TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ANTTI KOLEHMAINEN

REMOTE SOFTWARE BASED ATTESTATION IN THE INTERNET OF THINGS

Master of Science thesis

Examiners: Prof. Jarmo Harju and MSc. Joona Kannisto
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering
on 4th March 2015

ABSTRACT

ANTTI KOLEHMAINEN: Remote Software Based Attestation in the Internet of Things
Tampere University of Technology
Master of Science thesis, 48 pages
May 2016
Master's Degree Programme in Information Technology
Major: Data Security
Examiners: Prof. Jarmo Harju, MSc. Joona Kannisto
Keywords: IoT, Attestation, Embedded systems

When in the old days the Internet consisted mostly of workstations, servers, mainframes and networking devices, the rise of the Internet of Things has brought along smart embedded systems that are aware of their surroundings, make their own decisions and communicate with each other accordingly. These systems can be anything and anywhere from a lamp to a refrigerator.

These systems require mutual trust and their integrity has to be monitored. One way to achieve this is to use attestation. Attestation is a process that is used for ensuring trust and integrity of a device. Another important factor in designing IoT devices is their cost-effectiveness. It is desirable for the devices to be cheap to manufacture so any extra hardware might become costly. One mechanism that helps to create attestation without extra hardware is to use software based attestation.

The replacement of hardware attestation with software mechanisms enable faster provisioning of IoT devices to the network. One problem is that usually in IoT case the attestation traffic is communicated over insecure channels where an attacker might be listening. Another thing to be taken into consideration is the physical security, the theft of the device and its effects. One good thing of software-based attestation is the platform agnosticism.

TIIVISTELMÄ

ANTTI KOLEHMAINEN: Ohjelmistopohjainen etätodentaminen asioiden internetissä
Tampereen teknillinen yliopisto
Diplomityö, 48 sivua
Toukokuu 2016
Tietotekniikan koulutusohjelma
Pääaine: Tietoturvallisuus
Tarkastajat: Prof. Jarmo Harju, DI Joona Kannisto
Avainsanat: IoT, Attestointi, Sulautetut järjestelmät

Kun ennen Internetissä keskustelivat lähinnä työasemat, palvelimet, keskustietokoneet ja verkkolaitteet, ovat IoT:n mukana samaan yhtälöön tulleet mukaan myös erilaiset sensoreilla varustetut älykkäät sulautetut järjestelmät, jotka havainnoivat ympäristöään, tekevät päätöksiä toiminnastaan ja keskustelevat keskenään näiden päätösten perusteella. Järjestelmät voivat olla mitä ja missä vain kattolampusta jääkaappiin.

Näiden järjestelmien välille on rakennettava luottamus ja niiden eheyttä on valvottava mahdollisten muutosten varalta. Yksi keino näiden asioiden toteuttamiseen on attestointi. Attestoinnilla tarkoitetaan prosessia, jolla varmennetaan laitteiden luotettavuus ja koskemattomuus. IoT-laitteiden suunnittelussa on hyvä ottaa huomioon myös niiden kustannustehokkuus. Laitteista halutaan tehdä halpoja sekä yksinkertaisia valmistaa, jolloin ylimääräiset piirit saattavat koitua liian kalliiksi. Yksi mekanismi attestoinnin toteuttamiseen tavanomaisilla laitteistoilla, joissa ei välttämättä ole erityistä rautaa, on ohjelmistopohjainen attestointi.

Tavat, joilla rautapohjaisen attestaation mahdollistamia palveluita voidaan korvata ja raskaampia prosesseja suorittaa erillisillä laitteilla, mahdollistavat tehokkaan IoT-verkon laajentamisen. Huomioon on kuitenkin otettava se, että IoT:n tapauksessa attestaatioliikenne liikkuu verkossa, mahdollisesti Internetin yli, ja onkin oletettava että hyökkäjä salakuuntelee sitä. Toinen asia, jota on hyvä kehittää, on laitteiden fyysisen turvallisuuden vaikutus, eli mahdollinen laitteen varkaus ja mitä se mahdollistaa hyökkääjän näkökulmasta. Myös ohjelmistopohjaisen attestaatiojärjestelmän mahdollinen alustariippumattomuus kuuluu ohjelmistopohjaisen attestoinnin hyviin puoliin.

PREFACE

This thesis was written while I was a research assistant at the Tampere University of Technology. The research was conducted in the Internet of Things program of DIGILE (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT), funded by Tekes. The topic of this thesis was suggested by NIXU Oy.

Not counting some of the more general matters, the subject of this thesis was not familiar to me at all and thus required a great study of the related work and techniques. As someone with background in the University of Applied Sciences, writing a long scientific document required a lot of getting used to as well. With all its ups and down, the whole process has still been amazing and very educational. This experience most certainly has taught a lot new things for the future endeavors.

I would like to thank my supervisors Prof. Jarmo Harju and M.Sc. Joona Kannisto for their patience and invaluable feedback during the thesis process as well as NIXU Oy for the interesting topic to base the thesis on. I would also like to extend my thanks to all the amazing co-workers, past and present, in our department for the wonderful atmosphere and friendship. Special thanks go to my sweet friends for All the Remarkable support given during this project. Last but certainly not the least, I would like to thank my parents for all the support they have provided during all these years.

I guess I'm through, huh?

Valkeakoski, 25.5.2016

Antti Kolehmainen

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

ACL	Access Control List
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AIK	Attestation Identity Key
CA	Certificate Authority
CPRNG	Cryptographic Pseudo-Random Number Generator
DES	Data Encryption Standard
DDoS	Distributed Denial of Service
DLP	Discreet Logarithm Problem
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algoritm
GPOS	General Purpose Operating System
HMAC	Keyed-Hashing for Message Authentication Code
IoT	Internet of Things
ISLC	Information Security Life Cycle
IVP	Integrity Verification Procedure
LPS	Linear step, Permutation and Substitution
MITM	Man-In-The-Middle
NESF	NIXU Embedded Security Framework
NIST	National Institute of Standards
OS	Operating System
OTA	Over-The-Air
PARC	Palo Alto Research Center
PKC	Public-Key Cryptography
RTM	Root of Trust for Measurement
RTR	Root of Trust for Reporting
RTS	Root of Trust for Storage
RTOS	Real Time Operating System
SBA	Software Based Attestation
SBC	Single Board Computer
SDLC	System Development Life Cycle
SHA	Secure Hash Algorithm
SBox	Substitution Box
TBB	Trusted Building Blocks
TCG	Trusted Computing Group
TOPPERS	Toyohashi OPen Platform for Embedded Real-time Systems

TP	Transformation Procedure
TPM	Trusted Platform Module
TT	Transitive Trust
TZ	TrustZone
VCOS	VideoCore Operating System
WSN	Wireless Sensor Network

1. INTRODUCTION

The idea behind Internet of Things (IoT) was conceived in the early 90's when an engineer Mark Weiser at Xerox Palo Alto Research Center (PARC) described a concept of 'embodied virtuality' in which the computing will be advanced into ubiquitous phase [1]. This means that the computer will be hidden from the view and be turned into a such common thing that its existence will be taken as granted. Weiser explains this phenomenon as 'when people learn something sufficiently well, they cease to be aware of it', for example street signs that contain information that is absorbed consciously and not read per se. Devices participating in this community can consist of everything from computerized black boards to light switches.

Since these IoT devices are to be ubiquitous, there can be serious consequences if they can be compromised by a malicious user. For example, a sensor in a car tire could be set to feed incorrect data to the control unit, which could, in the worst case scenario, cause a blow-out in the middle of a highway. It is crucial that the devices communicating between each other can trust each other to be what they are supposed to be.

This thesis investigates the viability of using pure software based remote attestation in the context of IoT networks. Pure software is used to mean that the device does not require any extra hardware for any part of the protocol but can handle it using standard platform features. Attestation is the process where a device integrity is measured for detecting modification of software or hardware from the original configuration. Remote attestation is an attestation process over the network where an external server requests integrity information from a node and by comparing it to previously known one decides if it is valid or not.

Figure 1.1 depicts the concept of attestation on a high level. The robot is requesting a permission to go for a swim, but the swimming pool is marked for humans only. The security guard, or the attestation service, is requesting the robot to present proof of being a real human which the robot is doing by having a name tag and arguing that it surely must be a human. As the guard knows which features a real human must have he can trivially detect that the robot does not fit them and deny access.

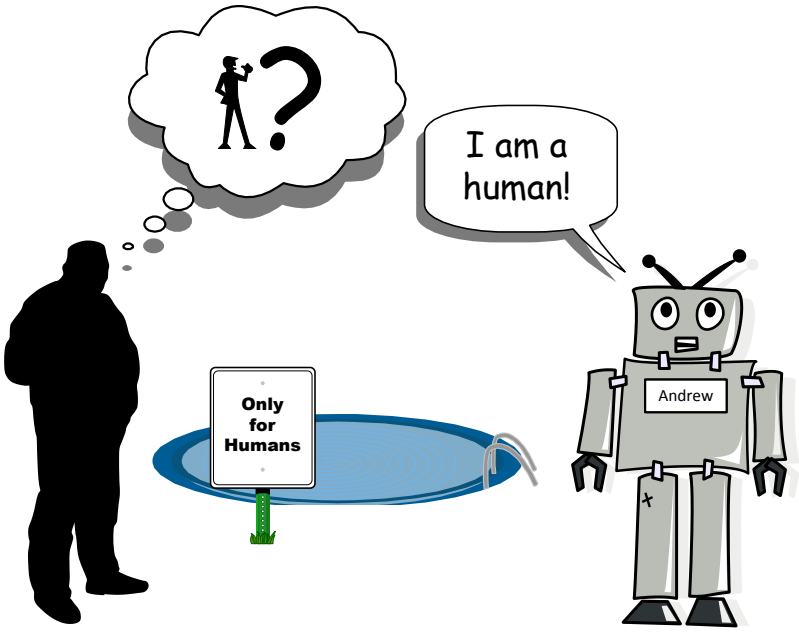

Figure 1.1 *High-level Attestation*

The security of an IoT device is a juggle between cost-effectiveness and the implementation of the security mechanisms. For example, one consideration is whether to include any special security hardware or to create their services with software. For an IoT device that should be cheap and widely deployable, a purely software-based solution is definitely a desirable option. For securing the trust between the devices, a mechanism called attestation is used. Implementing this mechanism in software is one of the problems to be solved considering IoT deployment.

Software Based Attestation (SBA) can be thought to have three primary properties that separate it from hardware-based designs:

Scalability SBA scales from one device to a network of several devices no matter what hardware is running on them. The same attestation process can be used on a tiny sensor node as well as on a laptop computer or even a server as the items queried for integrity data are universally the same, for example, memory and hardware configuration. The attestation server itself can handle multiple nodes within a network which can be scaled even more up by adding more attestation servers to it.

Portability The possibility for SBA to be deployed trivially on a wide variety of operating system/hardware combinations especially in IoT networks where one group of devices might include a variety of devices such as sensors, controllers or even

servomotors is an advantage. Another property of portability is the portability of the code. Using interpreted language, such as Python, makes the development easy as the code does not need to be customized to each different hardware platform and can in many cases run as is on them.

Cost-effectiveness SBA does not require any extra chips and hence it can be utilized on off-the-shelf hardware. Design and development of custom hardware might become expensive while general mass-produced hardware can keep the costs down. Using well-known development platforms also enables access to possible community support whereas custom hardware might require expensive support contracts or internal training.

The negative aspect of pure software based attestion compared to hardware-assisted one is the lack of services provided by hardware security elements. For example, one of these is the secure boot provided by the TPM. While there exists software implementations of the TPM standard [2][3], they are just like any standard software program and thus require boot into a running operating system before they can be used. The protection of such software TPM also is questionable as so far they do not offer isolation from the running system requiring the use of manual sandboxing technologies for protection.

From initial deployment and provisioning to the device end-of-life, the devices' life cycle has to be managed. Especially in the case of IoT devices deployed in mission critical roles, it's important to consider the security issues in each of the stages of the device life cycle.

As the IoT device deployment is to be ubiquitous, they require connectivity over wired or wireless networks. The remote attestation model separates attestation clients from the servers that initiate the protocol used to prove the legibility of the clients. These servers can as well be used to supply externally generated assets such as cryptographically stronger random numbers to the clients to use. Both client and server platforms have to be secured against malicious use. In case of attestation this generally requires collecting data from the system and using it to build a 'last good configuration' image that can be compared by the server to samples taken after deployment. However, in the IoT use case not only is internal security important but also external security has to be considered. Malicous user may steal the whole device and get physical access that enables them to research the hardware in order to replicate or emulate it. Therefore, it is important to try to design the attestation method to resist this type of attack.

Attestation builds upon well known cryptographic methods such as hash functions and key systems to attain the trust between the parties involved in the protocol [4]. The aforementioned methods might require unpredictable random number generation that might not be

as easy on the simple IoT devices as it is on more powerful or on dedicated hardware. The unpredictability of random numbers is essential in security of most cryptosystems. The problem of unpredictability on simple devices is one of the problems to be considered during the design of IoT systems.

The structure of the thesis has been set as follows. Chapter 2 starts by presenting the problem of security in embedded devices in the IoT context considering their deployment environment and possible attack vectors. Chapter 3 goes more into the building blocks of attestation protocols, such as cryptography and random numbers, and what problems are encountered on embedded low-cost devices. The next chapters will discuss the attestation itself, presenting matters related to existing attestation methods and how viable they are in the IoT context.

Chapter 4 discusses the concept of attestation with emphasis on remote protocols. What most higher-end devices, such as smart phones and personal computers, have for attestation is the TPM. Chapter 5 discusses the viability of using hardware assistance during the attestation protocol. This hardware assisted attestation can be either full attestation using extra TPM chip or the use of standard platform features such as secure processor registers.

Next chapters will present the details and discussion about the viability of pure software attestation. Chapter 6 introduces the platform that was selected to be investigated as a candidate for the attestation node with remarks on probable operating systems and their standard features that can be utilized when building attestation processes. In no part will any extra hardware be added to the system and as such the attestation must be able to run with standard configuration of the chosen hardware and software. Finally Chapter 8 discusses the presented matters and the thesis ends with closing remarks in Chapter 9.

2. CHALLENGES AND REQUIREMENTS

Embedded devices face several challenges regarding security. Wireless Sensor Network (WSN) and IoT deployments, where the activities are desired to mostly be autonomous and location agnostic, will be exposed to internal and external threats. Internal threats involve software-based attacks inside the device itself. External attacks are those coming from outside the device, such as physical tampering or network related attacks. Security properties a WSN should require are data confidentiality, data authentication, data integrity and data freshness [5].

Starting from provisioning the software to the device, going through updates and finally decommissioning it, another type of a challenge is the management of the devices during their life cycle. There are several life cycle models applicable even for IoT devices. The ones considered here are the System Development Life Cycle with the Information Security Life Cycle.

2.1 Security and Attack Models

Security can be modeled using a triad consisting of confidentiality, integrity and availability (CIA). These three properties can be applied to attestation as follows:

- Confidentiality means that only the ones participating in the information exchange can understand the contents of the data in question [6]. An eavesdropper can not copy or steal the data [7] and as such it needs encryption. For example, this data on an IoT device could mean the cryptographic keys and passwords stored within [7].
- Integrity assures that the content of the information exchange data can not be altered in any point of its delivery either maliciously or accidentally [7][6]. For example, this data can be the remote attestation protocol data between the client and the server or some data provided by the services running on the IoT device.
- Availability means that the data resources are available when requested by users authorized to access them [8].

- This list can also be augmented with more properties such as authentication that is used to confirm the identity of a party, especially in the context communication networks, and accountability or nonrepudiation that is used to confirm that the party can not deny their actions [9].

Attacks can be furthermore categorized according to the vectors they use. The vectors can be either software or hardware based such as viruses and malware targeting the operating system services or physical attacks that require external components or hardware for accessing the device [7]. The most intrusive physical attack is the one that uses techniques such as component reverse engineering, cryptographic analysis, signal analysis and similar [7]. Software-based attacks can also be harder to detect as the devices do not require external access such is the case with total reverse engineering or installation of attack components to the device itself. Another factor among the malicious software is the person operating it, the attackers themselves.

Another type of attack targeting Wireless Sensor Networks (WSN), that the IoT networks essentially are as well, is the power supply targeting and battery draining Denial of Sleep (DoS) [10][8]. It is also very probable for the attacker to be eavesdropping the communication in the IoT network. Because of this, the attestation system should resist replay attacks.

One of the more interesting physical attacks against small devices is the clone attack. This attack requires the attacker to physically clone the device in question either in hardware or software using emulators. If the attacker is successful there is no way in knowing if the device being attested is of malicious nature. This attack can also be performed as man-in-the-middle (MITM) where a modified device forwards valid attestation requests to unmodified device and then uses the checksum received to authenticate itself [11]. Shaneck et al. argue that this kind of impersonation attack can only be detected by using tamper-proof hardware [11] therefore being unable to be countered with pure software-based attestation processes.

For the purpose of this thesis, software-related attacks are considered to be the primary vectors. Denial of Service (DoS) attacks can be a nuisance on low cost low powered devices, especially if they communicate over wireless channels [8], as well as exploits on software bugs especially if the devices are running standard off-the-shelf software.

2.2 Deployment Scenarios

IoT is the next step up from WSN and as such is used in a very similar fashion. Typical applications for WSNs include energy management, emergency response information [5],

medical monitoring [8][5], logistics and inventory management, industrial applications and military applications [8][5].

One example of a modern IoT deployment is the industrial environment. While factories have been running under automation for decades it is these recent years when they are starting to "get smart". This integration of IoT technologies has been dubbed the fourth industrial revolution and has seen adoption around the world. For example, an initiative called "Industrie 4.0" was enabled in Germany [12] that is aiming to drive the integration of IoT to factories nationwide.

A smart factory would consist of standard manufacturing equipment augmented with IoT technologies that enable the factory environment to reconfigure itself according to specifications and provide a huge deal of data for processing in cloud based information systems [13]. Not only controlling devices would communicate with each other and the cloud, but also various smart sensors would provide data about conditions in the factory itself and inside the production line. Wang et al. [13] describe a framework for Industrie 4.0 (Figure 2.1) that separates the different parts of the factory into layers.

- Physical layer consists of resources that communicate with the cloud via the industrial network where the transmission medium can be wireless or wired. Wang et. al. argue that wireless networking is mandatory for smart factory operation due its superiority in provisioning of new machines to the network. However, in factory automation wireless latencies may become an issue on critical machine control communications where specialized wired protocols still are relevant. Modern industrial buses such as the EtherCAT [14] work using standard Ethernet based networks and so can be integrated to the smart factory quite trivially.
- Cloud layer in the Industrie 4.0 framework consists of the "brains" for the factories. All the data collected from various sensors and devices in the physical section will be processed and presented to the operators from the command and control layer. As the idea behind IoT is the global networking of "things", also the cloud layer will be connected to the Internet and via that to other systems that can interoperate with the factory systems.

Taking this framework and adapting it to a small-scale industrial example, we can think about a modern IoT enabled brewery. The physical layer consists of all the brewery machinery, their sensors and control surfaces. These sensors oversee values such as the temperatures within the process as well as in the brewery rooms, pH values for the liquids, states of latches, actuators, motors and other machine related values as well general

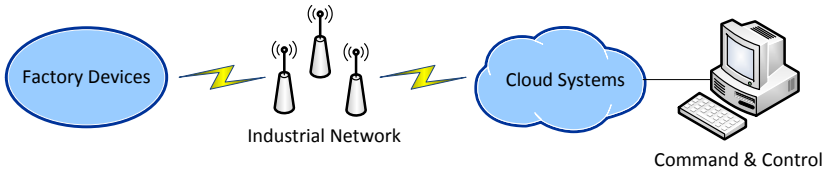

Figure 2.1 *Industrie 4.0 Framework (Adapted from [13])*

matters such as lighting, air quality, quality of electricity and so on. The data from sensors and machinery is then transmitted to the cloud layer for processing.

The cloud layer processes the data and presents information to the operators as well as back to physical layer devices. The brewing process is able to adapt to different environmental conditions due devices talking to each other and sharing information of everything. The cloud also will include data about customer orders and warehouse stocks which allow the brewery to be efficient and manufacture as much products as is at any time needed.

Another use case for IoT is the smart home. Sensors would be deployed to monitor home environment, such as temperature, electricity and lighting as was the case in the industrial case as well. These sensors would be extended into domestic devices such as kitchen appliances and washing machines. As an example, we look at a smart IoT enabled smart washing machine.

Figure 2.2 depicts a simple outline of an IoT enabled washing machine and what data will be processed within it. First, the machine is connected to the Internet or as minimum to a closed homenet where it is able to access all data from the home's IoT devices. Sensors feed data about water quality, such as hardness or temperature and whether the water lines are already filled and when they were filled, as in how long the water has been still. Water temperature informs whether the machine needs to activate warmers or coolers and sensors measuring the water hardness will control the amount of washing liquid required. Finally the clothes themselves relay information about the correct washing program and settings using technologies such as RFID tags. The machine also knows the amount of available products by fetching the information from the homenet global inventory database.

The sensors, buttons, actuators and similar within the machine are exposed individually for applications to use and build on [15]. This allows the integration of the washing machine system to any third party application, for example something that controls the individual components or just a smart phone application that presents data to the user. Another similar vision is of a 'smart' kitchen where refrigerators can discuss with pantries

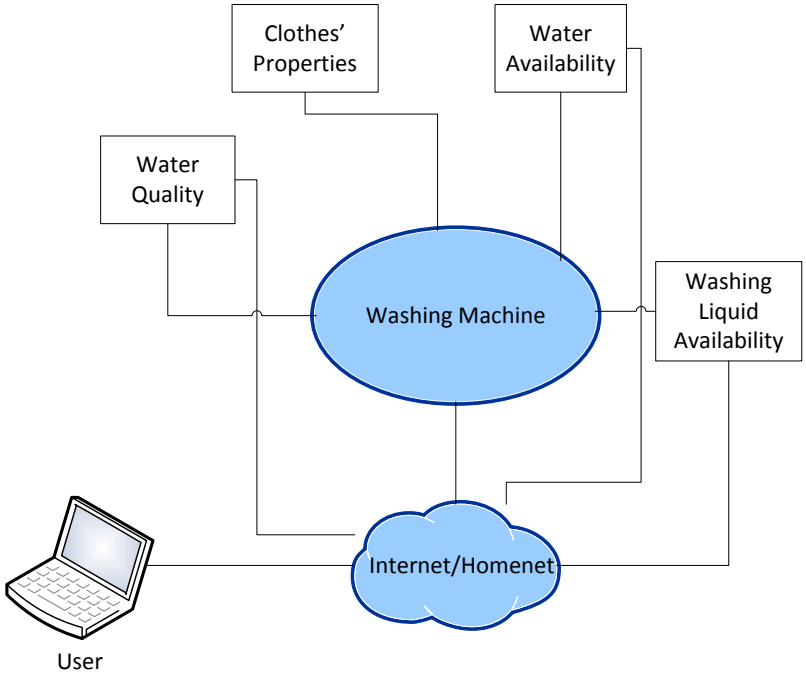

Figure 2.2 *Smart Washing Machine*

and sensors around the house to remind the owner about products expiring and items required for scheduled meals.

2.3 Life Cycle Management

An important part of security is the management of the IoT device's life cycle. From the initial provisioning to the disposal of the device, it's essential that their updates and replacements can be controlled with care. There needs to be lot of automation in handling the internal software life cycle, since numerous devices can't be trivially updated manually. When a device is taken out of commission, it's important to have a protocol that controls what happens when the device is removed from the network or even the whole hardware inventory. This protocol has to be designed to resist malicious users trying to access the device identification data.

Figure 2.3 depicts a five stage model for a System Development Life Cycle (SDLC). These stages work in the same way as in the well known Stage-Gate model used for new product development. Each of the five stages consist of several activities that are run through and gates that will control whether the process is ready to move to the next stage [16]. Taking security along in each of the stages is vital for it to be effective.

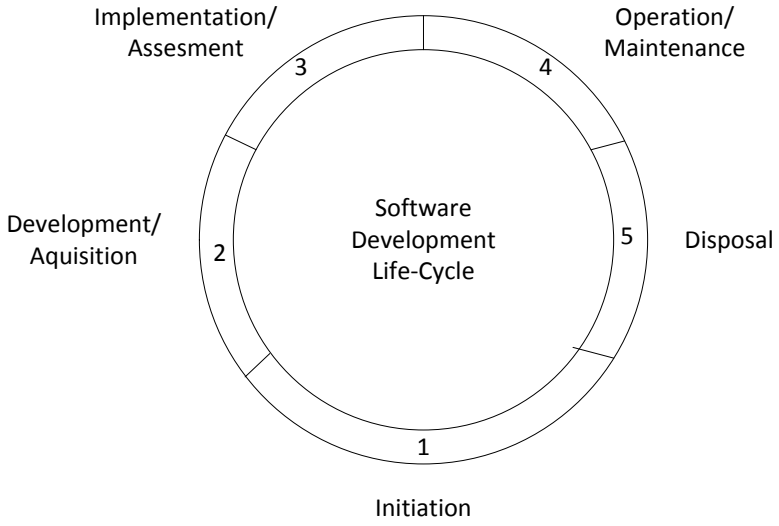

Figure 2.3 *System Development Life Cycle (Adapted from [16])*

The life of a system device starts with setting the goals and requirements for the actual design to be made. With IoT case it is essential to consider the exact job the device in question will be deployed in as in IoT the devices are usually created for a specific purpose and not multitasking, so to speak. From security perspective, the first stage will set the requirements in terms of the CIA triad regarding information handling and privacy [16].

The next stage will see the beginning of the development, or within the scope of this thesis the acquisition of the platform device to be used as the IoT node. This second stage security-wise will include risk assessments, security testing and the design of the security architecture as well as more analyzes for the security architecture [16]. When the device has gone through the development phase, or in the case of this thesis the installation of the attestation software, it will be deployed into use.

The third and fourth stages of the system development life cycle consist of installing and running the device on its operational environment [16]. During this operational phase the device should be continuously assessed for security, efficiency and if required, updated or replaced. These activities also mirror the Information Security Life Cycle (ISLC) (Figure 2.4) which is as well an on-going process. ISLC starts with assessment of all the risks regarding the device operation. After assessing the risks, a cost/benefit analysis has to be made in order to find out which risks are worth mitigating [17] and cost-effective to implement fixes for. After risk mitigation processes have been implemented, it's important to conduct an assessment to see if the implemented measures work as planned [17]. After that the whole ISLC process starts from the beginning running until the device is deemed

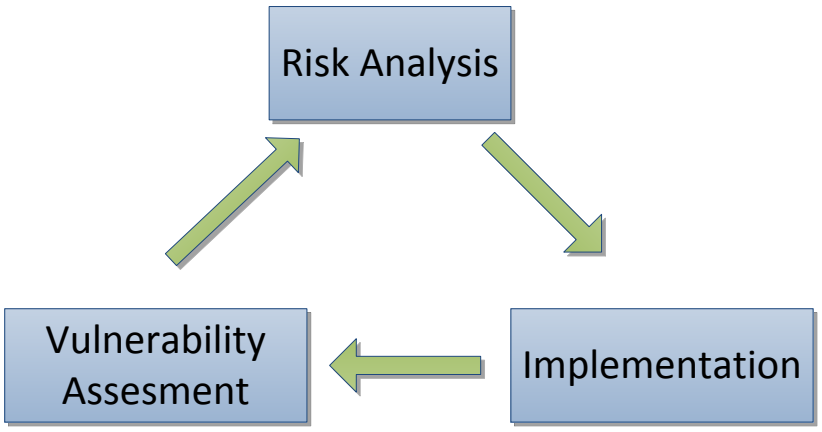

Figure 2.4 *Information Security Life Cycle (Adapted from [17])*

to be decommissioned.

The final stage of the SDLC is the disposal of the device. It is essential to make sure that any secure information from the device is not leaked but archived if required, sanitized and securely destroyed [16].

2.4 Vulnerability Models

As with WSNs, devices of the IoT get exposed to similar vulnerabilities. By design, these devices are limited in processing power [5], memory and as such, features, to keep the cost at minimum [8].

For the devices in the IoT network there are several issues to be taken into consideration. For example, a malicious user could get access to the device and replace it with an identical copy running the attackers' own software or hardware. This scenario can be avoided by attaining uncloneability for the device. If the device is cloned and deployed back to the network, there has to be a way for the network to detect and isolate the cloned device so it can not cause harm. With this comes the issue of keeping the integrity of the device even if the attacker could access the device either physically or remotely. The device has to resist attempts to influence the software it's running.

The attacker might also listen to the communication within the network, either over wireless or wired media. The devices should be able to communicate using an insecure channel and exchange integrity information between the controller and the nodes without the attacker being able to forge the messages. This also should prevent the insertion of a

malicious node to the network and it being able to participate in the communications. By attesting all the devices in the network, we are able to ensure that their integrity has not been compromised and that the devices are what they claim to be. Therefore, building a reliable attestation process is essential for networked IoT devices.

In the case of remote attestation, the attacker might try to influence the operation of the attestation server instead of the nodes. Furthermore, as the attestation server in many cases will have knowledge of the hardware configuration for the nodes as well as cryptographic keys and hashes, it becomes a desired target for the attacker. Therefore the attestation server itself must be secured even stronger than the nodes. Fortunately, as the server does not necessarily need to be an embedded system, it can leverage much more secure platforms such as hardware TPMs and firewalls.

By recreating the TPM-style services in software, we are able to build a system that is able to be deployed to virtually any system available without any more hardware than a processor and some storage running some operating system that is able to run code. One huge consideration is the implementation of secure boot to establish the initial chain of trust for the system in question. As we later see, this might not be as trivial as it would be using a hardware TPM solution. For example, boot in Raspberry Pi's default stage uses unsecured storage on a standard flash memory card containing all the boot firmware images [18].

3. EMBEDDED CRYPTOGRAPHY

Cryptography in embedded systems face some challenges that are not as relevant outside of it. The source of random numbers for cryptographic operations is a problem on resource constrained devices as is the available processing power for running the cryptographic operations themselves. However, there are mechanisms for delivering and generating random numbers and the amount of fast cryptofunctions are growing all the time.

3.1 The Tale of True Random

When one thinks of the word 'random', it's easily understood as something that can't be predicted and appears different every time. In nature, one example of this could be the randomness collected from thermal noise or radio active decay [19]. Computers, however, generate random numbers using Pseudo-Random Number Generators (PRNG).

The property of unpredictability for the random numbers requires a careful collection of randomness from multiple sources. This collected randomness can be mixed together in an entropy pool [20] that is furthermore used for number generation. Sources used for collection are usually timings of key presses or network activity [20][21]. In the case of a small IoT device, the collection of randomness could be achieved by using external sensors measuring thermal noise [19] or by an external entropy broker [22] that delivers the randomness data over the network. The issue with using an external entropy broker is in the communication channel. If the attacker is able to catch the entropy data being sent to the nodes, it might be possible to reset a device to initial values trivially. This attack can be mitigated by securing the communication channel and using devices that do not only rely on one source of randomness.

Random numbers are heavily utilized in the realm of cryptography. They are present in almost every single operation from simple nonces used to initialize values to constant generation of new numbers such as with the Kerberos protocol [23]. Random numbers are needed, as previously discussed, for seeding the random number generators that are in turn used to generate numbers for example, for cryptographic keys and signatures [23]. For all this to work as intended, the entropy value of the generated random numbers has to be high.

For use in cryptographic operations the PRNG needs to be a bit more specialised. These specialised generators are called cryptographic pseudo-random number generators (CPRNG). CPRNGs have to be secure against standard cryptographic attacks. They need to be unpredictable to an adversary who knows the algorithm used to generate the numbers and they should not give any usable information to an adversary even if the adversary can see a lot of the generated data [21].

3.2 Cryptographic Hash Functions

One major element used in cryptography is the use of hash functions. They are used in authentication, message authenticity, signatures and also as a part of PRNGs [24][25]. Cryptographic hash functions have properties such as being one-way and collision resistant [26] [27] [24]. The one-way property means that the functions can be computed with little effort but inverted only with great difficulty [25]. Collision resistance simply means that two different messages should not have the same hash value [26] and the fastest way to find a collision is by the birthday attack [28].

Secure Hash Algorithm 2 (SHA-2) is a family of hash functions with hash values, or message digests, of 224, 256, 384 and 512 bits [29]. It was designed by the National Security Agency (NSA) as the companion hash function for the AES encryption standard. SHA-2 is also used as one of the hash algorithms in the Keyed-Hashing for Message Authentication Code (HMAC).

Secure Hash Algoritm 3 (SHA-3) is the result of a competition arranged by the National Institute of Standards (NIST) for complementing the SHA-2 family of hash functions [30]. The Keccak primitive was selected as the implemented SHA-3 standard. The SHA-3 hash function uses a hash construction called "sponge". A cryptographic sponge construction works in two phases, the absorption and squeezing [31]. Just like water poured on an ordinary sponge, the absorption phase will take in a message and/or key after which some function is applied to it. After the sponge function has been applied, the output will be squeezed out of the sponge [31]. The output will be the desired length bitstream.

BLAKE2 hash function has been designed to be faster than the widely used yet weak MD5 and to be as secure as the SHA-3 Keccak function [32]. The hashing speed of BLAKE2 is especially important in embedded environments considering it is a fully software based function and as such it is portable for multiple system architectures trivially. BLAKE2 also comes as a standard with built-in support for salt for augmenting digital signature schemes [32].

3.3 Public Key Cryptography

Public Key Cryptography (PKC), or Asymmetric Cryptography, works with a pair of keys. One of these keys is the private key and the other is the public key (Figure 3.1). Public keys are used to encrypt messages intended to the owner of the used public key. The message is decrypted by using the private key of the message receiver. The most common use for public key cryptography is in digital signatures where authenticity of messages and peers is essential.

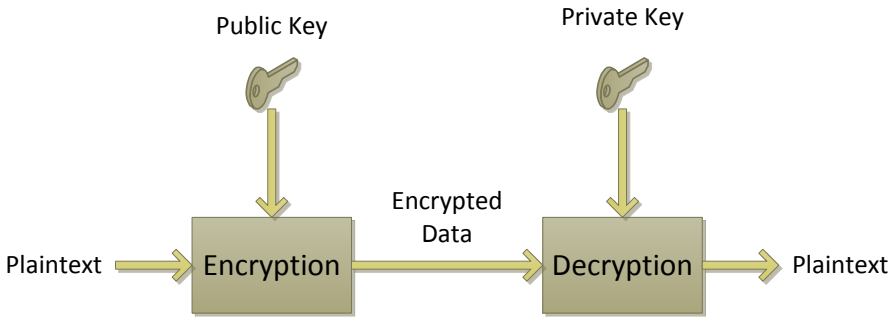

Figure 3.1 *Public Key Cryptography (Adapted from [6])*

Even though digital signatures have been compared to standard handwritten ones [33] they have some peculiarities that make them much more complex. The addition of a "third party", in this case the computer, used to create and use the signatures, creates issues with trust [34]. The internal workings of a computer used to create signatures have to be implicitly trusted to be sure that a malicious third party can not use the user's signatures for its own purposes.

The RSA Cryptosystem is based on modular arithmetic with prime numbers. The two parties create their private and public keys by selecting two large prime numbers p and q and computing the results of $n = pq$ and $\phi(n) = (p-1)(q-1)$. Next a random number e less than $\phi(n)$ and relatively prime with $\phi(n)$ is chosen. The next step is to select an integer d such that $d * e$ differs from 1 by a multiple of $\phi(n)$ [35]. The public key consists of the number pair (n, e) and the private key of the pair (n, d) [6][35].

The security of RSA is based on the one-way nature of the encryption function [36], or simply that there are no known algorithms that are fast enough for factoring numbers [6]. The property that allows decryption from the encryption is called a trapdoor. The trapdoor in RSA's case is the knowledge of the factorization of the prime product used in key generation [36].

3.4 Symmetric Cryptography

Unlike asymmetric, or public key crypto systems, a symmetric crypto system only has one key used for both encryption and decryption (Figure 3.2), thus forcing both parties to share the key to be able to communicate securely. The advantage of using symmetric encryption is that it is much faster to implement in software or hardware than asymmetric encryption and hence is used widely in applications where large amounts of data have to be encrypted efficiently [33]. Symmetric cryptosystems are separated into stream and block ciphers. Stream ciphers encrypt a stream of data bit by bit while block ciphers operate on blocks of data. The following describes some recently developed symmetric ciphers.

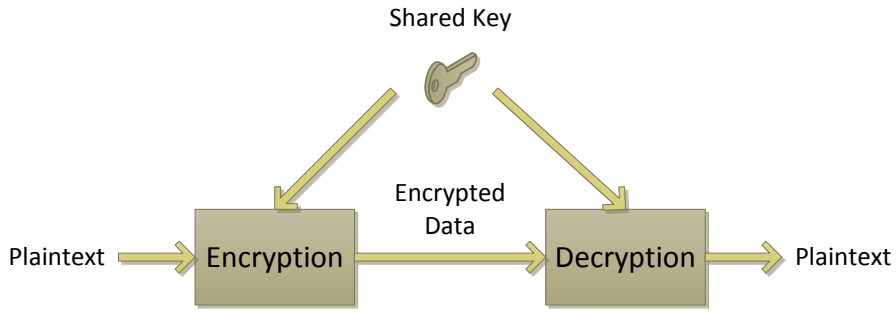

Figure 3.2 *Symmetric Cryptography (Adapted from [6])*

Salsa20 is a stream cipher designed to be fast in comparison to the widely used AES encryption [37]. It uses simple addition, exclusive or (XOR) and rotation operations on 32-bit words in a chain. One curiosity in Salsa20 is its non-use of Substitution Box (SBox) constructs like in AES. Salsa20's use of integer operations makes the cipher more secure against certain types of attacks and faster on wider range of hardware [38][37].

STRIBOB is an SBox based algorithm for Authenticated Encryption with Associated Data (AEAD) and is designed around of a Linear step, Permutation and Substitution (LPS) transform [39]. Twelve iterations of the LPS transform will be permutated with XOR inside the main cryptofunction. STRIBOB draws inspiration from the Russian GOST R34.11-2012 hash standard that also uses the LPS transform with SBoxes, but uses it differently while augmenting the algorithm with a sponge construct. [39]

WHIRLBOB is an AEAD algorithm that includes the sponge construct and parameters from STRIBOB but switch the used LPS construct for the one from the Whirlpool [40]

hash function. The only difference is the used twelve rounds instead of the Whirlpool's ten [41].

Symmetric cryptography in the context of IoT and attestation can be utilized for example with mechanisms such as cryptographic containers and encrypted file systems. These mechanisms are used to hide a part of the operating system behind a lock and a key, so to speak. If the device requires access to the encrypted areas, such as sensor accessing configuration data, it needs to run the attestation protocol first to prove itself to the network before receiving the key for the cryptographic container or file system. The key delivery and in-device key management methods are some of the important considerations in designing attestation systems as it has to be secured against malicious eavesdropping.

The requirement for secure key management implementations compared to PKC make symmetric cryptosystems more complex to set up, especially in a wide deployment as the IoT is. A common way to manage keys in symmetric cryptosystems is by using a mechanism called key-wrapping in which symmetric keys are used for encrypting other keys [42]. Key-wrapping might often be based on a tree-structure with one master key encrypting lower keys and so forth [43]. The problem in this implementation besides the complexity of keys within keys is the matter of compromised keys able to decrypt everything below their level. This is not an issue in PKC as every node would have their own key pair.

3.5 Elliptic Curve Cryptography

While the previously mentioned PKCs are based on mathematical structures such as multiplicative groups $(Z/NZ)*$ and $GF(q), q = p^n$, there also exists PKCs that are based on elliptic curves [44]. An elliptic curve is the set of solutions $(x, y) \in K^2$ to the equation $y^2 = x^3 + ax + b$ where $a, b \in K$ and where K is the field of characteristic $\neq 2, 3$ [44]. Figure 3.3 shows an elliptic curve $y^2 = x^3 - x + 1$ that has been plotted for all numbers. As with the previous cryptosystems, also with elliptic curves used for cryptography the range of numbers should be fixed with a prime number as the maximum value [45]. Elliptic Curve Cryptography (ECC) has many advantages compared to the previously used structures that make it very attractive especially for IoT device use.

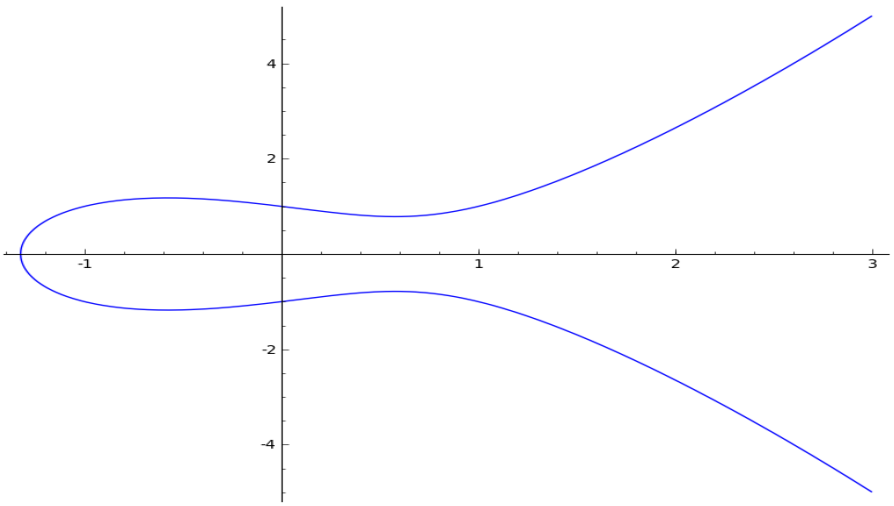

Figure 3.3 *Elliptic Curve $y^2 = x^3 - x + 1$ plotted for all numbers (Source [45])*

ECC primitives use scalar multiplication that consists of repeated point addition and doubling as the main operation [46]. Figure 3.4 shows and example of how points are selected based on these operations using the curve presented in Figure 3.3 as the base. The initial selected points are the points A and B through which a line will be drawn. The line will intersect the curve in point C.

From this we are able to get the sum of points A and B on the curve which is the point D [47]. To continue point selection, a line is drawn through A and D and as before, at the new intersection a vertical line will be drawn whose intersection will be the new point. For deriving the primitives from these point operations, we get the private key by selecting a number that is prime and large enough but less than the selected maximum and the public key that is a selected public point, e.g. A in Figure 3.4 that has been ran through the scalar multiplication operation as many times as is the value of the private key [45].

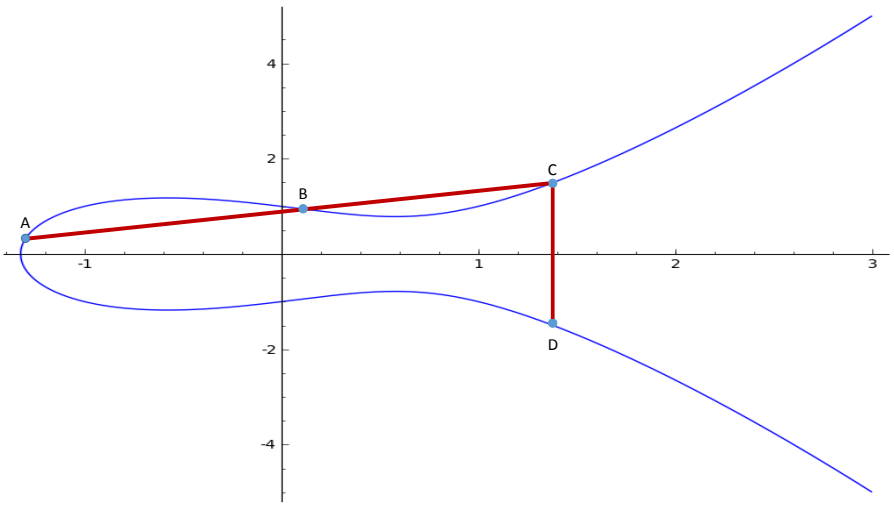

Figure 3.4 *Elliptic Curve Operations (Adapted from [45])*

The Elliptic Curve Digital Signature Algoritm (ECDSA) is a version of the Digital Signature Algorithm (DSA). As with DSA security being based on the Discrete Logarithm Problem (DLP), the ECDSA is based on the Elliptic Curve Discrete Logarithm Problem (ECDLP) [48]. Elliptic curve based cryptography offers benefits such as smaller key sizes and resource saving functions [49]. These benefits are enough to justify the use of ECC on resource constrained systems [50][49] such as the devices of IoT. Security-wise, the effort in solving the ECDLP is far greater than with just DLP which enables the use of smaller parameters for increased security [48].

The matter of public key cryptography and the required resources on small embedded devices is one of the questions when deciding which ciphers to use amongst IoT devices. ECC certainly looks to be a good choice as it can work fairly fast even on small devices and can scale to more powerful easily. However, depending on the operating requirements of the device itself, even non-ECC PKC can be used. For instance, Sethi et. al [51] argue that for a sensor that is not active constantly, the time used calculating public key ciphers does not interfere with the use scenario [51].

4. ATTESTATION

Going back to a similar scenario as depicted in the introduction chapter, let us imagine a conversation between a night-club bouncer and a patron trying to get in. To get in, this particular night club requires the bouncers to present a question, a puzzle if you will, to the customers wanting access to the club. If the customer can answer the question correctly, the bouncer knows they are what they claim to be and can allow them access. Otherwise access will be denied.

Attestation protocols can be modeled using the same metaphor as well. Our devices are the patrons queuing outside of the night club and our attestation server is the bouncer checking their validity. The bouncer will assign a task to each of the devices to prove their good intentions. The mechanisms used for this proving can be implemented in hardware, software or as a hybrid system. Especially considering IoT networks, this attestation service has to be provided remotely.

4.1 Integrity Measurement

The Trusted Computing Group (TCG) defines trust as the expectation that a device will behave in a particular manner for a specific purpose [52]. This trust can be assured with the process of attestation. However, before attestation can be successfully performed the trustworthiness of objects regarding their integrity, configuration and policies must be measured [53].

One model for integrity measurements was developed by Clark and Wilson [54]. In the Clark and Wilson model, the integrity policy is defined with two procedures, the Integrity Verification Procedure (IVP) and the Transformation Procedure (TP). The IVP is used to verify that the data is according to the integrity specification and TP is used to change the state of the data.

The model of integrity assurance according to Clark and Wilson is a two part process consisting of certification and enforcement. The three rules for assurance according to them are:

- Ensured validity of data during verification procedure.
- Transformation procedure validity where the procedure has to have a relation to the data it is allowed to manipulate.
- Maintaining of lists of relations and ensure of data manipulation by procedures that are valid for it.

This three rule basic framework can also be augmented with more rules to expand the scope for even more fine grained integrity validation. Building upon Clark and Wilson's model, Sailer et al. defined several tasks that will help attaining the Clark and Wilson integrity level verification [55]:

- Verification Scope that defines which processes must be measured for integrity information.
- Executable Content that enforces executable code to be of sufficient integrity.
- Structured Data that defines that only data handled by the OS processes will be measured and treated with the same manner as the executable files if the data is marked to have identifiable integrity semantics.
- Unstructured Data for data without identifiable integrity semantics that is dependent on the processes that access it where the data integrity depends on the integrity of the process handling it.

When these measurements are taken from the initial state, they can be later on compared to the more recent measurement data taken along the device life cycle. However, integrity measurement is not the only process in trusted systems. It is part of a bigger concept known as the trusted platform.

4.2 Trusted Platform

Trusted platform is the term the TCG uses for a platform that provides as minimum three features, protected capabilities, integrity measurement and integrity reporting [52]. Protected capabilities are a set of commands that are able to access protected locations, such as RAM or registers, where sensitive data can be operated safely.

The trusted platform builds on components called Roots of Trust that commonly exists as Root of Trust for Measurement (RTM), Root of Trust for Storage (RTS) and Root of Trust for Reporting (RTR). The functions for each of these are as follows:

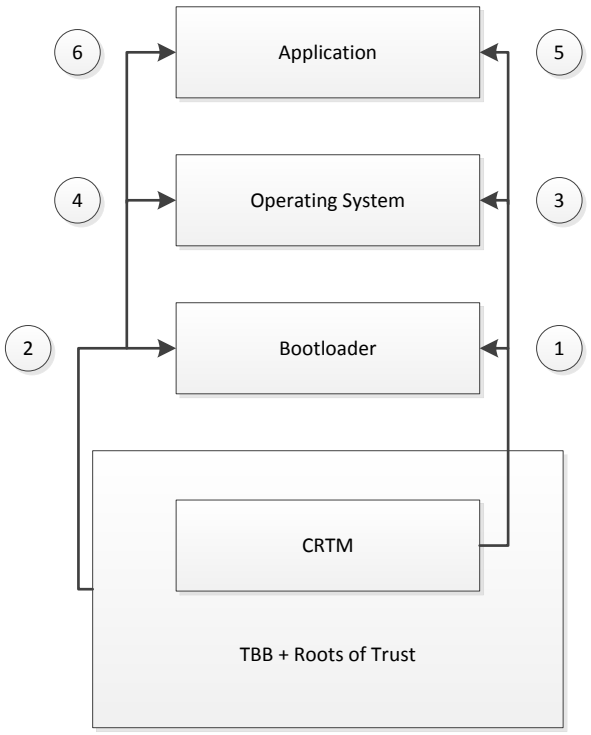

Figure 4.1 *Transitive Trust (Adapted from [7])*

- RTM is the primary engine for integrity measurements.
- RTS maintains integrity digest and digest sequence value summaries.
- RTR handles all the reporting of the information stored with RTS.

The primary assumption is that for the trusted platform to be trusted, the trust for each of the Roots of Trust has to be absolute thus requiring mechanisms to be deployed to secure them accordingly.

An important feature of the trusted platform is that it enables trust to be established from "bottom-up". A process called Transitive Trust (TT) [7] measures trust of each of the different parts of the system and only goes forward when the previous integrity has been verified. Figure 4.1 shows transitive trust from system boot onwards.

In Figure 4.1, Trusted Building Blocks (TBB) are parts of the Roots of Trust that do not have protected capabilities such as CPU instructions. The Core Root of Trust of Measurement (CRTM) instructions are executed when the platform acts as the RTM [7]. When the system is turned on, first set of measurements are taken and verified. If the

verification passes, the measurement process moves to the second part, the bootloader and verifies its integrity before execution control to that part is transferred. The process continues in stages until the whole system up to the application level has been verified.

4.3 Remote Attestation

Figure 4.2 shows a model of remote attestation from the attestation server's point of view. First the server queries a node of its identification data. This data can be hash of the device's internal configuration, software or memory, or information about the network environment, even the physical environment. The server knows the expected values for all the info and can compare the received data to it. If the data matches, the server grants access to whichever resource the node requires. If the attestation process fails, the server can deny further access or initiate a reinstall of the node firmware.

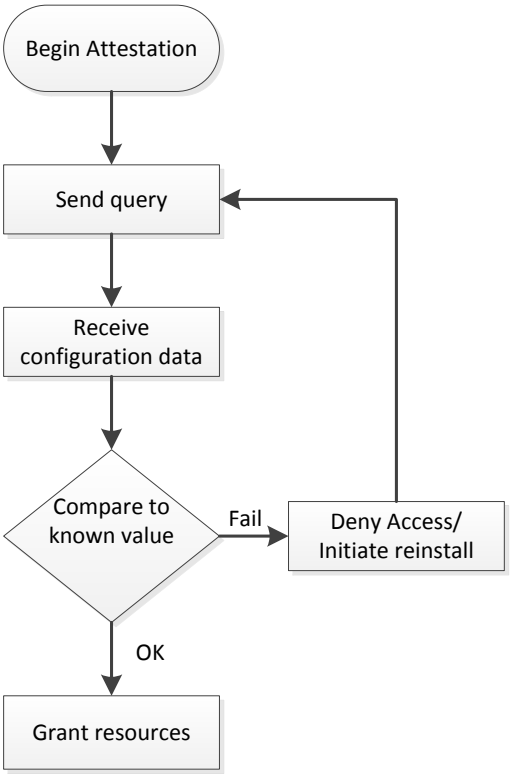

Figure 4.2 *Serverside Attestation*

However, for the attestation to work as intended, it needs to be built on components that

involve the use of cryptography and through that, random numbers. Moreover, the problem of random numbers with small embedded devices come from the fact that they might not have the capability to generate strong enough numbers. The strongness of numbers in this case is used to mean the ability of the attacker to predict the numbers being generated.

Furthermore, while attestation as a mechanism may be thought to be a great way to verify system integrity, there are some issues that have to be kept in mind when deploying systems using it. One of them is the fact that while attestation can verify the unchangeability of an executable program itself, it does not verify what the program in actuality executes [56]. Putting this into IoT context, it is possible to verify the software running on, for example, a temperature sensor node but further work is required for monitoring that the sensor actually sends correct data out.

Haldar et. al. also discuss the difference of open and closed systems with attestation [56]. Closed systems, such as Automated Teller Machines (ATM), have a life cycle that lends better to this type of standard attestation process especially considering system updates. As closed systems are tightly controlled software and hardware wise, managing the so called white-list of approved software and hardware components is easy. Devices in the IoT network can be modeled to be closed systems.

Open systems, such as standard desktop PCs, are harder to manage as their software and hardware configuration can be quite dynamic. Haldar et. al. discuss the issue of white-lists growing in size due to huge amount of different software and their versions that have to be kept updated as well [56]. The final issue with attestation mechanism is the key management, especially the process of revocation. There has to be a way to control the keys used for attestation in a situation where they become compromised.

5. HARDWARE ASSISTED ATTESTATION

The most common way to provide cryptographic services for attestation is to use dedicated hardware. This hardware can be used to handle the whole attestation process or augment a software-based mechanism. One of these hardware chips that handle the whole attestation process is the Trusted Platform Module (TPM). However, as the addition of extra hardware can be costly it might be wise to utilize the possible standard platform features such as the ARM TrustZone technology for building the attestation system.

5.1 Trusted Platform Module

A TPM is a specification defined by the TCG that combines cryptographic methods into a microchip that can be installed to a system to augment its security operations. A TPM consists of several key components each of which have their own important role in the operation of the module. A TPM consists of components that enable key management and a PRNG of its own for creation of the keys (Figure 5.1). On the key managament side, each TPM chip has a unique unchangeable key, the Attestation Identity Key (AIK), and a secure storage that can store also passwords and certificates among cryptographic keys [4].

There are however also software based implementations of the TPM functions. While the security might lack in comparison to the purely hardware-based implementations, there are several advantages to using software-based mechanisms as a part of the attestation protocol.

If the device in question has enough processing capacity, a semi-hardware based solution could be used. Lucyantie et al. proposed a framework that uses a separated trusted configuration servers for verifiying the client integrity [57]. This framework requires three different services to be provided for the client, application server that issues challenges to the client, foundation server that includes white and black lists of approved system components and a trusted Certificate Authority (CA) that issues credentials. Of these, the Trusted CA and the foundation server require an isolated network environment to quarantee their security. The framework also expects every participating device to have its

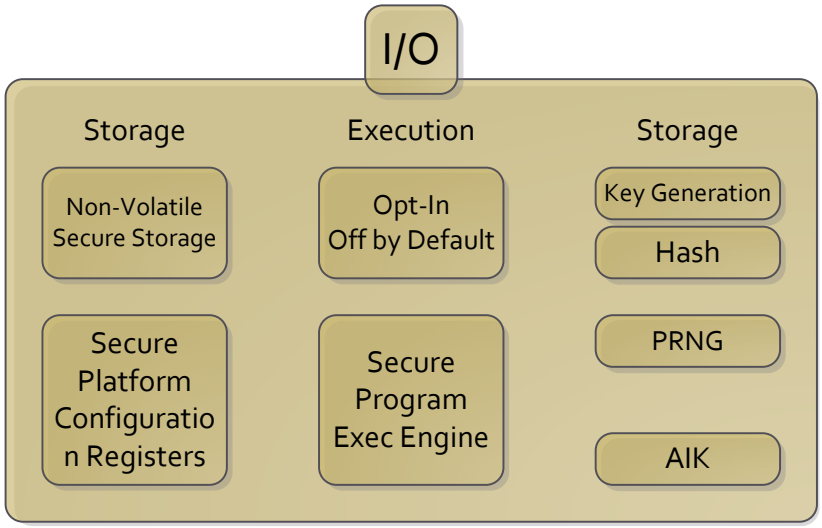

Figure 5.1 *TPM Components (Adapted from [4])*

own hardware TPM. As Lucyantie et al. states, this solution works fairly well with distributed systems such as cloud services. However, for the typical IoT scenario with simple low-power devices in public networks this kind of solution may prove to be difficult to deploy. The requirement of TPM in every device shuts down many small-size platforms and legacy hardware solutions and the network isolation might make device provisioning and update management more complex.

Another semi-hardware based attestation protocol was proposed by Francillion et al. [58]. Their solution uses only one external service, the challenger, to verify the integrity of the client, the prover. This protocol requires the platforms to have a set of features, such as protections for the key and the attestation algorithm that are manageable only by using hardware security features. Therefore also this solution might become too complex when devices to be used are desired to be as simple and cost-effective as possible.

Therefore, pure software based attestation methods could be hard to develop. However, their use within a single system for internal component verification is still a very relevant use scenario [58]. What are the issues in software remote attestation, then? One of the things is that most, if not all, of the previous ones are based on the challenge-response principle. One good example of a such protocol is SoftWare-ATTestation (SWATT) [59]. This protocol starts by the challenger sending a nonce to the prover which uses the nonce to seed its PRNG. This PRNG is used to initialize a pseudo-random memory traversal algorithm in which the prover takes a checksum of its memory and sends it back to the challenger.

The challenger has knowledge of the expected memory checksum and the hardware of the prover and thus can calculate the correct checksum and verify if the prover has or has not been compromised. The security of SWATT is based on the notion that if the attacker manages to run compromised code on the prover, the execution time of the algorithm will change from the expected one and such raises an alert of a failed integrity check. Seshadri et al. [59] also give an example of a highly optimized code that would not allow the attacker to insert their own checks within and thus calculate correct checksums. This protocol was further developed as the Pioneer primitive [60].

This timing-based verification is secure as long as there is no better optimized algorithms used to calculate the checksum. However, Castelluccia et al. managed to find one [61]. They argue that time-based attestation using pure software design has plenty of shortcomings and issues for secure enough design.

Shaneck et al. [11] proposed a pure software based remote attestation system as well. They build the attestation protocol on six blocks that are randomization of the attestation routine, encryption of the attestation code for adding complexity against disassembly, self-modifying code, opaque predicates and pointer aliasing and junk instructions. As usual with remote attestation, Shaneck et. al. use external base station to initiate the protocol and keep track of the expected memory configuration of the nodes. They argue that this scheme is even resistant against emulator-based attacks due the fact that emulation imposes slowdown in the code execution as instructions are run in software instead of hardware and thus can be detected if a proper time-out values are selected for the attestation process. However, modern emulation software can utilize hardware virtualization found from modern processors today which will indeed make the instruction execution faster.

5.2 Processor Features

Another way for adding an extra layer of security while still staying fairly cost-effective without specialty hardware components would be to utilize the security features provided by the system processor. One widely used processor in the embedded world that provides hardware security is the ARM. Most of the ARM cores come with TrustZone (TZ) technology that can be used to split each of the processor cores into two separate 'worlds' called the secure world and the normal world [7]. Processes running in the normal world can not see or access resources and processes that are running in the secure world without going through a secure monitor that acts as a gateway between the worlds.

The secure world could run completely different system, and be customized only to run the attestation related processes. By not using a standard operating system also on the

secure world the number of possible attack vectors could be lowered significantly. The normal world operating system can access only a defined subset of the hardware and has to communicate via the monitor to the secure world if it wants to access the rest. Nothing prevents running a whole operating system, such as Linux or BSD, on the secure world as well. As a matter a fact running in secure world is the default for many ARM devices such as the Raspberry Pi. Some of the devices are locked from fully utilizing the TZ technology as well.

However, the TZ technology does not necessarily allow for complete chain of trust for the device. The fact that (ZEDBOARD, RPi, for example) require the U-Boot loader to be loaded first before it is possible to load the secure monitor and the zone operating systems enables the attacker who has physical access to the device intercept the boot process. The TZ technology does also not protect against access to the boot medium, typically SD card, which houses the binaries for the secure monitor and operating system. These issues can be somewhat mitigated by burning the bootloader into a Read-Only Memory (ROM) chip and running it from there instead of the removable medium. However, if the bootloader has to be updated for any reason the whole ROM chip has to be changed.

6. HARDWARE PLATFORMS

For the purpose of this thesis, Raspberry Pi (RPi) was selected as the experimentation platform due its good software support and cheap price. Raspberry Pi in all its revisions is also widely used in industry which makes sure the older revisions are still being supported for several years [18]. The wide deployment in both industry and hobbyist use also has built a tremendous community that allows fast problem solving and good documentation to exist.

6.1 Raspberry Pi Models

The Raspberry Pi is a Single Board Computer (SBC) developed by the Raspberry Pi foundation. As of 2015, there exists 5 different models, each different in amount of memory and input/output (IO) options (Table 6.1) offered. All of the first generation models are based on the Broadcom BCM2835 System-on-chip (SOC) which includes a 700MHz single-core ARM1176JZF-S processor while the second generation model comes with a 900MHz quad-core ARM Cortex-A7 processor [**?**].The third generation model has been upgraded to a 64-bit quad-core ARM Cortex-A53 that runs at 1.2GHz and now integrates bluetooth and wireless network connectivity. All models include integrated Broadcom Videocore IV Graphics Processing Unit (GPU) which plays an important part in the inner workings of the Raspberry Pi.

Table 6.1 *Raspberry Pi Models*

	SoC	Memory	USB	GPIO	Power
Model A	BCM2835 ARMv6	256MB	1x	8	M
Model A+	BCM2835 ARMv6	256MB	1x	17	M
Model B	BCM2835 ARMv6	512MB	2x	12	T
Model B+	BCM2835 ARMv6	512MB	4x	17	
Gen 2 B	BCM2836 ARMv7	1024MB	4x	46	
Gen 3 B	BCM2837 ARMv8	1024MB	4x	40	

6.2 Raspberry Pi Boot Process

One curiosity that has to be taken into consideration when developing attestation on the Raspberry Pi is its boot process. Unlike standard Personal Computers (PC) that involve either Basic Input/Output Sytem (BIOS) or the newer Unified Extensible Firmware Interface (UEFI) based boot process or even U-Boot based embedded devices such as wireless routers, the Raspberry Pi involves its Videocore GPU heavily in the boot process [62].

Figure 6.1 shows the boot process from cold state. When the power is turned on, the ARM CPU is halted and a small first-stage bootloader is loaded to the GPU and executed [62][63]. This first-stage handles the mounting of the memory card and loading of the second-stage bootloader from it. This second-stage bootloader enables Random Access Memory (RAM) and loads the third-stage bootloader [62]. Now what is interesting is that the third-stage bootloader is also the firmware for the GPU as well as an Operating System itself, the Videocore Operating System (VCOS). The VCOS itself is still a mystery as any information about it is available only under Non-Disclosure Agreement (NDA) from Broadcom. However, the CPU itself provides some usable extensions for security [63].

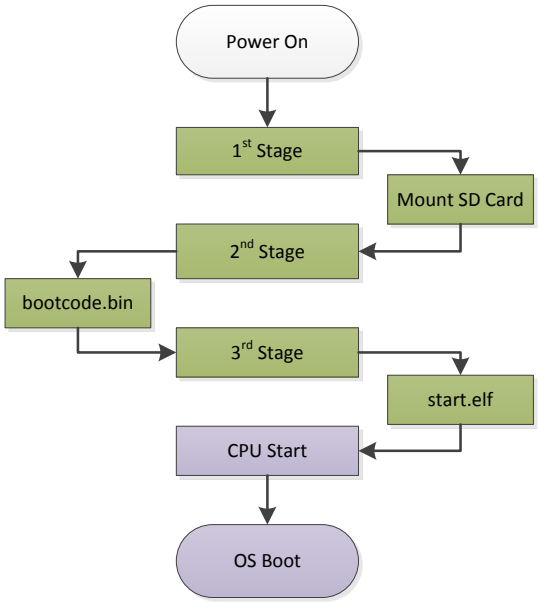

Figure 6.1 *Raspberry Pi Boot (Adapted from [62] and [63]). Green = running on GPU, Purple = Running on CPU.*

6.3 ARM TrustZone Security Extensions

What makes the Raspberry Pi interesting for attestation experimentation is the fact that the ARM CPU it has also includes the TrustZone (TZ) extensions. Figure 6.2 shows the ARM TZ implementation. The ARM core is separated into two different virtual cores called normal world and secure world. Between these worlds there is a gatekeeper dubbed monitor mode that handles the context switching between the two worlds [7].

The TZ technology essentially is hardware virtualization mechanism that allows multiple operating systems to be ran on one device at the same time yet separated from each other. Each core on a TZ enabled ARM processor comes with their own secure and normal world cores essentially doubling the available cores even on multicore ARM CPUs [7].

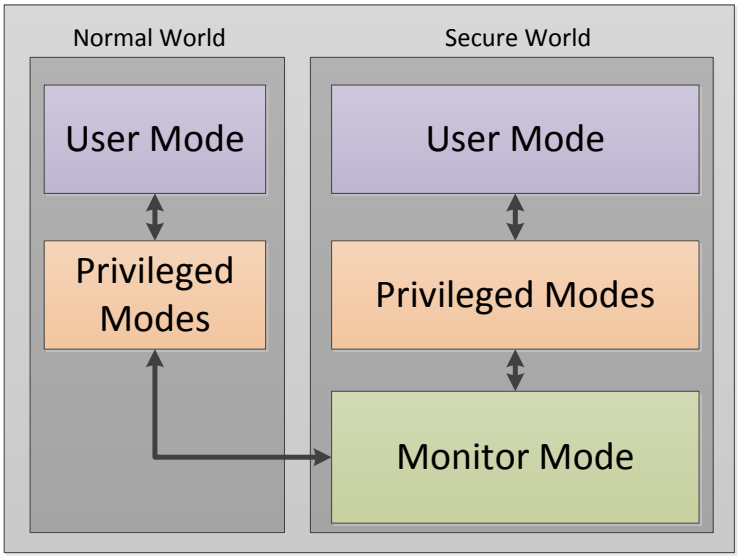

Figure 6.2 *TrustZone Architecture (Adapted from [7])*

The separation of these worlds enable access control to system resources as well. When the secure world can access all the resources available, the normal world processes can only access resources designated as non-secure [7]. An example scenario of such separation would be online payment where the device presents a keyboard application running in the secure world for processing secure user identification data instead of running everything on normal world [64]. After the secure information has been processed a context switch back to the normal world is done and normal operations resumed.

7. SOFTWARE PLATFORMS

Since Raspberry Pi is able to run various GPOSes, it can already support several software-based security mechanisms. Modern GPOSes already come with many security enhanced features that can be utilized for building attestation mechanisms. Of the GPOSes available for Raspberry Pi, FreeBSD and Linux-based NIXU Embedded Security Framework (NESF) were selected as the possible candidates for software attestation platforms. In addition, a hybrid software-hardware method utilizing TZ technology is proposed.

7.1 Built-in Security Features in Operating Systems

As seen in Table 7.1, these features range from Access Control Lists (ACL) to file system encryption as well as kernel based packet filtering. Especially in Open Source GPOS development, these security features can be audited better and by more people than closed-source systems.

Table 7.1 *GPOS Security Features*

	FreeBSD	OpenBSD	Raspbian	NESF
Chroot	X	X	X	X
Jail/Container	Jail	-	LC	Crypto
Kernel securelevels	4	4	-	-
MAC	X	-	X	X
File systems	UFS2, ZFS	UFS	ext4	ext4
FS Crypto	GELI	X	X	X
Memory protection	X	WX	X	X
App sandboxing	X	-	-	-

As seen from the table and owning to the fact that all of the discussed systems are either Unix-based or Unix-like, the most common security measure found is the chroot paravirtualization. This feature allows system process, for example a web server, to be confined to a single directory in the system. This adds an extra layer of protection as the server process can only access data in its own chrooted directory. If the server process gets compromised, the attacker can only modify and use the resources available in the

chroot. However, the standard chroot has been known to be vulnerable and possible to break which brought on the need for more security layers.

The next step up from plain chroot is the jail or container mechanism. This mechanism augments chroot with extra resource management and control. Most of the jails and containers also support encryption. Another mechanism that is not full jail/container is the Capsicum application sandboxing and capability framework found primarily in FreeBSD.

The concept of capabilities and application sandboxing basically means that any application running in capability mode can only access those resources allowed by set capability flags. For example, a web browser can be split into different processes such as the main browser process and the page renderer process. If an exploit is found under the renderer process, it is confined to this small subset instead of given full access to the system [65]. This light-weight sandboxing saves a lot more resources than running an entire copy of the operating system in a jail or a container.

Kernel securelevels are kernel-level protections which in various levels lock down the system access even from the superuser. Depending on the set securelevel, the restrictions prevent access to kernel memory, kernel module loader, filesystem flags and firewall rules. The only way to modify these is through rebooting the system. Kernel securelevels are primarily a BSD-based mechanism although development of a similar system on Linux is already commencing.

7.2 FreeBSD Specific Features

File systems can be encrypted as well on most of the GPOSes usable on the Raspberry Pi. FreeBSD comes with its own GELI framework that allows encryption of file systems [66], including the standard UFS as well as the more recent ZFS. Linux kernel includes an infrastructure called dm-crypt that allows stacking it with RAID or LVM volumes as well as mouting file systems directly [67]. File system access can be controlled with a physical token, such as USB keydisk, or passphrase.

The GELI framework with combination of the jail technology allows the creation of system jails that are encrypted and such simulating a cryptocontainer style system. Software package called ezjail can be used to automate the creation and management of such jails on FreeBSD systems [68]. Ezjail supports the use of partial read-only systems on jails which essentially mount the common system files from a shared directory as read-only filesystem to the jail. The only files that are modifiable in such jail are the user installed files. This increases jail security as the attacker can not affect the common configuration files or devices from the jail itself and can only attack the third party service running on

the jail.

Another type of jail that can be trivially provisioned using ezjail is the image jail. The image jail is essentially a file filled with zeroes and is of size n that is mounted as a jail-container and formated with any filesystem the system supports. Image jails are the ones that can be encrypted as well. As the image size can be selected during it's creation and as it essentially is just like any other file on the filesystem, image jails can be trivially shared over the network for provisioning and backup purposes.

When thinking of a system under attestation that would require some service to be run, an encrypted image jail could be created on the attestation server itself and then delivered to the device over encrypted channel after the device succesfully attests itself to the network. This would protect the initial service and device from even physical attacks as there is no services that can be exploited on the device before the attestation is succesfully completed.

NanoBSD is a build tool that allows FreeBSD release to be customized for embedded devices [69]. Among the possibility of scripting the whole deployment process from kernel configuration to package installation, NanoBSD also supports live upgrade of a running system. The NanoBSD system consists of three partitions:

Live partition that consists of the actual running system and is mounted in read-only mode.

Stand-by partition that is the partition where an upgrade image is uploaded and installed. After upgrade process is complete, this partition becomes the live partition and the old live partition turns into the backup one.

Config partition which is used to store the changeable system configuration files. The system will read the configuration during system boot before unmountin it and setting the running system into read-only mode.

The dual-partition design combined with read-only nature of the operating system make NanoBSD a good deal more secure than a standard installation of FreeBSD. When building the FreeBSD release for our target Raspberry Pi using the automated Crochet building tool, NanoBSD-style image compilation can be selected from the configuration settings.

One of the good sides of using a free open source GPOS as the main OS in an IoT device is the size of the community around it. As with Raspberry Pi itself, the OS also benefits from community support. The down side of development is the fact that support for different hardware platforms might be nonexistent until someone does it either free or for money. The more popular the OS is, more probable is the third party hardware support.

7.3 NIXU Embedded Security Framework

One way for attaining fairly secure purely software based attestation mechanism is to combine different techniques. The NIXU Embedded Security Framework (NESF) is a client-server remote attestation framework built on key-splitting and cryptocontainers [70]. The system itself is protected from reverse engineering by using obfuscation techniques and multiple hash functions.

As with the SWATT and Pioneer, also NESF uses an external server to control the node access and authenticity. Furthermore, the nodes are protected by encapsulating the file system inside a cryptocontainer that must be decrypted before the node can be fully booted [70]. The key to open this container has been split between the node itself and the server. During the attestation protocol the server queries information from the node about the node's hardware inventory and operating environment. The server has knowledge of the expected hardware and environment and thus can verify if the hash sent by the node matches the correct one.

NESF builds on Linux kernel and uses Busybox as the userland binary. Busybox is a single binary consisting of typical UNIX shell utilities and can be customized during the build phase [71]. Its primary use is to provide embedded systems a lightweight and simple yet highly customizable userland binary.

The kernel used is a standard Linux kernel with minor modifications to accommodate the attestation functionality. One of them is that the attestation client has been integrated to the Initial RAM Filesystem (initramfs) to allow the attestation process to run before the actual running system has been booted.

Another method NESF uses for tampering detection is the use of a construct of random hash functions. When building the OS binary, a set of hash functions will be linked to the client binary to random places [70]. The attestation server will have knowledge of the correct hash function set and their expected locations and so can verify the integrity data during the attestation process. A heavy use of code obfuscation protects the attestation client binaries from trivial disassembly making reverse engineering for the malicious user much harder.

If during the attestation process the server detects anomalies in the device, access to the key-half is denied and alarm raised for the operators to go and physically inspect the device. NESF supports over-the-air (OTA) image upgrades as well that can be used to reset the device in case of integrity check failure.

7.4 TOPPERS SafeG Dual-OS

As the Raspberry Pi comes with the ARM TrustZone extensions, it can be utilized to separate the running system into two different OSs running concurrently. The Toyohashi OPen Platform for Embedded Real-time Systems (TOPPERS) project [72] provides an open source implementation of such system running a GPOS on the TZ normal world and a real-time OS on the secure world concurrently [73]. Between the two worlds sits the SafeG secure monitor which will handle context switching between the normal and the secure world. Figure 7.1 presents the SafeG architecture.

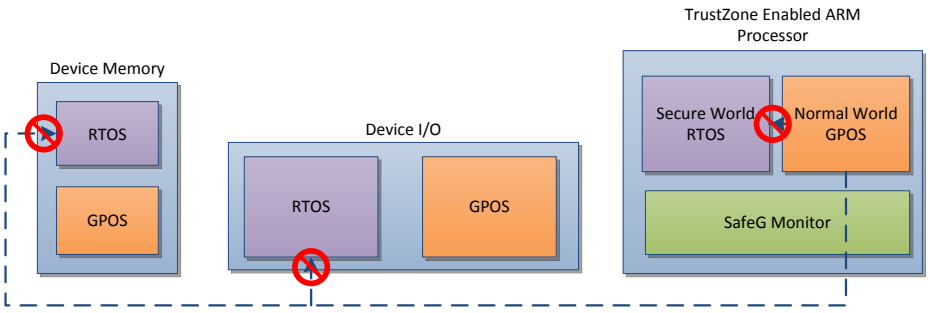

Figure 7.1 *TOPPERS SafeG Architecture (Adapted from [72])*

Figure 7.2 shows a typical run of the SafeG Dual-OS system on an U-Boot enabled embedded ARM device. Linux is used as the GPOS and the TOPPERS FMP as the RTOS. On power on, the device will load the U-boot loader which will load the SafeG, Linux Kernel and the FMP to memory. Next the SafeG monitor is executed and it starts loading the FMP which will start running its tasks. When FMP becomes idle, the SafeG monitor is called which will make a context switch to the normal world and execute the Linux kernel on it. When FMP timer interrupt rises, SafeG switches back to FMP execution [74].

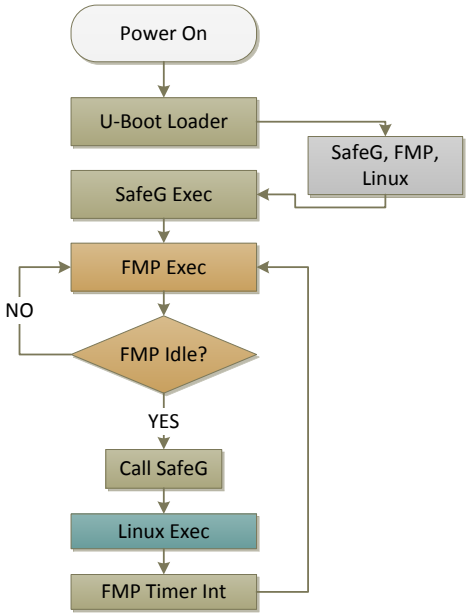

Figure 7.2 *TOPPERS SafeG OS Boot (Adapted from [74])*

The Linux system running on the normal world is isolated from system resources and can only access them after requesting the access via SafeG from the secure world. This enables the creation of an attestation service running on the secure world that allows access to system components only after the normal world proves that it has not been tampered with to the secure world attestation service.

The secure world operating system in the TOPPERS/FMP case is a kernel that runs tasks that have been compiled in the kernel binary during the install phase. This means that the FMP system can not be changed while it's running requiring a recompile and reload of the kernel image every time something needs to be updated. From security perspective this system helps to prevent malicious code injections on a running system and combined with the TZ security can deter even a brute force attack on the real-time OS. It is good to note, though, that unlike in the pure software based systems such as the NanoBSD, updates on the live system require a reboot so that the new images can be loaded by the U-Boot loader.

8. DISCUSSION

This thesis set out to investigate the viability of software-based remote attestation in the context of IoT devices. As the platform of choice for experimentation was selected to be the Raspberry Pi, it opened a possibility of using General Purpose Operating Systems as the platform to build attestation mechanisms on. One of the questions to be answered was to find out whether General Purpose Operating Systems already come with mechanisms usable for attestation especially in the case of software implementations. We also looked into a more specific OS, namely NESF, that builds on the Linux kernel and has been developed to include remote attestation features from ground up.

We also investigated the viability of the hardware platform itself as embedded devices, what the Rasberry Pi essentially also is, are constrained in processing power and hardware features. The requirements for securely running cryptographic operations can be hard to attain without external hardware or processing. Finally considerations for a hybrid-model where a part of the attestation mechanism would be hardware-based yet still keeping the fairly portable feature of software based solution along.

The previous work of Seshadri et. al [59][60] as well as development made in NESF does show that the concepts are viable but not without problems. If physical security is to be taken into consideration as well, the devices become quite exposed due the lack of chain of trust established during the device boot process. This leads to more effort required in securing the post-boot system. The lack of TPM and its provided secure configuration registers will make the device integrity measurement storage design important. This is clearly evident especially considering a standard GPOS running on the default Raspberry Pi installation. The open boot process of the Raspberry Pi allows easy manipulation of the boot image and access to the GPOS file system by simply taking the SD-card out of the device and mounting it externally. To our knowledge, access to the Raspberry Pi Video-Core OS source code is locked under a non-disclosure agreement making development of a third party boot mechanisms harder.

The use of third party attestation server will mitigate some of these matters. The most common way for verifying the attested device integrity is to compare it to checksums taken of the hardware and software configuration during the initial install and deployment

of the device. These hash values are stored on the server in a secure space. Another way for the server to verify the attestation data is to actually calculate it itself. As the attestation servers are required to know the configuration of the device, they can use the same processes that the IoT device uses to calculate it's own integrity data to verify if it matches the expected one. Since the servers are not constrained by hardware, they can have a lot of processing power to calculate the expected integrity values quickly.

The augmentation of the attestation using standard platform hardware features could be one area of development. As the devices used for IoT are more and more using ARM-based hardware, it could be wise to utilize the TZ technology in development of attestation mechanisms. By using a hybrid TZ-Software attestation system, one could get the best of both worlds in terms of security and cost-effectiveness. What would be sacrificed is a little bit of portability as TZ is only available in ARM-processors.

Combining technologies, or layering them, is one way to provide extended security measures for the devices as the NESF shows. The idea of separating the system into two different "worlds" is a well known hardware feature but as shown, can be somewhat created in software as well. The way how NESF handles the attestation process is also quite workable. By integrating part of the attestation service to the kernel itself it can boot the system to working condition before running the attestation protocol and enabling access to the rest of the system. As NESF targets the Raspberry Pi it also is vulnerable to attacks that enable access to the mounted SD-card. Cryptocontainers and obfuscation mechanisms do prevent direct attacks and require effort from the attacker to access the services. If NESF could be booted mostly from the network as the U-Boot bootloader used in the Raspberry Pi is able to do, attacks against the kernel and the cryptocontainers could be mitigated. Howeever, the fact that the SD-card is encrypted and at minimum require unsecured binaries of the second and third stage boot images still remain.

The TZ would allow the attestation client run on the secure world and as such, be protected against attacks even from the normal world operating system. If the secure world OS is a RTOS with attestation tasks compiled into the OS binary itself, as can be done with the TOPPERS FMP kernel, the attack surface will be significantly lower even if the attacker gets access to the GPOS running on the normal world. Considering the chosen experimentation platform, this separation of secure and normal world does not protect the device against physical attacks as even with TZ enabled, the boot process does not enable chain of trust when rebooting cold.

The key could partly be integrated in the secure world OS binary and the other side could be hidden to external device. The normal world OS could access part of the key via secure monitor API and the rest of the key directly via network using cryptographic protocol or

a puzzle. Another way would be to have the normal world OS issue a key request to the secure world OS which would make the secure world OS to attest the normal world and then request the part of the key itself from the external device. The secure world OS would then handle all the key management and just give the whole key to the normal world OS for opening the containers.

9. CONCLUSIONS

This thesis investigated the viability of remote software based attestation in the context of Internet of Things devices. As the number of devices with varying states of automation increases globally in situations as diverse as domestic households or factories, their management and security become more and more an issue. By using attestation methods for authenticating and securing the devices in the networks we can be sure that domestic and business consumers can continue operating safely in the constantly changing networked world.

Since the typical way of enabling device attestation is by hardware based mechanisms, a way to provide similar services using only software based ones was deemed to be a viable choice in the IoT context. For example, cost-effectiveness, portability and scalability are notable advantages when comparing software to hardware mechanisms. However, securing software based attestation mechanisms can be more difficult than hardware implementations. One of the problems in this regard was the lack of secure boot especially on the chosen experimentation platform, the Rasperry Pi.

As the Raspberry Pi is powerful enough to run Operating Systems such as Linux or FreeBSD it is also possible to utilize the built in security features integrated in them for building the attestation mechanisms. These mechanisms offer support for e.g. encrypted filesystems, access control and application or system-wide sandboxing that all are part of the attestation concept. Specialty distributions such as NESF extend the operating system with custom functionality but sacrifice portability as the addons are quite operating system specific.

Another issue with software based attestation mechanisms on constrained devices is the possible lack of cryptographic hardware. While the Raspberry Pi is quite a powerful device capable of running various cryptosystems, the IoT network may also include sensors that have a lot less capacity to run functions and while a way to provide cryptographic services, such as PRNG or entropy delivery over the network would mitigate some of the issues, it would still require the deployment of secure channels for delivery. Luckily the field of cryptography is advancing and the rise of ECC has given a lot of options in implementing usable cryptofunctions even on the tiniest of devices.

However, by utilizing the integrated features of the hardware platform itself the attestation functionality can be further secured. As more and more embedded devices are built on top of ARM-based hardware, an attestation solution built on the TZ technology could be a viable solution. By splitting the attestation process to run inside the TZ secure world, another layer of security is added to the attestation process itself as well as the GPOS that is running on the normal world. However, depending on the used platform even this system can be vulnerable to physical attacks and as is the case with the Raspberry Pi, the boot process still remains unsecured.

Software based remote attestation is a challenge to get right. Issues with physical security and constrained hardware definitely support the inclusion of specific cryptographic hardware in the devices. The hybrid system utilizing hardware features still keeping most of the functionality in software is a fine compromise if using the same hardware platforms even if running different Operating Systems. However, as IoT devices are being deployed everywhere with a huge amount of different hardware configurations, the need for a cost-effective and portable software based remote attestation is definitely there.

BIBLIOGRAPHY

[1] M. Weiser, "The computer for the 21st century," *Scientific american*, vol. 265, no. 3, pp. 94–104, 1991.

[2] M. Strasser and P. Sevnic, "A software-based TPM emulator for Linux," *Semesterarbeit, ETH Zurich*, 2004, accessed on May. 23, 2016. [Online]. Available: http://archiv.infsec.ethz.ch/education/projects/archive/TPMEmulatorReport.pdf

[3] IBM Corporation, "Software TPM," accessed on May. 23, 2016. [Online]. Available: http://ibmswtpm.sourceforge.net/

[4] Trusted Computing Group, "Trusted Platform Module (TPM) Summary," accessed on May. 23, 2016. [Online]. Available: http://www.trustedcomputinggroup.org/resources/trusted_platform_module_tpm_summary

[5] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "Spins: Security protocols for sensor networks," *Wireless networks*, vol. 8, no. 5, pp. 521–534, 2002.

[6] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach: International Edition*. Pearson Higher Ed, 2013.

[7] ARM Limited, "Arm security technology: Building a secure system using trustzone technology," 2009.

[8] D. R. Raymond and S. F. Midkiff, "Denial-of-service in wireless sensor networks: Attacks and defenses," *Pervasive Computing, IEEE*, vol. 7, no. 1, pp. 74–81, 2008.

[9] P. Charles and S. L. Pfleeger, *Analyzing Computer Security: A Threat/vulnerability/countermeasure Approach*. Prentice Hall, 2012.

[10] D. R. Raymond, R. C. Marchany, M. Brownfield, S. F. Midkiff *et al.*, "Effects of denial-of-sleep attacks on wireless sensor network mac protocols," *Vehicular Technology, IEEE Transactions on*, vol. 58, no. 1, pp. 367–380, 2009.

[11] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim, "Remote software-based attestation for wireless sensors," in *Security and Privacy in Ad-hoc and Sensor Networks*. Springer, 2005, pp. 27–41.

[12] H. Kagermann, J. Helbig, A. Hellinger, and W. Wahlster, *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0: Securing the Future of German Manufacturing Industry; Final Report of the Industrie 4.0 Working Group*. Forschungsunion, 2013.

[13] S. Wang, J. Wan, D. Li, and C. Zhang, "Implementing smart factory of industrie 4.0: an outlook," *International Journal of Distributed Sensor Networks*, vol. 2016, 2016.

[14] D. Jansen and H. Buttner, "Real-time ethernet: the ethercat solution," *Computing and Control Engineering*, vol. 15, no. 1, pp. 16–21, 2004.

[15] M. Kovatsch, S. Mayer, and B. Ostermaier, "Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*. IEEE, 2012, pp. 751–756.

[16] National Institute of Standards and Technology, "Security considerations in the system development life cycle," 2008.

[17] T. R. Peltier, J. Peltier, and J. A. Blackley, *Information Security Fundamentals*. CRC Press, 2005.

[18] E. Upton and G. Halfacree, *Raspberry Pi user guide*. John Wiley & Sons, 2014.

[19] H. F. Murry, "A general approach for generating natural random variables," *IEEE Trans. Comput.*, vol. 100, no. 12, pp. 1210–1213, 1970.

[20] T. De Raadt, N. Hallqvist, A. Grabowski, A. D. Keromytis, and N. Provos, "Cryptography in openbsd: An overview." in *USENIX Annual Technical Conference, FREENIX Track*, 1999, pp. 93–101.

[21] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography engineering: design principles and practical applications*. John Wiley & Sons, 2012.

[22] D. Mahu, V. Dumitrel, and F. Pop, "Secure entropy gatherer," in *Control Systems and Computer Science (CSCS), 2015 20th International Conference on*. IEEE, 2015, pp. 185–190.

[23] J. Kelsey, B. Schneier, and N. Ferguson, "Yarrow-160: Notes on the design and analysis of the yarrow cryptographic pseudorandom number generator," in *Selected Areas in Cryptography*. Springer, 2000, pp. 13–33.

[24] M. Stevens, "Fast collision attack on MD5," *IACR Cryptology ePrint Archive*, vol. 2006, p. 104, 2006.

[25] R. Impagliazzo, L. A. Levin, and M. Luby, "Pseudo-random generation from one-way functions," in *Proceedings of the twenty-first annual ACM symposium on Theory of computing*. ACM, 1989, pp. 12–24.

[26] B. Preneel, "Cryptographic hash functions," *European Transactions on Telecommunications*, vol. 5, no. 4, pp. 431–448, 1994.

[27] M. Naor and M. Yung, "Universal one-way hash functions and their cryptographic applications," in *Proceedings of the twenty-first annual ACM symposium on Theory of computing*. ACM, 1989, pp. 33–43.

[28] M. Girault, R. Cohen, and M. Campana, "A generalized birthday attack," in *Advances in Cryptology—EUROCRYPT'88*. Springer, 1988, pp. 129–156.

[29] National Institute of Standards and Technology, "Descriptions of SHA-256, SHA-384 and SHA-512," 2001.

[30] S.-j. Chang, R. Perlner, W. E. Burr, M. S. Turan, J. M. Kelsey, S. Paul, and L. E. Bassham, *Third-round report of the SHA-3 cryptographic hash algorithm competition*. Citeseer, 2012.

[31] M. Kelly, A. Kaminsky, M. Kurdziel, M. Lukowiak, and S. Radziszowski, "Customizable sponge-based authenticated encryption using 16-bit s-boxes," *in Military Communications Conference, MILCOM 2015*, pp. 43–48, Oct 2015.

[32] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein, "BLAKE2: simpler, smaller, fast as MD5," in *Applied Cryptography and Network Security*. Springer, 2013, pp. 119–135.

[33] H. Delfs and H. Knebl, *Introduction to Cryptography: Principles and Applications*. New York, NY, USA: Springer-Verlag New York, Inc., 2001.

[34] B. Schneier, "Why digital signatures are not signatures," 2000, accessed on May. 23, 2016. [Online]. Available: Crypto-GramNewsletter,https://www.schneier.com/crypto-gram-0011.html

[35] M. Hellman, "An overview of public key cryptography," *IEEE Communications Magazine*, vol. 16, no. 6, pp. 24–32, 1978.

[36] D. R. Stinson, *Cryptography: theory and practice*. CRC press, 1995.

[37] D. J. Bernstein, "The salsa20 family of stream ciphers," in *New stream cipher designs*. Springer, 2008, pp. 84–97.

[38] D. Bernstein, "Cache-timing attacks on AES," 2005.

[39] M.-J. O. Saarinen, "The STRIBOBr1 authenticated encryption algorithm," 2014. [Online]. Available: CAESAR,1stRoundwww.stribob.com

[40] V. Rijmen and P. Barreto, "The WHIRLPOOL hash function," p. 72, 2001, accessed on May. 23, 2016. [Online]. Available: Website.http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html

[41] M.-J. O. Saarinen and B. B. Brumley, "WHIRLBOB, the whirlpool based variant of STRIBOB," in *Secure IT Systems*. Springer, 2015, pp. 106–122.

[42] R. Housley, "Triple-DES and RC2 Key Wrapping," Internet Requests for Comments, RFC Editor, RFC 3217, December 2001.

[43] R. Gennaro and S. Halevi, "More on key wrapping," in *Selected Areas in Cryptography*. Springer, 2009, pp. 53–70.

[44] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[45] N. Sullivan, "A (relatively easy to understand) primer on elliptic curve cryptography," *Everything you wanted to know about the next generation of public key crypto*, 2013, accessed on May. 23, 2016. [Online]. Available: http://arstechnica.com/security/2013/10/a-relatively-easy-tounderstand-primer-on-elliptic-curve-cryptography

[46] L. Batina, N. Mentens, K. Sakiyama, B. Preneel, and I. Verbauwhede, "Low-cost elliptic curve cryptography for wireless sensor networks," in *Security and Privacy in Ad-Hoc and Sensor Networks*. Springer, 2006, pp. 6–17.

[47] J. H. Silverman, "An introduction to the theory of elliptic curves," 2006, accessed on May. 23, 2016. [Online]. Available: http://www.math.brown.edu/jhs/Presentations/WyomingEllipticCurve.pdf

[48] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, 2001.

[49] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing elliptic curve cryptography and rsa on 8-bit cpus," in *Cryptographic hardware and embedded systems-CHES 2004*. Springer, 2004, pp. 119–132.

[50] P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab, "Nanoecc: Testing the limits of elliptic curve cryptography in sensor networks," in *Wireless sensor networks*. Springer, 2008, pp. 305–320.

[51] M. Sethi, J. Arkko, A. Keranen, and H.-M. Rissanen, "Practical considerations and implementation experiences in securing smart object networks," Working

Draft, IETF Secretariat, Internet-Draft draft-aks-crypto-sensors-02, March 2012, accessed on May. 23, 2016. [Online]. Available: http://www.ietf.org/internet-drafts/draft-aks-crypto-sensors-02.txt

[52] T. C. Group, "Architecture overview, revision 1.4," 2007.

[53] L. Chen, H. Löhr, M. Manulis, and A.-R. Sadeghi, "Property-based attestation without a trusted third party," in *Information Security*. Springer, 2008, pp. 31–46.

[54] D. D. Clark and D. R. Wilson, "A comparison of commercial and military computer security policies," in *Security and Privacy, 1987 IEEE Symposium on*. IEEE, 1987, pp. 184–184.

[55] R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn, "Design and implementation of a tcg-based integrity measurement architecture." in *USENIX Security Symposium*, vol. 13, 2004, pp. 223–238.

[56] V. Haldar, D. Chandra, and M. Franz, "Semantic remote attestation: a virtual machine directed approach to trusted computing," in *USENIX Virtual Machine Research and Technology Symposium*, vol. 2004, 2004.

[57] M. Lucyantie, H. Habibah, M. Mohd Anuar, and A. A. Norazah, "Attestation with trusted configuration machine," in *Computer Applications and Industrial Electronics (ICCAIE), 2011 IEEE International Conference on*, Dec 2011, pp. 570–573.

[58] A. Francillon, Q. Nguyen, K. Rasmussen, and G. Tsudik, "A minimalist approach to remote attestation," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, March 2014, pp. 1–6.

[59] A. Seshadri, A. Perrig, L. Van Doorn, and P. Khosla, "Swatt: Software-based attestation for embedded devices," in *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*. IEEE, 2004, pp. 272–282.

[60] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla, "Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems," *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5, pp. 1–16, 2005.

[61] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente, "On the difficulty of software-based attestation of embedded devices," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 400–409.

[62] S. Paranagama, "How the Raspberry Pi Boots Up," accessed on May. 23, 2016. [Online]. Available: https://thekandyancode.wordpress.com/2013/09/21/how-the-raspberry-pi-boots-up/

[63] OSDev Wiki, "ARM Raspberry Pi," accessed on May. 23, 2016. [Online]. Available: http://http://wiki.osdev.org/Raspberry_Pi/

[64] W. Kurisu, "Securing IoT Devices With ARM TrustZone," 2014, accessed on May. 23, 2016. [Online]. Available: http://www.eetimes.com/author.asp?doc_id=1323543

[65] R. N. Watson, J. Anderson, B. Laurie, and K. Kennaway, "Capsicum: Practical capabilities for unix." in *USENIX Security Symposium*, vol. 46, 2010, p. 2.

[66] M. Schiesser, "Complete hard disk encryption using freebsd's geom framework," in *Proc. 4th European BSD Conf. Available from http://events. ccc. de/congress/2005/fahrplan/attachments/586-paper_Complete_Hard_Disk_Encryption. pdf [Accessed on May. 15, 2016]*, 2005.

[67] , "dm-crypt/Encrypting an entire system," accessed on May. 25, 2016. [Online]. Available: https://wiki.archlinux.org/index.php/Dm-crypt/Encrypting_an_entire_system

[68] D. Engling, "ezjail - Jail administration framework," 2016, accessed on May. 25, 2016. [Online]. Available: http://http://erdgeist.org/arts/software/ezjail/

[69] D. Gerzo, "Introduction to NanoBSD," 2006, accessed on May. 25, 2016. [Online]. Available: https://www.freebsd.org/doc/en/articles/nanobsd/

[70] NIXU Oy, "NESF and obfuscation," *NIXU Development Projects Wiki*, 2013.

[71] D. Vlasenko, "BusyBox," 2012, accessed on May. 16, 2016. [Online]. Available: http://busybox.net

[72] Toyohashi Open Platform for Embedded Real-time Systems (TOPPERS, "SafeG," 2016, accessed on May. 16, 2016. [Online]. Available: http://toppers.jp/en/safeg.html

[73] D. Sangorrin, S. Honda, and H. Takada, "Dual operating system architecture for real-time embedded systems," in *Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT), Brussels, Belgium*. Citeseer, 2010, pp. 6–15.

[74] D. Sangorrin, "TOPPERS/SafeG build process overview," *SafeG Documentation*, 2013.