *2011 4th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*, pages 104–109. IEEE, 2011.

[254] Sebastian Buschjäger and Katharina Morik. Decision Tree and Random Forest Implementations for Fast Filtering of Sensor Data. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2017.