Tampere University

Arttu Paju

# DISTRIBUTED EAAS SIMULATION USING TEES

A case study in the implementation and practical application of an embedded computer cluster

Master of Science Thesis
Faculty of Information Technology and Communication Sciences (ITC)
Examiners: Prof. Billy Bob Brumley
M.Sc. Antti Kolehmainen
D.Sc. Juha Nurmi
November 2022

# ABSTRACT

Arttu Paju: Distributed EaaS simulation using TEEs
Master of Science Thesis
Tampere University
Master's Programme in Information Technology
November 2022

---

Internet of Things (IoT) devices with limited resources struggle to generate the high-quality entropy required for high-quality randomness. This results in weak cryptographic keys. As keys are a single point of failure in modern cryptography, IoT devices performing cryptographic operations may be susceptible to a variety of attacks.

To address this issue, we develop an Entropy as a Service (EaaS) simulation. The purpose of EaaS is to provide IoT devices with high-quality entropy as a service so that they can use it to generate strong keys. Additionally, we utilise Trusted Execution Environments (TEEs) in the simulation. TEE is a secure processor component that provides data protection, integrity, and confidentiality for select applications running on the processor by isolating them from other system processes (including the OS). TEE thereby enhances system security.

The EaaS simulation is performed on a computer cluster known as the *Magi cluster*. Magi cluster is a private computer cluster that has been designed, built, configured, and tested as part of this thesis to meet the requirements of Tampere University's Network and Information Security Group (NISEC). In this thesis, we explain how the Magi cluster is implemented and how it is utilised to conduct a distributed EaaS simulation utilising TEEs.

Keywords: Computer Cluster, TEE, Trusted Execution Environment, EaaS, Entropy as a Service, Cryptography, RNG, Random Number Generator, IoT, Internet of Things

# TIIVISTELMÄ

Arttu Paju: Hajautettu EaaS simulaatio TEE:tä hyödyntäen
Diplomityö
Tampereen yliopisto
Tietotekniikan DI-ohjelma
Marraskuu 2022

Esineiden internetin (Internet of Things, IoT) laitteilla on tyypillisesti rajallisten resurssien vuoksi haasteita tuottaa tarpeeksi korkealaatuista entropiaa vahvan satunnaisuuden luomiseen. Tämä johtaa heikkoihin salausavaimiin. Koska salausavaimet ovat modernin kryptografian heikoin lenkki, IoT-laitteilla tehtävät kryptografiset operaatiot saattavat olla haavoittuvaisia useita erilaisia hyökkäyksiä vastaan.

Ratkaistaksemme tämän ongelman kehitämme simulaation, joka tarjoaa IoT-laitteille vahvaa entropiaa palveluna (Entropy as a Service, EaaS). EaaS-simulaation ideana on jakaa korkealaatuista entropiaa palveluna IoT-laitteille, jotta ne pystyvät luomaan vahvoja salausavaimia. Hyödynnämme simulaatiossa lisäksi luotettuja suoritusympäristöjä (Trusted Execution Environment, TEE). TEE on prosessorilla oleva erillinen komponentti, joka tarjoaa eristetyn ja turvallisen ajoympäristön valituille ohjelmille. TEE:tä hyödyntämällä ajonaikaiselle ohjelmalle voidaan taata datan suojaus, luottamuksellisuus sekä eheys eristämällä se muista järjestelmällä ajetuista ohjelmista (mukaan lukien käyttöjärjestelmä). Näin ollen TEE parantaa järjestelmän tietoturvallisuutta.

EaaS-simulaatio toteutetaan *Magi*-nimisellä tietokoneklusterilla. Magi on Tampereen Yliopiston Network and Information Security Group (NISEC) -tutkimusryhmän oma yksityinen klusteri, joka on suunniteltu, rakennettu, määritelty ja testattu osana tätä diplomityötä. Tässä diplomityössä käymme läpi, kuinka Magi-klusteri on toteutettu ja kuinka sillä toteutetaan hajautettu EaaS-simulaatio hyödyntäen TEE:itä.

Avainsanat: Klusteri, TEE, Luotettu Suoritusympäristö, EaaS, Entropia Palveluna, Kryptografia, RNG, Satunnaislukugeneraattori, IoT, Esineiden Internet

# PREFACE

This Master of Science thesis was done in the Network and Information Security Group (NISEC) at the Faculty of Information Technology and Communication Sciences (ITC) in Tampere University. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 804476).

I wish to express my deepest gratitude to my supervisors professor Billy Bob Brumley, Antti Kolehmainen, and Juha Nurmi for their invaluable guidance and support, not only for this thesis but also as colleagues.

On this note, I would also like to thank my other current and former NISEC colleagues, particularly Markku, Nicola, Cesar, Nacho, and Owais, with whom I have had the pleasure of working closely. Working at NISEC has provided me with an extraordinary opportunity to hone my skills and academic knowledge in an ambitious, yet laid-back research group that generates world-class research, for which I am grateful.

Finally, I would like to thank my family and friends for all their support!

Tampere, 4th November 2022

Arttu Paju

# CONTENTS

# LIST OF FIGURES

# 1. INTRODUCTION

With the proliferation of embedded systems and the Internet of Things (IoT), the world is becoming pervasive. As the number of IoT devices continues to rise, securing them becomes essential. On the basis of real-world examples and academic research, we know IoT devices are susceptible to multiple security flaws. We are also aware that cryptography can eliminate the majority of these security vulnerabilities. However, because IoT devices have limited resources, it is difficult for them to generate the robust randomness required for effective cryptographic use. Entropy as a Service (EaaS) is one possible solution to this issue. In this thesis, we implement an EaaS simulation using an embedded computer cluster called Magi cluster, which we design and implement for the internal use of Tampere University's Network and Information Security Group (NISEC).

## 1.1 Motivation and goals

The security of embedded systems and IoT devices is one of NISEC's key research areas. This is the impetus for implementing an embedded computer cluster and simulating EaaS. We are particularly interested in implementing our own EaaS system due to the paucity of academic literature on the topic. We are also interested in determining the viability of distributed Trusted Execution Environment (TEE) utilisation with the EaaS simulation. As NISEC is also researching the security differences between ideal and real-world cryptosystem implementations, we also attempt to account for them in this thesis. Besides the EaaS simulation, NISEC has additional applications for an embedded computer cluster. Examples include evaluating the embedded performance of security-focused protocols and efficiently running ARM-based cryptographic software. This justifies the implementation of the Magi cluster.

Our goal with the thesis is to implement an embedded computer cluster that satisfies NISEC's research requirements and use it to provide a functional EaaS service for generating strong entropy for IoT devices. In addition, we intend to implement the simulation using TEEs in order to add an additional layer of security to both the IoT devices utilising the generated entropy and the EaaS server itself.

## 1.2 Contributions

We present the following two novel contributions in this thesis:

- We design and implement an embedded computer for NISEC's internal use.
- We use the cluster to design and implement an EaaS simulation for resource-constrained IoT devices. The EaaS simulation utilises TEEs.

In support of open science, we have published the EaaS simulation code repository [1] under open-source (MIT) license. In addition, as part of the research for this thesis, we authored a Systemisation of Knowledge (SoK) paper on Trusted Application (TA) development for TEEs [2].

## 1.3 Structure and research questions

This thesis is structured as follows:

In Chapter 2, we give relevant background information about the current state of the knowledge in the fields related to this thesis. Chapter 3 explains how the Magi cluster is designed and implemented based on the requirements of the NISEC. In Chapter 4, we present an EaaS simulation scheme that is carried out with the Magi cluster. The purpose of the simulation is to address the issue of modern IoT devices being incapable of generating high-quality randomness, which renders their cryptographic-level communication insecure. Chapter 5 goes through the results of the EaaS simulation. Finally, in Chapter 6, we provide overall conclusions of the work done in this thesis, give clear answers to all the research questions, and propose future work.

The research questions we seek to address in this thesis are the following:

**RQ1.** How did we implement a computer cluster to meet the needs of our research group?

**RQ2.** How does IoT RNG fail in practice?

**RQ3.** Is it feasible to create an EaaS system using an embedded computer cluster?

**RQ4.** How can TEEs be used to harden the security of an EaaS system?

**RQ5.** Is it feasible to develop TAs that utilise TEEs for the EaaS system?

**RQ6.** Is EaaS capable of resolving existing IoT RNG issues?

Table 1.1 highlights the mapping between the research questions and the sections in which they are answered.

***Table 1.1.*** *The mapping between research questions and the sections in which they are answered.*

| **Research Question** | **Answered in** |
|---|---|
| **RQ1** | Section 2.1, Chapter 3 |
| **RQ2** | Section 2.2, Section 2.4, Section 2.5 |
| **RQ3** | Chapter 4, Section 5.1 |
| **RQ4** | Section 2.3, Chapter 4, Section 5.2 |
| **RQ5** | Section 2.3, Chapter 4, Section 5.2 |
| **RQ6** | Chapter 4, Chapter 5, Chapter 6 (based on **RQ2** and **RQ3**) |

As can be seen from Table 1.1, each research questions is addressed in multiple sections and chapters. This is because we need both theoretical background information and practical experience from the Magi cluster and the EaaS simulation implementations in order to properly answer the research questions.

# 2. BACKGROUND

In this chapter, we summarise the current state of the current knowledge in all the fields relevant to the thesis' topic. In Section 2.1, we describe relevant background information for computer clusters. This knowledge serves as the foundation for our design and implementation of the Magi cluster in Chapter 3. The remaining sections of this chapter address embedded systems and IoT, TEEs, cryptography, randomness and random number generators, and randomness beacons and EaaS respectively. These sections provide the foundation for our EaaS simulation, which is introduced in Chapter 4 and whose findings are introduced in Chapter 5.

## 2.1 Cluster computing

A computer cluster is a group of computers that perform individual tasks together [3]. Computer clusters consist of administrative servers and computing nodes interconnected via a high-speed network. Each node in a computer cluster is scheduled to perform the same tasks [4]. The servers are responsible for task scheduling and controlling the computer cluster. Typically, they do not contribute to the actual computation, which is performed by the nodes. By definition, each computer cluster has to consist of multiple nodes in order to form a cluster of computers. Cluster computing refers to the computing done with computer clusters.

Each computer cluster has its own use cases, and no two computer clusters are identical or utilised in the same manner. Consequently, the design and implementation of a given computer cluster must be based on the cluster's specific requirements. The requirements vary based on the computer cluster's use cases, the available resources, and the stakeholders' preferences. Given that computer clusters are large combinations of multiple computers, their peripherals, and cables, designing the cluster's architecture is a complex procedure. Typically, data storage is implemented with a separate pool of multiple hard drives. All nodes share this pool of shared data. In addition, nodes may have their own smaller data storage systems that can be provided to individual users.

Even though the requirements and use cases for the various computer clusters are vastly different, there are some design choices that recur across clusters. For instance, almost all of the top 500 most powerful supercomputers in the world currently use Linux as their operating system [5]. According to Yeo et al. [6], Linux is the most widely used cluster operating system because it follows the Free and Open-Source Software (FOSS) principles and has a wide user and developer community. In contrast to FOSS OSs, there are also commercial cluster OSs that are proprietary and shipped along with commercial clusters. Examples include IBM AIX [7] and Sun Solaris MC [8].

### 2.1.1 Classification and usage of computer clusters

In this thesis, we employ our own classification scheme for computer clusters. The classification is based on the cluster's use cases. The classes are:

1. Shared production-level clusters
2. Private computer clusters designed for specific use cases

High-Performance Computing (HPC) clusters [4, 6] are the most powerful computer clusters, and they are typically funded by large consortia consisting of multiple individuals, organisations, businesses, or even governments. They are intended for **shared production level use between multiple parties**. HPC clusters are widely utilised for research and development applications in science, engineering, commerce, and industry according to Yeo et al. [6].

Practical applications for HPC clusters in the scientific community are virtually limitless. For instance, mathematicians can use the computing power provided by computer clusters in weather forecasting [9–11], biomedical scientists can use it to simulate and analyse the efficacy of various medical treatments [12–14], and astrophysicists can simulate the formation of galaxies [15] or study the structures of accretion disks formed by supermassive black holes [16] among many other interesting applications. All of these problems are complex and involve enormous amounts of data that must be taken into account during simulations. Standard consumer-grade workstations would require an excessive amount of time to process them. Therefore, HPC clusters are utilised to solve such problems in a timely manner.

In contrast to shared production-level clusters, there are also **private computer clusters designed for specific use cases**. Many companies, research groups, and even enthusiastic individuals design their own computer clusters that are not meant to be used by anyone else. Private computer clusters have the advantage that their design can be based solely on the needs of specific users. Therefore, they can be optimised to perform only the tasks the users require, whereas a shared computer cluster with a general purpose may be inefficient or unsuitable for these specific tasks. For instance, if the use case for the computer cluster is to conduct research on the behaviour of the computer cluster when it is acted upon maliciously, a production-level cluster is unsuitable for the task because the malicious behaviour could result in abnormal behaviour and outages for the computer cluster's users.

Use cases for the private computer clusters include commercial HPC use, such as performing real-time stock trend analysis and automated trading [17], High Availability clusters [6, 18, 19], and Load Balancing clusters [19–21]. Additionally, smaller research groups can use private clusters to practise HPC use in a safe environment [22, 23]. Finally, private computer clusters can be built simply for the sake of learning and teaching cluster computing [24–26].

## 2.2 Embedded systems and Internet of Things

**Embedded systems** are computer applications with particular properties. According to Noergaard [27, Chapter 1], embedded devices possess the four characteristics listed below:

- Embedded systems are computer systems developed for a specific application.
- Embedded systems have higher quality and reliability requirements than other types of computers.
- Embedded systems have fewer hardware and/or software capabilities than personal computers.
- Many devices referred to as embedded devices are not, in fact, embedded systems.

Car engine control and brake systems, televisions, cameras, dialysis machines, cardiac monitors, routers, printers, and scanners are all embedded devices found in the automotive, consumer electronics, medical, networking, and office automation markets [27, Chapter 1]. These examples do a good job of highlighting how embedded devices are designed for specific applications, how they must meet higher reliability standards than personal computers or the consequences could be disastrous for health and/or the environment, and how embedded devices frequently have limited hardware capacities due to space and power constraints [27, Chapter 1].

Embedded systems are gaining popularity because they can be optimised to perform their designated tasks in a cost-effective, reliable, and efficient manner in terms of the computer system's size, power consumption, and performance. This is because embedded systems are designed for a specific application and, unlike personal computers, do not need to perform a variety of tasks. Consequently, personal computers are typically unsuitable for embedded device tasks.

**Internet of Things** (IoT) is a network that connects physical (embedded) devices or "things" to the internet. The IoT devices are equipped with sensors, processors, and software components that allow communication, data processing, and computing through the network with limited or no human intervention [28, 29]. The primary goal of IoT is to improve a variety of aspects of daily life by sharing, processing, and analysing data from connected devices. IoT devices, which are typically embedded systems, exhibit the same characteristics as embedded systems. Specifically, this means that IoT devices frequently have limited processing performance, power consumption, memory, and hardware functionality, among other limitations. In the past few decades, the prevalence of IoT devices has increased dramatically [28, 30].

### 2.2.1 IoT security

Unfortunately, as the popularity of IoT devices has grown, so have attacks on IoT devices. Examples of notable IoT security vulnerabilities include the rise in ransomware attacks [31], and the Mirai botnet [32], which was used to execute Distributed Denial-of-Service (DDoS) attacks using infected IoT devices. As the prevalence of IoT devices continues to rise, ensuring their security becomes increasingly crucial.

Butun et al. [33] provide exhaustive coverage of IoT security-related vulnerabilities, attacks, and countermeasures in their review. While a comprehensive security review is far beyond the scope of this thesis, it is worth noting that they provide five distinct passive attacks and 26 distinct active attacks against IoT devices based on their classification. In addition, they present seven unresolved issues pertaining to IoT security and thirteen security proposals that show promise in resolving specific cyber-security issues for IoT devices. For us, the most intriguing finding is that cryptography, in one form or another, can solve a large number of the presented security issues. However, proper implementation of a strong RNG, which is required for strong cryptography, is difficult in IoT devices due to their limited hardware resources [34]. We limit our scope to IoT security issues pertaining to IoT RNG for the remainder of this thesis.

## 2.3 Trusted Execution Environments

A Trusted Execution Environment (TEE) is a secure area of the main processor. The primary objective of TEEs is to provide an additional layer of trust for applications running inside the processing unit. This is achieved by isolating security sensitive application logic from other processes, including the OS [35, 36]. TEEs are utilised to enhance system security by providing data protection, integrity, and confidentiality, as well as by reducing the attack surface. For example, typically, when a cloud (the server or the backend) is compromised, the adversary gains access to the cloud's processes and data. TEEs provide protection against compromised infrastructure by preventing the adversary from accessing select portions of the TA, thereby protecting sensitive code and data. Remote attestation is accomplished through the use of trusted firmware. When an application is attested, the untrusted component loads the trusted component into memory: the TA is then protected against modification. Each manufacturer has its own implementation of a TEE. In this thesis, we focus on the TrustZone TEE technology developed by ARM [37].

TrustZone divides processes to *secure worlds* and *normal worlds*. Both of these worlds have their own hardware and software runtime environments. The secure world serves as the TEE and processes application logic that is security sensitive. Normal OS and standard applications operate within the normal world. Communication between the worlds happens through a separate secure monitor at the most privileged execution level (EL3). It is the only means of communication between the worlds. The basic architecture of TrustZone is presented in Figure 2.1.

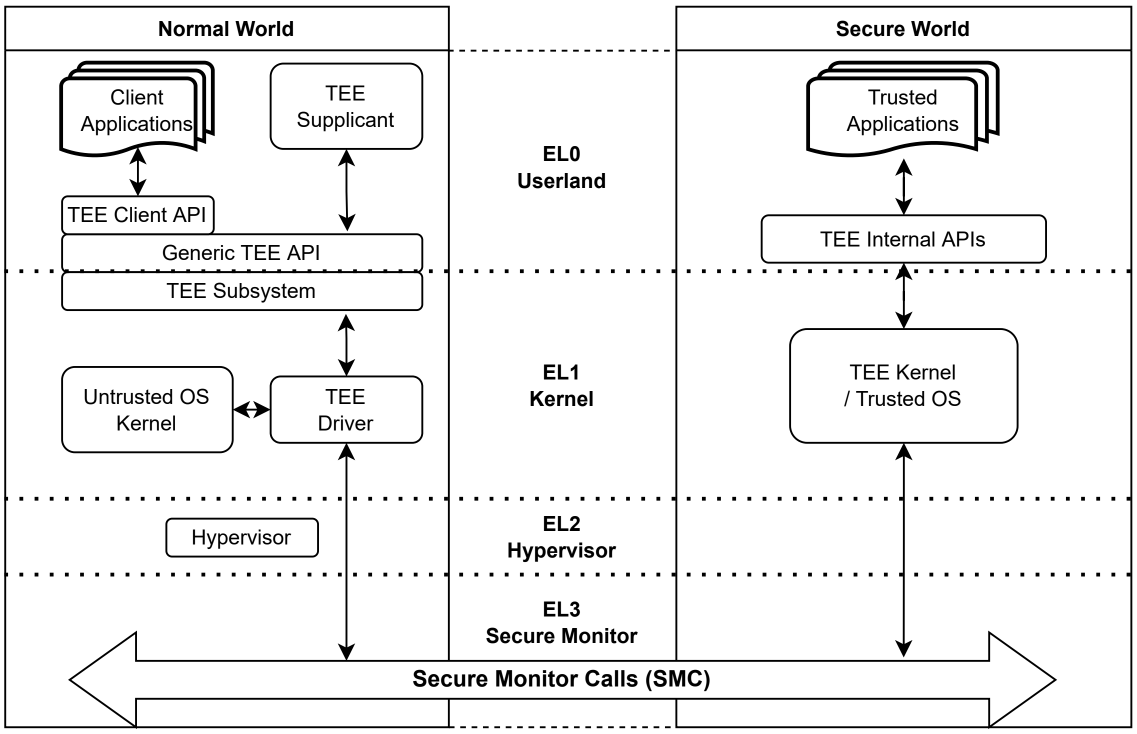

***Figure 2.1.*** *Simplified TrustZone Architecture.*

We have published a Systematisation of Knowledge paper concerning TA development for TEEs [2]. In the paper, we review how developers can use TEEs in the development of TAs and how TEEs are currently used in real-world applications. Thus, we provide only a general overview of the relevant aspects of TEEs in this thesis, while more in-depth information is available in the SoK paper.

### 2.3.1 Enarx

Enarx is an open-source middleware TEE development container. It enables the execution of existing applications within TEE instances or “Keeps” without the need to rewrite the application, implement attestation separately, or trust a large number of dependencies [38]. Enarx offers attestation, packaging, and provisioning services. In practise, this means that following the verification of attestation and the authenticity of the TEE instance, the application and associated data are encrypted and sent to the host for execution within a Keep. The application middleware interface utilised by Enarx is WebAssembly System Interface (WASI). WebAssembly enables developers to use a variety of programming languages, including Rust, C, C++, C#, Go, Java, Python, and Haskell [38].

Enarx aims to be CPU-architecture independent, meaning it can run transparently across different hardware architectures so that the developer does not need to make hardware-specific modifications [38]. Enarx currently supports Intel Software Guard Extensions (SGX) and AMD Secure Encrypted Virtualization (SEV) architectures [38]. For unsupported hardware architectures, Enarx provides a “nil” backend [39]. The Raspberry Pi 3 can be used with the nil backend. Thus, Enarx enables us to develop TEE-enabled TAs using the Magi cluster.

## 2.4 Cryptography

Cryptography is the study of techniques for secure communication. The primary objective of cryptography is to ensure message confidentiality, i.e., to conceal a message’s meaning so that only its sender and intended recipient can understand it [40]. Other important applications of cryptography include ensuring data integrity and authenticating messages. Cryptography is a subfield of the broader field of *cryptology*, which also includes the subfield of cryptanalysis. Whereas the purpose of cryptography is to create secure cryptosystems, the goal of cryptanalysis is to break them. Figure 2.2 provides an overview of the general taxonomy of cryptology.

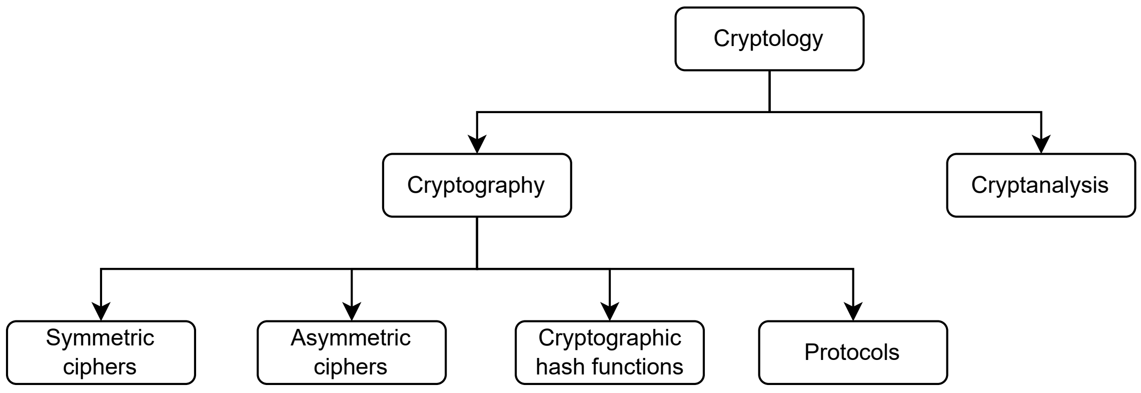

***Figure 2.2.*** *Taxonomy of cryptology. Based on the taxonomy by Paar and Pelzl [40].*

Cryptography can be further divided into the following four categories:

1. Symmetric ciphers
2. Asymmetric ciphers
3. Cryptographic hash functions

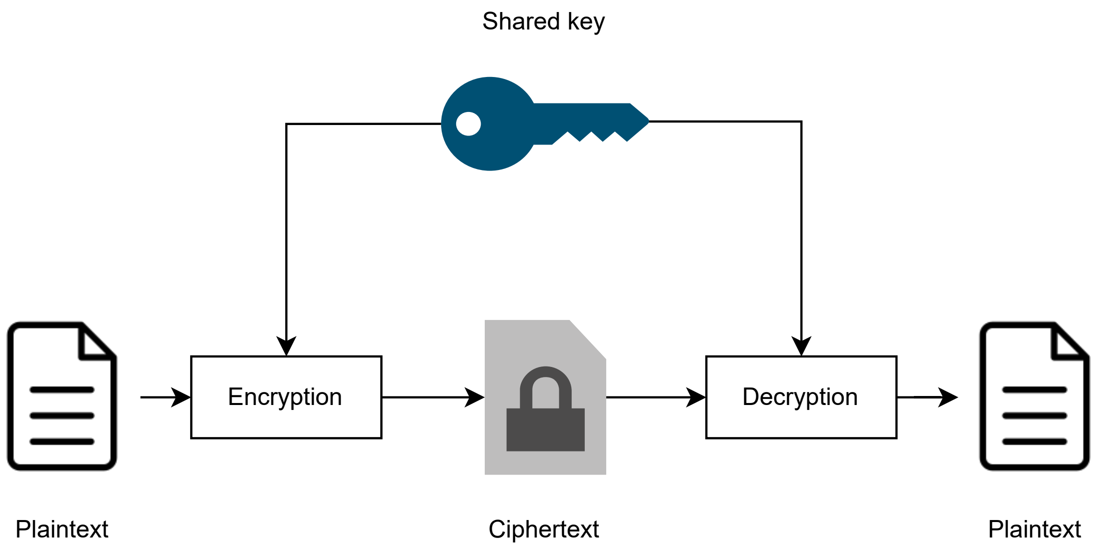

***Figure 2.3.*** *Symmetric cipher*

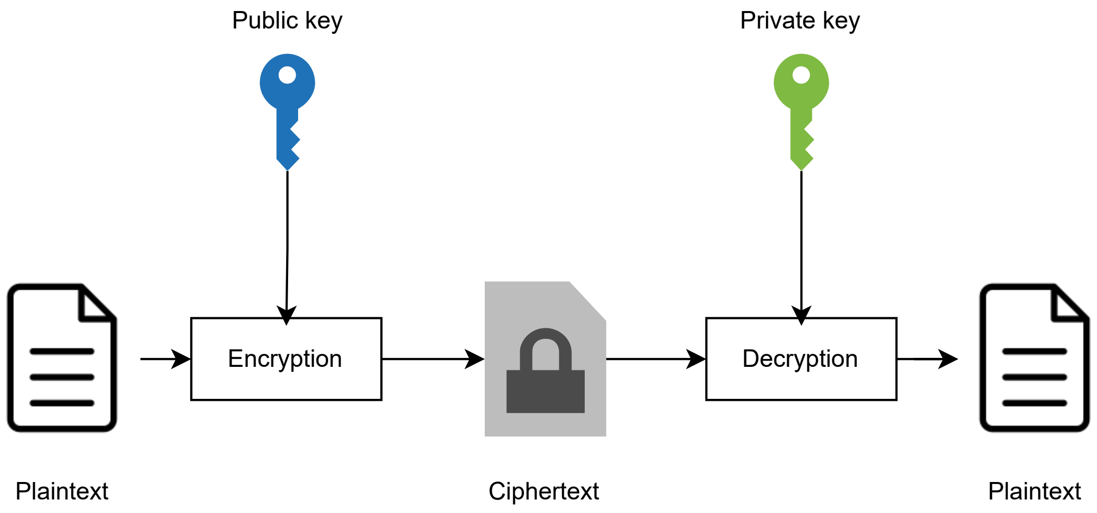

***Figure 2.4.*** *Asymmetric cipher*

4. Protocols

Both cipher types aim to *encrypt* messages, or conceal the meaning of a plaintext message by transforming it into an unreadable ciphertext. This is accomplished using *cryptographic keys*. The only way to convert the ciphertext back to the original plaintext is by *decrypting* the ciphertext by using the correct key. Thus, encryption provides confidentiality for messages. Asymmetric ciphers use different (public and private) keys for encryption and decryption, whereas symmetric ciphers use the same secret key for both encryption and decryption. Figure 2.3 and Figure 2.4 demonstrate the differences between the ciphers.

Most modern cryptographic schemes are *hybrid schemes*, consisting of both symmetric and asymmetric ciphers. Typically, asymmetric ciphers are employed for the purpose of key exchange, i.e., a shared key is agreed upon and exchanged between the communicating parties. Using this shared key, the parties then continue communicating with symmetric ciphers. Encryption and decryption can be performed significantly faster with symmetric ciphers, but creating and sharing a key for symmetric ciphers over untrusted communication channels is problematic without asymmetric ciphers [40].

Hash functions compress messages of arbitrary size into fixed-size messages known as message digests or hashes. Cryptographic hash functions are hash functions with added security characteristics. To qualify as a cryptographic hash function, a hash function must be *pre-image resistant*, *second pre-image resistant*, and *collision resistant* [40]. In practise, cryptographic hash functions are used for authentication and storage of passwords, digital signatures, blockchains, and message authentication codes, among other applications. Cryptographic hash functions therefore provide message integrity and authentication.

Cryptographic protocols can be roughly defined as a set of rules that specify how cryptographic primitives must be employed for the secure execution of practical applications. Different ciphers and hash functions can be regarded as protocol building blocks [40]. Typical protocol components include key generation, user authentication, key exchange, and message encryption using the generated key. Transport Layer Security (TLS), for instance, is a popular cryptographic protocol that provides communication security over a computer network. TLS supports a large variety of combinations of ciphers, modes of operation, key exchange algorithms, message authentication codes, and authenticated encryption (with associated data) schemes. Together, they provide data integrity, confidentiality, and authentication between the communicating applications.

### 2.4.1 Implementation failures

Cryptanalysis focuses on identifying vulnerabilities in cryptographic systems. This can be accomplished either by identifying fundamental flaws in the algorithms themselves or by identifying inconsistencies between the ideal and real-world implementations of the algorithms. In this thesis, we focus on the failures and attacks against the real-world implementations and ignore the theoretical attacks against the actual algorithms.

In modern cryptography, the algorithms are public knowledge but the cryptographic keys are considered secret [40]. Therefore, the security of a given cryptosystem depends on keeping the keys secret. There are many ways for the secret keys to leak out of system memory including defects in software implementations of cryptographic algorithms, side-channel attacks (SCAs) [41], and fault injection attacks [42]. Fault injection attacks inject faults into computations. This is possible, for instance, by adjusting the supply voltage during a computation. As demonstrated by the Bellcore attack [43, 44], it is feasible to induce incorrect calculations that leak information about the secret keys in this way. SCAs utilise unwanted leakage of sensitive information through side channels, which an adversary can exploit to gain information about the secret keys. Side channels include execution time, power consumption and electromagnetic emanations among others.

Heartbleed (CVE-2014-0160) [45], which exploits a software flaw in the OpenSSL library [46], and Meltdown (CVE-2017-5754) [47], Spectre (CVE-2017-5715 & CVE-2017-5753) [48], PortSmash (CVE-2018-5407) [49], and Foreshadow (CVE-2018-3615) [50], which all leverage microarchitectural SCAs [51], are examples of notable attacks against cryptosystems from the past decade. These examples also demonstrate that TEEs are susceptible to SCAs. Intel has even made the explicit decision to exclude SCA threats from the SGX threat model. Instead, they argue that the prevention of SCAs is the developers' responsibility [52]. Consequently, we must consider SCAs as an attack vector while constructing cryptosystems utilising TEEs.

## 2.5 Randomness and Random Number Generators

Since cryptographic protocol level communication is frequently "automatic", i.e., users do not need to specify key values to the used ciphersuites, a method is required to generate random key values. Given that the cryptographic key is a single point of failure in cryptosystems, the generated keys must be secret and unpredictable in order to ensure the security of the cryptosystems. There are also other cryptographic uses for random numbers, such as initialisation vectors and nonces (numbers used only once) [53]. In this section, we examine randomness and Random Number Generators (RNG) in computer systems to determine how to implement a secure RNG.

### 2.5.1 Defining randomness in computer systems

Randomness is a peculiar quality in that it is defined more by what it is not than by what it is. If an occurrence has a pattern, it is not random. Similarly, if we can predict what will happen next with a higher probability than pure chance, the event is not random. Nonetheless, we can define certain characteristics that randomness must possess.

Since traditional computers operate on binary values, randomness in computer systems refers to sequences of random bits. A perfect random bit stream is analogous to a sequence of fair coin flips in which one side displays the value "1" and the other side displays the value "0". Assuming the coin cannot land on its edge, $p = \frac{1}{2}$ represents the probability of each side landing face up. By repeating the flips indefinitely, a random sequence of zeros and ones emerges in which both values appear roughly equally frequently.

Coin flip sequence with a fair coin follows a *uniform distribution*, meaning that all outcomes are equally likely to occur. In contrast, when all outcomes are not equally likely to occur, the distribution is called *nonuniform*. For example, a toss of a weighted coin could yield the value “1” with a probability of $p = \frac{2}{3}$ and value “0” with a probability of $p = \frac{1}{3}$. This coin toss event would therefore be *biased* towards the value “1” and be nonuniformly distributed. A perfect random bit sequence must be unbiased and, consequently, uniformly distributed. A 128-bit key generated by a perfect Random Bit Generator (RBG), for instance, must have the same probability of $p = \frac{1}{2^{128}}$ for each of the $2^{128}$ possible key values.

In addition to being unbiased, the randomness used in cryptographic applications must be *unpredictable*. Unpredictability includes both forward unpredictability and backwards unpredictability. Forward unpredictability implies that even with knowledge of the RBG’s previous values, it should be impossible to predict future values. Backward unpredictability implies that it should be impossible to retrieve the *seed* value, even if the RGB output is known. Seeds are explained in greater detail in Subsection 2.5.2.

### 2.5.2 RNGs and PRNGs

There are two distinct types of random bit sequence generators: RNGs and Pseudorandom Number Generators (PRNGs). While the objective of both is to generate random sequences of ones and zeros, the means by which they accomplish this objective are distinct. Typically, cryptographic systems employ both RNGs and PRNGs [53], as demonstrated by Figure 2.5.

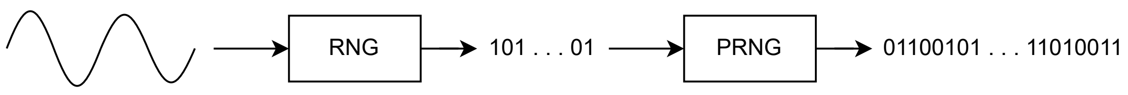

***Figure 2.5.*** *RNGs produce few unreliable bits from analogue entropy sources, while PRNGs expand those bits to a longer stream of reliable bits. Based on the Figure 2-1 in “*Serious Cryptography: A Practical Introduction to Modern Encryption*” [53, Chapter 2].*

**Random number generators** are nondeterministic generators that use analogue sources, also referred to as *entropy sources*, to convert entropy from the real world into digital bits. To accomplish this, RNGs require two components: a measurement to collect data from the entropy source and a distillation process that compensates for any flaws in the entropy source that could result in the production of obvious nonrandom bits (e.g. a long string of zeros or ones) [54]. RNGs are sometimes referred to as *True* Random Number Generators (TRNGs) to emphasise the fact that their randomness is intended to be nondeterministic, in contrast to PRNGs. RNGs collect entropy by observing expectedly random physical phenomena. Noise in an electrical circuit, the user’s mouse movements, quantum effects in a semiconductor, and radioactive decay are examples of entropy sources [53, 54]. RNGs may utilise various combinations of entropy sources.

Based on our current knowledge of physics and quantum mechanics, quantum phenomena-based entropy sources are guaranteed to be truly random [53, 55]. Consequently, utilising RNGs based on quantum phenomena (also known as Quantum Random Number Generators or QRNGs) is a common method of implementing TRNGs [56–58]. Nevertheless, even QRNGs can be susceptible to bias due to flaws in the entropy distillation procedure [59, 60]. QRNGs are also not the only TRNG sources. Jitter from ring oscillators, for instance, can be used as an entropy source for TRNGs [61–66]. This is a common method of implementing TRNGs in resource constrained embedded devices.

**Pseudorandom number generators**, on the other hand, are deterministic generators that transform their input bit stream, also known as the *seed*, into a longer and uniformly distributed bit stream. The same seed value will always result in the same output. Whereas RNGs use entropy sources as their input, PRNGs obtain their input seed values from an *entropy pool*.

PRNGs consist of three distinct functions that modify the PRNG's internal state [53]:

1. **init()**, which initialises the PRNG's entropy pool and internal state.
2. **refresh()**, which updates the entropy pool with a seed value typically derived from an RNG.
3. **next()**, which returns specific amounts of pseudorandom bits and updates the entropy pool.

The init() operation is used to initialise the state and entropy pool to their default values and prepare the PRNG to execute the refresh() and next() operations [53]. The next() operation invokes a Deterministic Random Bit Generator (DRBG) algorithm, which expands a subset of the entropy pool's bits into a significantly longer output. It also modifies the entropy pool to guarantee that the DRBG never receives identical inputs twice (which would result in the same output) [53]. The refresh() operation is typically requested by the operating system, while the next() operation is typically requested by applications [53].

There are both software and hardware PRNGs available. RDRAND in Intel microprocessors, which is based on NIST SP 800-90A [67], is an example of a hardware PRNG, while Mersenne Twister [68] is an example of a software PRNG.

Ironically, due to the use of cryptographic algorithms that introduce additional randomness properties with each transformation in the DRBG algorithm, PRNGs often generate statistically superior and faster outputs than TRNGs [54]. Thus, PRNGs are frequently chosen over pure TRNGs in applications requiring randomness.

### 2.5.3 Entropy source pitfalls and considerations

It may be possible to manipulate some entropy sources to produce biased output if the entropy distillation procedure is poorly designed. For instance, if the only source of entropy is the user's mouse movement, an attacker can simply stop moving the mouse. This halts the rate of entropy production by the entropy source. Many RNGs employ blocking mechanisms to prevent their use until sufficient entropy has been accumulated to generate good randomness. A well-known example is Linux's $/dev/random$ [69]. In environments with low entropy, the blocking mechanism can also cause issues, as the execution of a program can be silently blocked for an extended period of time without the user being aware of the reason. Thus, unblocking implementations that output data regardless of the quality of the entropy are commonly used.

Unintented operational conditions can also cause biased results for TRNGs. For instance, ring oscillators must operate under particular conditions to generate appropriate randomness for TRNGs. If the operating temperature or voltage deviates from the ideal conditions, the TRNG output cannot be relied upon [70, 71]. As a result, there are numerous active and passive attacks against ring oscillators that result in data bias [72, 73]. It is extremely difficult to determine whether data from an entropy source is truly random or merely *appears* random.

PRNGs must combine multiple entropy sources to mitigate the effects of biased entropy sources. This is also essential for reducing the attack vector of a device employing PRNGs. Even if the NSA has designed a backdoor in a hardware RNG component, a Chinese manufacturer has hacked it during manufacturing, a Russian hacker has tampered with the network drivers, and all other sources of randomness have been tampered with by various state-level actors, by mixing all these untrusted sources of randomness together (e.g. by using the XOR operation), none of them should have any insight into the random numbers generated by the upstream system such as $/dev/urandom$ [74] in Linux.

### 2.5.4 Cryptographically secure PRNGs

Cryptographically Secure PRNGs (CSPRNGs) are PRNGs with additional security requirements. According to the taxonomy by Kelsey et al. [75], CSPRNGs must be resistant to three types of attacks: *state compromise extension attacks*, *direct cryptanalytic attacks*, and *input-based attacks*. Forward secrecy requires the PRNG to withstand state compromise extension attacks, whereas backwards secrecy requires the PRNG to withstand both direct cryptanalytic attacks and input-based attacks. If a PRNG cannot withstand all three of these attacks, it cannot be considered cryptographically secure.

CSPRNGs require a method for generating truly random seed values [53]. Consequently, cryptographic applications commonly rely on TRNGs to provide the entropy pool for CSPRNGs. Due to the potential for a low entropy rate, the direct output of TRNGs is unsuitable for use in cryptographic applications, as it would either block or cause predictable output. Because they transform even a small amount of true entropy from TRNGs into longer and uniformly distributed pseudorandom bitstreams, CSPRNGs generate cryptographically usable output without blocking.

PRNGs that are not cryptographically secure, but still generate a uniformly distributed output without statistical bias, are also widely used in real-world applications due to their speed and ease of implementation [53]. In fact, most of the PRNGs exposed to programmers are non-cryptographic PRNGs [53]. Examples include Python's Random module and libc's rand() function, which are both based on Mersenne Twister [68].

Inappropriate use of PRNGs can lead to implementation failures in upstream cryptographic applications. This often involves using PRNGs that are not cryptographically secure in cryptosystems when CSPRNGs should be used instead. The Debian OpenSSL random number bug (CVE-2008-0166) and factorisable (weak) RSA implementations [76] discovered by Nemec et al. [77] in 2017 (CVE-2017-15361) are good examples of such implementation errors, but there are numerous other examples, such as CVE-2009-3278, CVE-2009-3238, and CVE-2009-2367.

The vast majority of commonly used CSPRNGs are based on hash functions, stream ciphers, or number-theory problems [78]. Typical contemporary CSPRNGs include ISAAC [79], Yarrow [80], Fortuna [81, Chapter 9], and ChaCha20 [82].

### 2.5.5 Statistical test suites

Random number generators can be evaluated using statistical test suites that seek to detect nonrandom behaviour. Dieharder [83], TestU01 (BigCrush) [84], and NIST's Statistical Test Suite (STS; described in SP800-22 [54]) are examples of such test suites. Their objective is to ensure that the output of the RNG is uniformly distributed and free of statistical bias. STS is developed specifically for testing CSPRNGs, while Dieharder is designed for testing all PRNGs, and TestU01 is more applicable for testing TRNGs [85]. Nonetheless, each of the test suites can be utilised to test both PRNGs and TRNGs. All of the tests included in a test suite should be statistically *independent*, and there should be a sufficient number of statistical tests for adequate *coverage* of testing nonrandomness [86].

A PRNG may not be cryptographically secure despite its exceptional statistical properties. This type of PRNG is exemplified by the Mersenne Twister [68], which passes STS [78] despite being insecure for use in cryptographic applications. It is possible to predict all future values of the generator with sufficient iterations (624 for MT19937 [78]), despite its evenly distributed output and absence of statistical bias. Therefore, it fails to meet the requirement for forward unpredictability and is unsuitable for use in cryptography. Thus, passing STS *does not* guarantee the PRNG's cryptographic security. To be suitable for cryptographic applications, a CSPRNG *must*, however, pass this test suite

[78]. Consequently, the STS should only be used as a starting point when estimating the cryptographic security of a particular PRNG.

As STS is computationally intensive, embedded devices with limited resources cannot utilise it. However, IoT devices, which are often embedded devices with limited resources, must also test the randomness of their TRNG. To address this issue, Lee et al. [87] published NIST-Lite, a streamlined version of NIST's STS that drastically reduces its energy consumption. Therefore, NIST-Lite would be appropriate for testing the entropy produced by our EaaS simulation.

Hurley-Smith and Hernandez-Castro [85] provide a critical analysis of the problems associated with statistical test suites. Many RNGs deemed secure by specific test suites under specific conditions may not be secure in reality, according to their findings. A small sample size and a small number of tested devices, for instance, can lead to misleading results and incorrect conclusions. They argue that a simple pass/fail result from a single test suite is insufficient to deem a particular device with a particular RNG *suitably random* for a given application. Rather, a more comprehensive process is required to evaluate risks, attack vectors, and countermeasures, taking into account any known biases. Additionally, RNGs should be tested with a sufficiently large sample size across a sufficient number of statistically independent tests. In addition, they find that well-known and widely-used test suites such as NIST's STS, Dieharder, and TestU01 exhibit some degree of correlation in some of their tests, and that STS and Dieharder have a high degree of duplication in their test selection. Efforts should be made to ensure that statistical test suites do not include correlated tests. In conclusion, they assert that the current RNG tests are useful for detecting blatant deviations from randomness, but fail to provide useful information in more subtle cases. This is why simple pass/fail or random/nonrandom results should be avoided. Even further, Aumasson [53] asserts that statistical tests have little bearing on cryptographic security.

### 2.5.6 IoT RNG

Embedded IoT devices, like all other modern internet-connected devices, require both general-purpose and cryptographically secure randomness [34]. However, the limited hardware capabilities of many IoT devices prevent them from generating high-quality randomness. These resource-constrained IoT devices lack components suitable for proper TRNG distillation and have limited computational power, energy resources, and hardware protection features, making randomness generation difficult [34]. The DEF CON 29 talk “You're Doing IoT RNG” by Petro and Cecil [88] does an excellent job of presenting the vulnerabilities with IoT RNGs and describing how they can be exploited. Based on this talk, we present the most significant security issues with IoT RNGs, which are:

- IoT devices typically lack access to CSPRNG subsystems due to their lack of modern operating systems.

- Developers frequently make incorrect calls to the hardware TRNG (HWRNG) component.
- Handling HWRNG errors correctly is difficult, if not impossible, at times.
- Oftentimes, the documentation for bare-metal function calls is inadequate, which makes their proper use challenging.
- Numerous IoT devices generate HWRNG of subpar quality. This may either be due to their resource-constrained nature or a flaw in their entropy distillation procedure.

IoT devices typically employ bare-metal environments or simple IoT operating systems. In a bare-metal environment, CSPRNG subsystems are absent entirely. The majority of IoT operating systems also lack proper CSPRNG implementations [34]. According to Kietzmann et al. [34], mbed OS [89], and Zephyr [90] are the only IoT operating systems with a proper CSPRNG implementation and an OS-level random API. For instance, Contiki-NG [91] lacks a CSPRNG implementation and instead uses the rand() function from libc for all PRNG calls [34, 88].

When an IoT device requires a random number, it calls the dedicated HWRNG via the device's SDK or IoT operating system. Calls to HWRNG are handled through a hardware abstraction layer (HAL). Incorrect HWRNG calls result in errors, which may return faulty states or provide data that is not truly random. This results in "random numbers" that are either only partial entropy, the number "0", or uninitialised memory [88].

HALs frequently only permit developers to handle errors by aborting the entire process that called the function or by leaving the process in a blocking state [88]. Neither of these are generally acceptable solutions. Therefore, developers often disregard the error conditions, resulting in biased and low-entropy randomness in upstream applications [88].

A proper implementation of CSPRNG would be crucial, as it would solve the issue of HAL functions that either block execution of the program or fail. Nevertheless, many CSPRNG algorithms are computationally too intensive to run on resource-constrained IoT devices, which lack the computational power to run these algorithms effectively [34]. Similarly, many CSPRNG algorithms consume a significant amount of energy, and their use in energy-constrained IoT devices could enable denial-of-service attacks [34]. These impose additional constraints on the design of potential CSPRNG subsystems for IoT devices.

In addition to the aforementioned issues, IoT devices are susceptible to generating weak entropy during the boot process, even if they employ an OS that supports a proper CSPRNG subsystem. Embedded devices frequently lack the hardware components, such as the hard drive, keyboard, and mouse, that are present on desktop computers. These are used as entropy sources by a number of CSPRNG subsystems. When the CSPRNG algorithm begins with an empty entropy pool (either with no seed or a hardcoded initial seed), its entropy sources will require time to generate sufficient randomness. The fewer entropy sources that are available, the longer the process will take. Worse still, the problem is exacerbated by the low computing power of IoT devices.

If the CSPRNG subsystem's next() function is called before sufficient randomness has been generated, the output will be biased and predictable. This results in vulnerabilities in upstream applications, as demonstrated in 2012 by Heninger et al. [92], who scanned the entire internet for 12.8 million TLS and 23 million SSH servers and found that 5% of all TLS hosts and 10% of all SSH hosts shared the same RSA keys or factors. This enabled them to retrieve the private keys of 0.5% of TLS hosts and 1% of SSH hosts. Nearly all of the vulnerable hosts were embedded devices that suffered from lack of entropy in the early boot-time process when the keys were generated. In their simulation, Vassilev and Staples [93] demonstrated that Linux's $/dev/urandom$ [74] CSPRNG subsystem can take up to 45 seconds to reach the bare-minimum threshold of 112 bits of random data in environments with limited entropy sources, invalidating all randomness generated prior to this point.

## 2.6 Randomness Beacons and Entropy as a Service

As expanded in Subsection 2.5.6, IoT devices with limited resources struggle to generate the high-quality entropy required for robust randomness. Since IoT devices are connected to the internet, this issue can be resolved by employing services that broadcast strong entropy across the network. The concept is that a third-party service broadcasts high-quality entropy to IoT clients who cannot generate it in sufficient quantity or quality. This can be accomplished cryptographically securely using either *Randomness Beacons* or purpose-built *Entropy as a Service* (EaaS) servers. The related terminology is explained in Subsection 2.6.1.

### 2.6.1 Terminology

**Randomness beacons** broadcast cryptographically signed and timestamped random numbers at regular intervals [94, 95]. They utilise high-entropy sources to extract TRNG from unpredictable natural phenomena. If multiple entropy sources are used, the randomness beacon combines the entropy from multiple TRNG sources into random data blocks by, for example, XORing them. Random blocks of data are published on the internet alongside timing information and a cryptographic signature as well as additional metadata. Each published block of random data is referred to as a *pulse*. The sequence of all pulses is known as a *chain*. Randomness beacons may be either *centralised* or *decentralised*.

**Centralised Randomness Beacons** (CRBs) utilise Trusted Third Parties (TTP) to collect entropy from entropy sources, combine them together, and then publish pulses containing blocks of randomness to the network. The TTP is an entity that is trusted to behave properly. By colluding, the TTP can violate the security properties of randomness beacons, since the beacon operator is in a position to reveal random numbers in advance to favoured parties, manipulate the public random numbers, or even rewrite the history of the chain by lying about the previous pulses [94]. Thus in CRBs, the TTP serves as a single point of trust.

**Decentralised Randomness Beacons** (DRBs) do not rely on TTPs. Multiple sources instead combine the data from their respective entropy sources into a single published pulse. Thus, the single-point-of-trust problem is eliminated. The revelation of the backdoor in the NSA-designed Dual EC PRNG [96, 97] eroded faith in centralised single-point-of-failure systems. This is what inspired the concept of DRBs [95].

Interactive and noninteractive DRBs can be distinguished [95]. Interactive DRB protocols generate a beacon output through multiple rounds of participant communication. Noninteractive DRB protocols do not require participant interaction to generate a pulse for each round. Consequently, noninteractive DRBs are preferred for decentralised applications.

DRBs have the security properties of *Availability (or Liveness)*, *Bias Resistance*, *Unpredictability*, and *Public Verifiability* [95]. Unpredictability means that an adversary cannot predict future beacon outcomes. Bias-Resistance property means that a single participant or a colluding adversary cannot bias the future beacon values. Availability means that a single participant or a colluding adversary cannot prevent the generation of new beacon values. Public Verifiability indicates that any third party can verify the validity of the new pulse values. The SoK by Raikwar and Gligoroski [95] provides additional information regarding the use cases of DRBs and how they can be constructed.

An **Entropy as a Service** (EaaS) server, in contrast to the randomness beacon, only distributes entropy to clients that specifically request it. Consequently, EaaS generates considerably less data than randomness beacons. Otherwise, the concept remains largely unchanged. EaaS server also uses timestamps and digital signatures to harden the protocol against response-replay attacks, Man-in-the-Middle (MitM) attacks, and Domain Name System (DNS) poisoning attacks [93].

EaaS is designed from the ground up to provide robust entropy to IoT devices [93], enabling them to generate strong cryptographic parameters, whereas randomness beacons are designed for multiple additional use cases [95]. If the IoT client has a CSPRNG subsystem, it is seeded with entropy from both the EaaS service and the clients own HWRNG sources. Otherwise, clients simply use their own HWRNG sources and combine their entropy with the incoming entropy from the EaaS server (with e.g. XOR) prior to using it for upstream applications. Similarly to randomness beacons, EaaS can be implemented with a centralised or decentralised architecture.

To transfer data between the server and IoT clients, EaaS employs application layer protocols such as HTTP. In NIST's EaaS implementation [98], for instance, the client requests entropy via an HTTP GET request that includes the client's public key and the requested amount of entropy [93]. The server then sends the requested data to the client. The data is encrypted using the public key provided by the client. To obtain the initial strong keys for the IoT clients, it is assumed that the IoT device's manufacturer has generated and provisioned a strong key. This model of key provisioning is frequently used in practise for the shipment of IoT devices including secure hardware components such as the Trusted Platform Module (TPM) or ARM TrustZone [93].

### 2.6.2 History and Related works

Using someone else's randomness as a service is not a novel concept. Randomness beacons were first described by Rabin [99] in 1983. Originally Rabin proposed the randomness beacons to be used for digital signature schemes and implementing confidential disclosures. Nowadays, there are many real-world implementations of randomness beacons. NIST launched their own Randomness Beacon initiative [100] in 2011. 2013 saw the release of the first public prototype of the NIST beacon (version 1.0), and in 2018 NIST upgraded their randomness beacon to version 2.0. The final version of the beacon is expected to be published by NIST in 2022. The NIST randomness beacon employs a QRNG source that passed the Bell test [101].

In the past five years, there has been an increase in scientific interest and publications concerning DRBs. Examples of DRBs (in order of publication) include Ourobros [102], RandHerd and RandHound [103], SCRAPE [104], HydRand [105], Albatross [106], RandPiper [107], and SPURT [108]. Additionally, *League of Entropy* runs a DRB called *drand* [109]. League of Entropy is a consortium consisting of different global organisations and individual contributors. Each contributor operates their own unique source of high entropy that they provide to drand.

In addition to the randomness beacon project, NIST launched an EaaS project [98] in 2016. Their EaaS implementation also generates randomness from QRNG sources that passes the Bell test [101]. However, the project appears to be on hold as the EaaS project page was last updated in 2020 and the EaaS-Events page has not been updated since 2017. Unlike the randomness beacon project, there is no related comprehensive documentation (in the form of NISTIR or NIST SP drafts). The only other relevant EaaS academic publication appears to be "Entropy as a Service: A Lightweight Random Number Generator for Decentralized IoT Applications" by Ullah et al. [110], released in 2020. Additionally, there are active EaaS projects in the industry. Crypto4A's Root of QAOS [111], QNu Labs' QOSMOS [112], Qrypt's EaaS [113], Quantropi's SEQUR [114], and

QuintessenceLabs's qRand [115] are recent examples of EaaS services provided by the industry.

There are also additional sources for public randomness besides randomness beacons and EaaS servers. For instance, random.org [116] has been providing random numbers via the internet as a service since 1998. Random.org is a service that generates random numbers using atmospheric noise and then offers applications such as list randomiser, password randomiser, and lottery quick pick based on these numbers. It differs from randomness beacons and EaaS servers in that random data is not accompanied by a cryptographic signature or timestamp as metadata.

# 3. IMPLEMENTING AN EMBEDDED COMPUTER CLUSTER

As demonstrated in Section 2.1, computer clusters can have a variety of requirements and use cases. In this chapter, we describe the use cases, requirements, and available resources that must be understood before designing and implementing the Magi cluster to meet our specific needs. First, we present the existing related works in Section 3.1 to get an understanding of the related and already implemented clusters. We explore both shared production-level clusters and clusters designed for specific use cases. In Section 3.2, we then discuss the requirements for the Magi cluster in more detail. Finally, we present our design for the Magi cluster, which is based on the related works and the requirements. The design overview is detailed in Section 3.3, while Section 3.4 and Section 3.5 focus on the design choices related to the nodes and the server, respectively.

## 3.1 Related works

We begin our search for relevant computer clusters in Finland by examining the shared production-level clusters that are currently accessible for research purposes. Locally, Tampere University has a shared production-level computer cluster called Narvi [117]. The x86-based Narvi cluster is intended for use by researchers, faculty members, and students for their HPC needs. It consists of 140 CPU-only nodes with 3000+ CPU cores and 22 GPU nodes with 4 GPUs each, and Slurm Workload Manager is used for job scheduling [117]. Several other universities in Finland also have their own computer clusters for HPC research. For instance, Aalto University has the Triton cluster [118] and Helsinki University has the Turso cluster [119]. They share a striking resemblance to the Narvi cluster in their design. The primary difference between the clusters is the amount of computing power they offer.

CSC, the Finnish IT centre for science, also has numerous computer clusters that Finnish researchers can utilise. Among these are the current supercomputers LUMI [120], Puhti [121], and Mahti [122]. The most recent of them, LUMI, is one of the world's most powerful supercomputers, according to TOP500 [5]. For HPC applications, CSC also provides cloud services Pouta [123] and Rahti [124], and research data storage Allas [125]. Incorporating increasingly more computing power, offering cloud-based Infrastructure as a

Service (IaaS) or Platform as a Service (PaaS) for the users, and combining the computer clusters with data management services suitable for the large amounts of data related to HPC research demonstrate how computer clusters can be scaled well beyond our needs and requirements for the Magi cluster. While these computer clusters and services are beyond the scope of this thesis, we introduce them to establish how the requirement elicitation must drive the design of the computer cluster and to illustrate the Finnish HPC research infrastructure at the cutting edge of the field.

All of the previously mentioned clusters are x86-based. Nonetheless, ARM-based clusters exist as well. Embedded computer clusters based on the ARM architecture are of particular interest to us, as the Magi cluster is also an embedded computer cluster. A 2015 survey by Jahanzeb et al. [126] revealed that ARM-based clusters have a better performance-to-power ratio than x86-based clusters, but that at the time, x86-based clusters outperformed ARM-based clusters for typical HPC use cases. A more recent survey by Yokoyama et al. [127] from 2019 demonstrates that ARM architecture may be a better fit for modern HPC systems employing co-processors, and that companies such as Amazon already offer ARM-based computer instances for cloud computing. Moreover, they argue that ARM HPC could become dominant in the future due to its matured software ecosystem, more aggressive architectural improvements, and cheaper prices when compared to x86.

The most recent ARM based embedded clusters include computer clusters based on the ARM ThunderX2 processor family, such as the experimental infrastructure for the Mont-Blanc 2020 project [128]. Mantovani et al. [129] analyse the performance and energy consumption of the aforementioned cluster and conclude that ThunderX2 provides comparable or superior performance-to-power ratio and scalability to x86 processors with a comparable software ecosystem availability. Therefore, ARM-based clusters appear to be viable candidates for the next generation of HPC systems and supercomputers [129]. Pardos et al. [130] also present a study on the performance of ThunderX2-based HPC clusters, concluding that ThunderX2 is capable of matching the performance of its x86 counterparts, but that its power efficiency still lags behind x86 when AVX512 extensions are used. All of these surveys on ARM-based clusters demonstrate that energy-efficient ARM-based clusters have the potential to replace power-hungry x86-based clusters in exascale HPC in the future, and there are already indications that this change is occurring.

We have only presented shared production-level clusters thus far. However, for a variety of reasons, many NISEC research projects are not suitable for shared production-level clusters. For instance, root access may be required to enable insecure hardware debug mechanisms, such as performance monitor units, which should never be exposed to user space on servers used for production. Some of our research necessitates malicious behaviour towards the computer cluster, the effects of which are studied at multiple levels of abstraction, ranging from microarchitectural to networking. Consequently, we are interested in information regarding existing private computer clusters designed for specific use cases.

In the scientific literature, Raspberry Pi (RPi)-based clusters are by far the most prevalent private embedded computer clusters. Typical use cases include the teaching and learning of distributed computing concepts [24–26], as well as the construction of cost-effective and energy-efficient clusters for small HPC applications that do not require a great deal of computing power, such as image processing and big data analysis [131–136].

According to studies by Mappuji et al. [137], Cicirello [138], and N. Gupta et al. [139], RPi clusters are not particularly effective for HPC use. Due to the limited memory bandwidth of the RPi's, the performance of RPi's appears to peak at two or three cores, meaning they are unable to properly utilise all four cores. In addition, the limited networking capability of RPi's severely limits the parallel computing capabilities of the clusters, rendering them unsuitable for tasks requiring the distribution of a significant amount of data between the nodes relative to computation time. The studies reveal, however, that RPi clusters can deliver modest performance at a reasonable cost and with a reasonable energy footprint.

Intriguingly, the study by Hawthorne et al. [140] reveals that RPi clusters appear to offer comparable encryption/decryption performance for storage encryption tasks while consuming less power than x86-based systems when employing the Twofish algorithm. As the AES instruction set is optional on ARMv8 processors and is not implemented in any single-board computers known to Hawthorne et al. [140], x86-based clusters continue to outperform RPi clusters when executing the AES algorithm. If AES hardware instructions were introduced for RPi, ARM-based RPi clusters could be preferable to x86-based clusters in the future for storage encryption tasks.

## 3.2 Requirements

The initial phase of the implementation of the computing cluster is the elicitation of requirements. Those accountable for implementing the computing cluster must learn the cluster's requirements from the cluster's stakeholders. The stakeholders may have various functional and nonfunctional requirements for the cluster, including performance, security, architectural, and usability requirements. The design must meet all of these specifications. We will now describe how the requirements for the Magi cluster are gathered and what those requirements are.

The elicitation of requirements for the Magi cluster entailed identifying possible use cases for the computer cluster and then gathering requirements based on these use cases. Since the Magi cluster is only utilised by the NISEC group for our own internal research, all of the collected use cases are based on the NISEC's existing and potential future research projects. The stakeholders of the Magi cluster are the NISEC group leaders and all group members who conduct research using the cluster. The use cases for the Magi cluster include:

**ARM-based software testing.** As ARM-based software is used and researched in many NISEC projects, NISEC requires an ARM-based computer cluster to efficiently run and test software built for ARM processors. One example use case is fuzzing ARM-based cryptographic software in order to find bugs. The existing computer cluster at Tampere University, known as the Narvi [117] cluster, is based on the x86 architecture, making it incompatible with ARM-based software. Therefore, an ARM-based architecture is required for the Magi cluster.

**Distributed network security testbed for embedded devices.** NISEC requires a testbed that can measure network throughput when a core is saturated. In addition, the testbed can be used to investigate, for instance, DDoS vulnerabilities, attack effects, and mitigation techniques. Consequently, the Magi cluster requires multiple embedded network-connected devices as nodes.

**Testing the embedded performance of security-oriented protocols.** Members of the NISEC group conduct extensive research and contribute to the development of security-focused protocols such as TLS and OpenSSL. On embedded devices with limited computational capabilities, protocol performance is crucial. Therefore, the Magi cluster must be able to test the performance of experimental cipher suites on embedded devices.

**Testing applications of distributed hardware-assisted security technologies.** Using TEEs in a distributed manner, as we do in our EaaS simulation, is a prime example of this use case. NISEC has already conducted research on Intel SGX and AMD SEV-enabled cloud technologies; therefore, other potential use cases include simulating TEE-enabled cloud technologies using TrustZone TEE in ARM processors.

**Simulating attacks against consensus in distributed systems.** Many distributed network applications are vulnerable to majority attacks, where an adversary gains control of the majority of the participating nodes of the network. Majority attacks against blockchains are already a quite well understood and researched phenomena [141], but similar attacks also exist, for example, in Tor network, where a well-funded adversary, such as a nation-state actor, can achieve consensus blocking against Directory Authorities [142]. With the Magi cluster, we can simulate such attacks against consensus in distributed systems.

Based on the aforementioned use cases, the Magi cluster nodes must be embedded devices with ARM-based processors that support TrustZone. Additionally, a large number of nodes must exist to permit the testing and simulation of various distributed applications.

In addition to the requirements extracted from the use cases, the stakeholders have numerous other requirements regarding the usability, security, and software licence for the cluster's software. The cluster must primarily be as user friendly as possible. This includes a straightforward method for controlling all nodes with the server without requiring individual configuration of the nodes, easily distinguishable nodes, and the capacity for future expansion. Additionally, the cluster must be accessible from the external network via the server, but individual nodes must not be accessible from the external network. Therefore, the server must be capable of providing the nodes with firewalling and Network Address Translation (NAT). Additionally, the cluster must be able to withstand high levels of stress without crashing; this has been an issue with some previous NISEC projects. Finally, FOSS should be utilised.

In the end, we end up with the following list of requirements for the Magi cluster:

- The cluster must be an embedded computer cluster based on the ARM architecture.
- The cluster must be easy to manage. Specifically, the cluster must have a server capable of controlling all nodes via a job scheduler.
- The cluster shall be easily extendable in the future.
- The nodes must be able to communicate with each other.
- The server must be accessible from the outside network.
- The nodes must not be accessible from the outside network.
- The server must have a firewall capable of providing NAT for the nodes.
- The nodes must be distinguishable from each other.
- The nodes must function without the need for individual configuration. Specifically, the cluster must support Preboot eXecution Environment (PXE) booting of the nodes.
- Both the server and the nodes must utilise FOSS.
- The nodes must be able to withstand high levels of stress without crashing.

The design of the Magi cluster is based on these requirements.

## 3.3 Design overview

The design of the Magi cluster is based on the requirements gathered during the phase of requirement elicitation. RPis are utilised as nodes since they satisfy our hardware requirements. The RPi nodes run a Devuan ASCII Linux distribution. For the server, we install FreeBSD on an HPE ProLiant bare-metal system. The FreeBSD server provides all required server utilities, such as traffic routing, firewalling, NAT, PXE booting of the

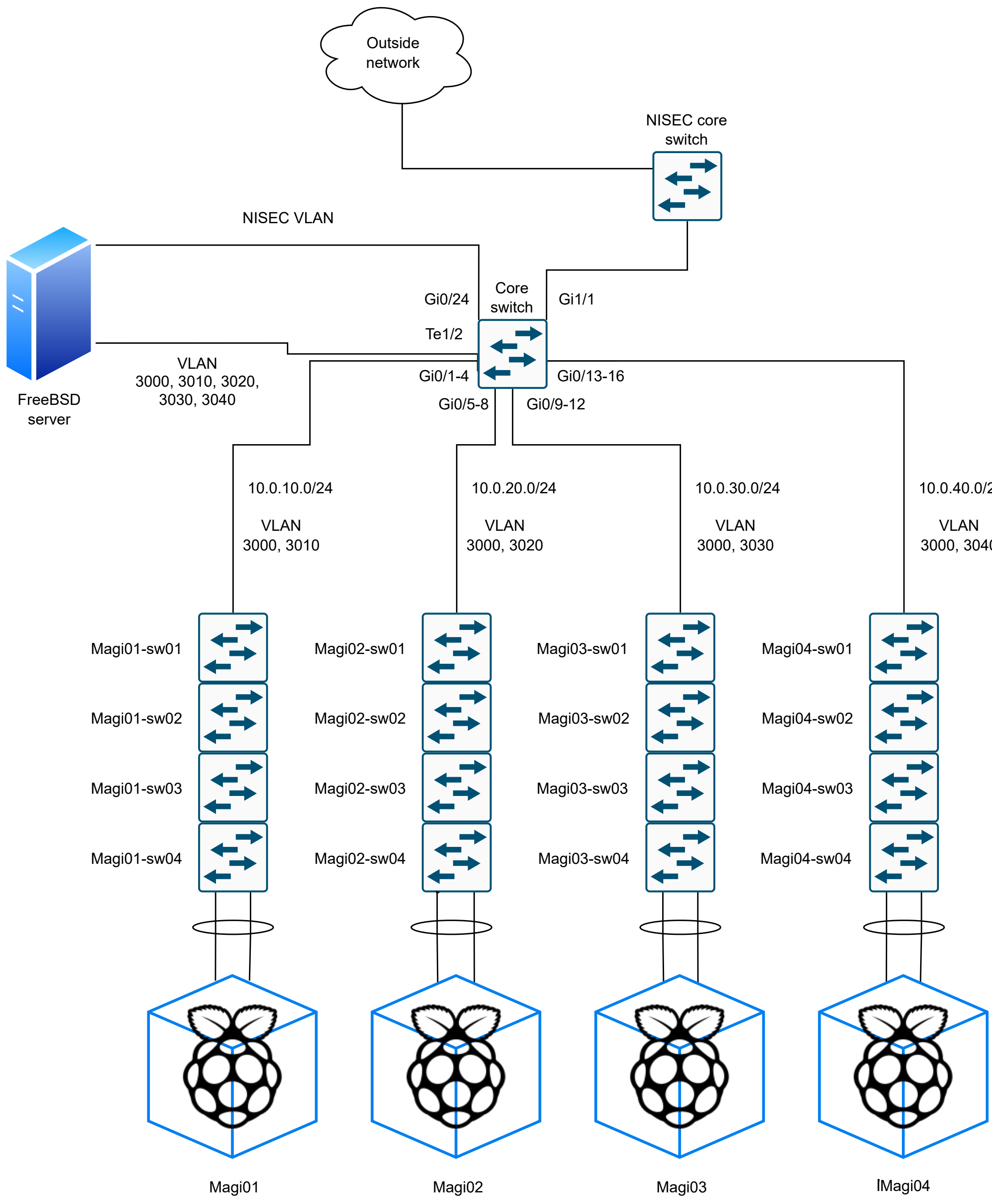

***Figure 3.1.*** *Overview of the Magi cluster*

Raspberry Pi nodes, and job scheduling. Devuan ASCII and FreeBSD both meet the FOSS criterion.

On the cluster, 640 RPi devices are physically and logically partitioned into four "cubes". Therefore, each of the four logically and physically distinct cubes contains 160 RPi nodes. Section 3.4 describes the design decisions for the RPi nodes, whereas Section 3.5 describes the design decisions for the FreeBSD server in more detail. The overview of the Magi cluster and its network configuration is presented in Figure 3.1.

### 3.3.1 Network configuration

As illustrated in Figure 3.1, each cube resides behind its own VLAN with its own subnetwork and four switches that are connected via a single “core switch” to the internet and the FreeBSD 13 server. The entire cluster resides behind a `10.0.0.0/16` Classless Inter-Domain Routing (CIDR) block. Each cube has its own `/24` CIDR block, allowing each cube to theoretically support up to $256 - 2 = 254$ RPi nodes. In practise, each cube only contains 160 RPi nodes, but this number can be increased in the future to facilitate simple expansion. As there are 256 `/24` CIDR blocks available within the `/16` CIDR block, and we are only utilising 4 blocks for the RPi nodes and 2 blocks for the management networks, there is ample room for future project expansion.

## 3.4 Raspberry Pi nodes

Raspberry Pi is an embedded single-board computer. As RPis are readily available, user-friendly, well supported by open-source software, and have a wealth of existing documentation for various projects, they are suitable for use as Magi cluster nodes.

Due to the plethora of RPi models, we must examine the characteristics of each model before selecting one for the Magi cluster. As we describe in Section 3.2, the RPi nodes must support PXE booting. The RPi model 3b is the first model to support PXE booting without an SD card [143]; however, PXE booting with RPi model 3b requires the setting of an OTP software flag. Due to the need for manual labour on each RPi node, we determine that the RPi model 3b is inapplicable to the Magi cluster. Additionally, many known problems with the PXE boot process with the RPi model 3b were fixed in the RPi model 3b+ [143]. The RPi model 3a+ does not support PXE booting [143]. Consequently, only the RPi models 3b+ and 4b are suitable choices for the Magi cluster, as they are the only models to properly support PXE booting out of the box. RPi 4b is more powerful than RPi 3b+, but also considerably more expensive. Since the Magi cluster will not be used for HPC, we will not noticeably benefit from the increased computing power. As a result, we prioritise the lower price of RPi 3b+ over the increased computing power of RPi 4b, and choose RPi model 3b+ as the base model for the Magi cluster.

### 3.4.1 Network boot process

We use PXE to boot the RPi clients over the network. The PXE booting procedure functions fundamentally as follows (and is illustrated in Figure 3.2):

1. The RPi client broadcasts a `DHCP Discovery` message.
2. The server responds with a `DHCP Offer` containing the client’s IP address, bootfile name (`filename`), TFTP server address (`next-server`), and NFS server path

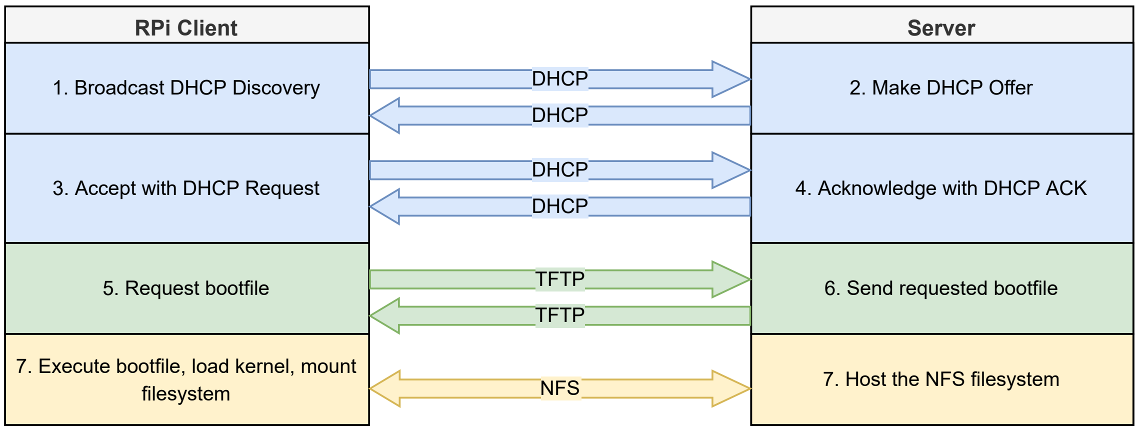
RPi Client
1. Broadcast DHCP Discovery
3. Accept with DHCP Request
5. Request bootfile
7. Execute bootfile, load kernel, mount filesystem
Server
2. Make DHCP Offer
4. Acknowledge with DHCP ACK
6. Send requested bootfile
7. Host the NFS filesystem
DHCP
DHCP
DHCP
DHCP
TFTP
TFTP
NFS

***Figure 3.2.*** *The network boot process*

(`root-path`) values.

3. If the RPi client finds the `DHCP Offer` satisfactory, it broadcasts a `DHCP Request`.
4. The server responds to the newly assigned IP address with `DHCP ACK`.
5. Using the `next-server` and `filename` values included in the `DHCP Offer` message, the RPi client requests the bootfile from the server via TFTP.
6. The server sends the requested bootfile to the client via TFTP.
7. The client executes the received bootfile, which loads the kernel. The NFS filesystem is then mounted using the `root-path` value obtained from the server during the `DHCP Offer`.
8. From this point forward, the OS handles booting the RPi client.

In step 1, the RPi client transmits the `DHCP Discovery` five times, with five seconds between each transmission. The RPi client will enter a low-power state and the PXE boot will fail if the server fails to respond within this time. This could occur if the server is utilising a Spanning Tree Protocol (STP) with listening and learning states, as this procedure could take longer than 25 seconds. In order for the server to see the `DHCP Discovery` messages, it is necessary to configure the connected switches to use STP PortFast mode (or disable STP entirely). Using the SD card's bootcode.bin file, which is stored on the device, is a known workaround for this issue. However, this would necessitate manual labour on each RPi node, which would contradict the requirements.

### 3.4.2 Operating system

Based on Section 3.2 and our prior knowledge, we have the following criteria for selecting the operating system for the RPi nodes:

- The selected OS must be FOSS. We do not wish to be reliant on proprietary, closed-source software; rather, we desire total control over the cluster.

- The operating system must be stable under high stress and have minimal overhead to avoid wasting the computing power of the RPi nodes. According to our previous tests, systemd [144] can cause nodes to crash under heavy load and adds a substantial amount of unnecessary overhead. Thus, we avoid operating systems and distributions with systemd.
- The selected OS should not have reached the end of its product lifecycle (end-of-life product, EOL). We do not want to select EOL products in the event that a critical patch is required (e.g., a newly discovered security flaw), but the OS will no longer be updated.
- The selected OS should require few manual updates. Since we want the Magi cluster to be as stable as possible, we will only apply critical OS updates to a stable OS build. For this reason, we disregard all frequently updated OS development builds.

We select Devuan ASCII Linux distribution [145] as the OS for the nodes in the Magi cluster based on these criteria. It is a stable, lightweight, systemd-free Linux distribution with ready-to-use RPi 3 images, thus meeting all of our criteria.

## 3.5 FreeBSD server

We choose FreeBSD 13 as the server for the Magi cluster. FreeBSD is a free open-source OS derived from the BSD version of Unix that fulfils our requirements listed in Section 3.2. It is developed and maintained by a large open-source community and can be freely downloaded from the FreeBSD project website [146]. We list below the most important features of FreeBSD we use for the Magi cluster:

**Dynamic Host Configuration Protocol** (DHCP) is utilised for automatically assigning IP addresses to RPi client nodes and distributing all necessary information for the network boot procedure. Subsection 3.4.1 describes how and which files are sent through DHCP during the network boot procedure. In practise, the DHCP server residing on the FreeBSD 13 server machine comprises configurations for six separate VLANs; one for each of the four cubes, one for the management VLAN, and one for the NISEC's internal VLAN.

**Network File Share** (NFS) provides RPi client nodes with a network-based file system without requiring an SD card on each node computer. PXE utilises TFTP to transfer required files from the server to the client nodes during startup. All changes to the client's filesystem can be made on a single server computer, which then propagates to all client nodes. This makes the computer cluster more scalable and easier to maintain.

**Packet Filter** (pf) is an internal packet filtering tool in FreeBSD that we use for the Magi cluster. It acts as a firewall between our private subnetwork and Tampere University's private network. Furthermore, all packets originating from outside Tampere University's network are blocked, thus hardening the security of the server.

**Slurm Workload Manager** serves as the cluster's job scheduler. It enables us to run and monitor parallel jobs on multiple RPi nodes and to allocate groups of RPi nodes to cluster users for their work.

**Trivial File Transfer Protocol** (TFTP) is used to transfer files from the server to the RPi client nodes over the network so they can be network booted using PXE. Subsection 3.4.1 describes how and which files are sent through TFTP during the network boot procedure.

**ZFS** filesystem is used on the server. ZFS is a filesystem that was originally designed for Sun Solaris and later ported to other Unix-like operating systems, including Linux and FreeBSD. It has advanced file system properties, such as snapshots and replication at the pool level, that are useful for our uses for the Magi cluster. It also enables users to create pools of hard drives in their own pool of mirrors without the need for RAIDs at the hardware level. Our server configuration makes use of two distinct pools of mirrors. The system host resides on its own pool of two hard discs in mirror, whereas the cluster resides on its own pool of $2\times2$ mirror.

# 4. DESIGNING AN EAAS SIMULATION WITH THE MAGI CLUSTER

To develop a practical application for the Magi cluster described in Chapter 3, we implement an EaaS simulation. This chapter highlights why and how the EaaS simulation is developed. We use the background information provided in Section 2.6 as a baseline for our simulation. Initially, we examine existing works on EaaS implementations in greater detail in Section 4.1. Then, we explain the security considerations and the threat model on which our design is based in Section 4.2. The objectives of the simulation are then discussed in Section 4.3, followed by a thorough explanation of its design and methodology in Section 4.4. Finally, we describe the limitations and biases of our simulation in Section 4.5. The results of the simulation are presented in Chapter 5.

## 4.1 Related works

Before we begin designing our own EaaS implementation, we examine existing EaaS solutions. Existing related works serve as an inspiration for our design. For our search of relevant related works, we first take a look into implementations that are supported by pertinent scientific literature. We exclude proprietary industry solutions whose implementation details are not publicly available from this search. Our search for scientific literature commences with the following search engines:

- *ACM Digital Library*[1]
- *Andor*[2]
- *arXiv open-access archive*[3]
- *dblp computer science bibliography*[4]
- *Google Scholar*[5]
- *IEEE Xplore*[6]

[1]`https://dl.acm.org/`
[2]`https://andor.tuni.fi/`
[3]`https://arxiv.org/`
[4]`https://dblp.org/`
[5]`https://scholar.google.com/`
[6]`https://ieeexplore.ieee.org/`

We use the following search terms and their combinations during the search: “EaaS”, “Entropy”, “as”, “a”, “Service”, “IoT”, and “randomness”.

According to our search results, “EaaS” is not commonly understood to mean “Entropy as a Service”. Specifically, the letter “E” in “EaaS” represented different words such as “Ecosystem”, “Edge”, “Education”, “Electricity”, “Emulation”, “Encryption”, “Energy”, “Entanglement”, “ERP (Enterprise Resource Planning)”, “Evaluation”, “Everything”, “Execution”, and “Exostructure” in many of the search results, where the paper was about offering something else as a service. After eradicating all search results that are not about “Entropy as a Service”, we are left with only the two works presented in Table 4.1.

***Table 4.1.*** *Related EaaS works.*

| **Title of the paper (abbreviated)** | **Authors** | **Year** | **Reference** |
|---|---|---|---|
| EaaS: Unlocking Cryptography’s Full Potential | Vassilev and Staples | 2016 | [93] |
| EaaS: Lightweight RNG for Decentralised IoT Apps | Ullah et al. | 2020 | [110] |

The first related work by Vassilev and Staples [93] focuses on the NIST EaaS project [98]. It employs a QRNG source that passes the Bell test [101]. The design is centralised, meaning it has a single point of trust. In the design, the HTTP protocol is used to transfer entropy from the service to clients, and the generated entropy undergoes continuous TRBG health test monitoring. Timestamps and digital signatures are also used to strengthen the design’s security.

Ullah et al. [110] design a CSPRNG that uses sensor data as a source of randomness in the second related work. In addition, they present a proof of concept for the potential use of sensor data as a source of randomness and evaluate the generated entropy using the NIST SPS [54]. While the contents of the paper are relevant to this thesis, it does not actually implement an “Entropy as a Service” system in the sense that we understand the term. Therefore, the paper’s title is somewhat deceptive, and it is of little assistance to us in designing the EaaS simulation.

Based on the search results for related scholarly literature, EaaS does not appear to be an active area of research. This is in stark contrast to randomness beacons, which return a plethora of search results from recent years using the same search engines with terms such as “randomness”, “random”, “beacon”, “beacons”, “protocol”, and “protocols”. This begs the question of whether it is worthwhile to implement an EaaS system or whether a randomness beacon would be more advantageous. Nonetheless, if we expand the search for related works to include industry solutions and use the same EaaS-related keywords on *Google search*[7], we find numerous results of real-world EaaS implementations by companies such as Crypto4A, QNu Labs, Qrypt, Quantropi, and QuintessenceLabs [111–

[7]`https://www.google.com/`

115]. All of these industrial solutions utilise QRNG sources and emphasise the “quantum” aspect of the solution in their marketing. These results indicate, in our opinion, that EaaS implementations, particularly with QNRG sources, have real-world monetary value for businesses and therefore should be studied more by the academic community in addition to randomness beacons.

## 4.2 Security considerations and threat model

In our simulation, we have full control over both the server and the IoT clients. In addition, all EaaS simulation-related computers are contained within our private subnetwork. Only the FreeBSD server controlling the Magi cluster is accessible from the external network. In this case, the external network is the Tampere University network. All packets originating outside the university’s network are dropped. Additionally, many packets originating from this network are blocked by default due to the fact that the FreeBSD server employs packet filtering that whitelists only specific protocols. Furthermore, all computers associated with the simulation are physically located in a server room with restricted access. Few individuals are physically capable of entering this server room. These safeguard our EaaS system against the vast majority of attacks and adversaries, as we assume adversaries cannot gain physical access to the computers and remote access requires adversaries to first connect to the Tampere University network.

However, we use a conservative threat model, in which we assume that an adversary will be able to pivot into our private subnet and is therefore capable of launching both active and passive attacks within the network. Due to this, we assume that the attacker can conduct MitM attacks, response-replay attacks, DNS poisoning attacks, and even gain root access to the server.

To prevent MitM attacks, we employ both asymmetric encryption and digital signatures. Since the adversary does not have access to the IoT client’s private key in our threat model, they cannot read or modify the entropy generated by the server. The digital signature verification offers protections for authenticity, integrity, and non-repudiation, hardening the system against MitM and DNS poisoning attacks. We use the Network Time Protocol (NTP) to provide timing information for both the IoT client’s request for entropy and the server’s generation of entropy in order to prevent response-replay attacks. The IoT client verifies that entropy is only generated in response to a request for it.

We employ TEEs to safeguard our implementation from adversarial root access on computers participating in the simulation. All of the code and data associated with the EaaS simulation is executed within Enarx’s keeps; therefore, even with root access to the server or IoT clients, an adversary cannot read or process the data, enhancing the system’s integrity and confidentiality. Consequently, even we as the operators of the EaaS server are unable to read or modify the generated entropy. Users do not need to rely on *us* being trustworthy and operating the server with good intentions, but rather on the TEE technology to function properly and prevent us from observing or altering the data on the server. While the server continues to function as a single point of trust, it is not a typical Trusted Third Party (TTP) server, as in the NIST's EaaS implementation [93]. Instead, we refer to our server as the Trusted Execution Server (TES) to emphasise that trust is placed in the TEE technology and not the server operator.

In order to use asymmetric encryption effectively, we employ a key provision model. This indicates that we anticipate the IoT clients to be shipped with robust keys provided by the manufacturer. Without the initial strong keys provided by the manufacturer, the IoT clients would need entropy from the EaaS server to generate strong keys; however, using the EaaS service securely against powerful active adversaries requires strong keys. Using the key provision model gets rid of this chicken-and-egg problem. In addition, we assume that IoT clients already know the server's public key prior to the simulation.

## 4.3 Goals

The main goals of our simulation are the following:

1. Implement a functional EaaS system. Due to the scarcity of academic reference implementations of other EaaS systems, our simulation is vital for demonstrating that building an EaaS system is actually feasible.
2. Find out whether EaaS is a practical solution for solving IoT RNG problems.
3. Determine whether the use of TEEs to enhance the security of the EaaS system is feasible in practise. Based on our experience writing the SoK paper [2], the development of TAs is not a simple process; therefore, we are interested in determining whether Enarx can be used to harden the EaaS system security with TAs.

As a consequence, our simulation aims to give answers to the research questions **RQ3**, **RQ5**, and **RQ6**. We will discuss the extent to which our simulation achieves these objectives in Chapter 5.

## 4.4 Design and methodology

The primary components of our simulation are the Trusted Execution Server (TES), the entropy sources, and the IoT clients that utilise the TES's entropy. The TES appears to IoT clients as a black box that returns entropy on demand. From the perspective of the entropy sources, the TES is a black box that requests and receives entropy from the entropy sources. From the perspective of the TES, the architecture is somewhat more complex because the server must communicate with IoT clients that request entropy as well as entropy sources that gather and distil entropy from nature. The TES must also utilise a CSPRNG function to combine the entropy sources.

When IoT clients request entropy from the TES, they send an HTTP GET request along with their own public key ($pk_{IoT}$) and the quantity of entropy required ($\Delta x$). Additionally, they utilise NTP to store the request time ($t_1$) locally. The server then responds with the requested entropy ($x$), its cryptographic digital signature ($s$), and the time the entropy was generated ($t_2$). The response is encrypted using the public key of the IoT client. When the IoT client receives the returned data, it verifies it prior to employing the entropy. The entropy is discarded if the verify function returns an invalid result. The verify operation includes three steps:

1. The IoT client verifies that the data is encrypted correctly using its own public key and that the response is in the correct format, i.e. the response contains the requested entropy $x$ with the requested amount of entropy $\Delta x$, the digital signature $s$, and the time the entropy was generated $t_2$.
2. The client verifies that the entropy was generated only after the request for entropy ($t_2 > t_1$).
3. The client then verifies the validity of the digital signature.

Figure 4.1 demonstrates the simulation protocol from the perspective of IoT clients.

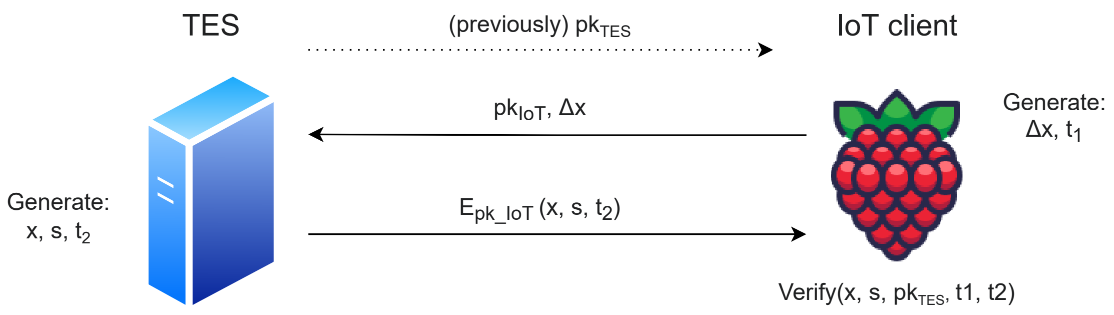

***Figure 4.1.*** *Protocol for EaaS simulation from the perspective of the IoT client.*

If the verify function fails, we employ a blocking implementation that terminates the process that requested entropy from the EaaS. Because we control and trust our entire simulation implementation, which is located within a self-controlled private subnet, and because we don't use IoT devices for anything critical during the simulation that would always require a functioning state for the devices, a blocking implementation is suitable for us. However, a non-blocking implementation could also be used by simply discarding the entropy received from the EaaS and proceeding with only the IoT device's own HWRNG entropy sources.

We use Enarx to run the simulation inside a TEE. Using a TEE enhances system security because, even if the computers comprising the EaaS system are compromised, the in-

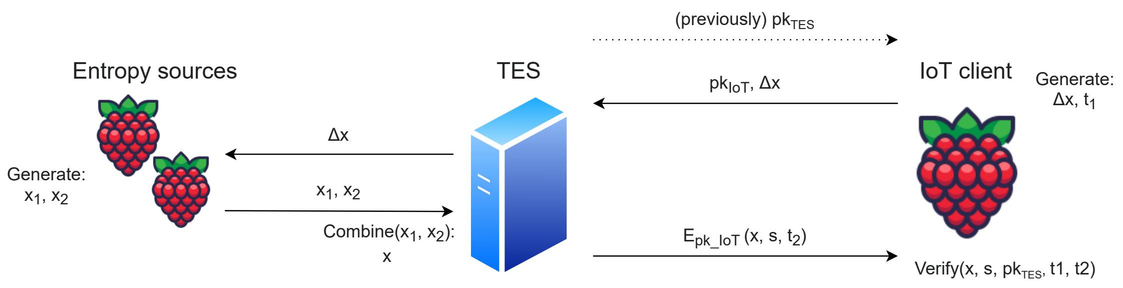

***Figure 4.2.*** *Protocol for EaaS simulation from the perspective of the TES.*

tegrity and confidentiality of the processed data remain intact, as the high-level operating system is unable to view or modify the data.

In our simulation, both the IoT clients requesting entropy and the entropy sources are RPi 3b+ nodes from the Magi cluster. In practise, two distinct nodes serve as entropy sources. The process of entropy distillation is simulated utilising the CSPRNG function ($/dev/urandom$ [74]) included in the Devuan ASCII Linux distribution [145]. When requested by the server, the RPi nodes serving as entropy sources generate and send their generated entropy to the server. We use another RPi node in the Magi cluster as the TES because FreeBSD does not properly support Enarx. After receiving entropy from the entropy sources, the server uses its own CSPRNG function to combine these two distinct entropy sources. Finally, the server sends the properly mixed entropy to the IoT device that requested the entropy in the first place. Figure 4.2 highlights the simulation protocol from the perspective of the TES.

## 4.5 Limitations and bias

Since we are using Enarx's "nil" backend, our design will not be secure in practise. Consequently, we will not utilise ARM's TrustZone capabilities in reality. According to our SoK of TEE development tools and applications [2], there are no open-source tools readily available for TrustZone-enabled development. Enarx, which supports actual TEE usage on other architectures such as Intel SGX and AMD SEV, with the nil backend allows us to develop applications in the same manner as if we were actually using the proper TEE backends. This is sufficient for our simulation, but for a secure implementation in the real world, we would need a middleware framework that supports TrustZone-enabled development.

Our architecture includes a TES. Consequently, we are limited to a single point of trust. Since we do not share the entropy from our EaaS simulation over the internet and we have complete control over the TES within our private subnet, a more conventional TTP server would also meet our needs. Using a TES, on the other hand, enables us to scale our EaaS implementation in the future for a more robust design in which the entropy is actually broadcast to other internet users. In addition, our design does not include a remote attestation mechanism because it is unnecessary in a centralised and self-controlled design. In a decentralised architecture, integrity guarantees require appropriate remote attestation, and the entropy sources should be independent and controlled by separate parties, in contrast to our design in which we control all entropy sources. This further mitigates the issue of a single point of trust, as a single colluding entity (capable of breaking the memory isolation of the TEE) is unable to manipulate the output entropy of the EaaS.

Our entropy sources are also limited in this design, as we only use identical hardware choices and TRNG collection techniques. Multiple, distinct, and unrelated entropy sources must be utilised in a more secure implementation. We could improve our simulation by employing e.g. Physical Unclonable Function (PUF) based ring oscillator TRNGs [61–66] or QNRG devices [56–58] as entropy sources. Even in these designs, it is essential to evaluate the conditions under which they can generate strong entropy. In a ring-oscillator-based PUF design, for instance, the temperature and voltage must remain within the proper condition ranges; otherwise, the entropy will be biased [70, 71]. Therefore, the source of entropy must be guarded against fault injection attacks that seek to interfere with its condition range.

Since many CPU manufacturers exclude SCAs from their TEE threat model [52], TEEs do not often provide adequate protection against SCAs in the real world, as demonstrated in the surveys by Cerdeira et al. [147] and Fei et al. [148]. Therefore, our design is fundamentally insecure against highly skilled and well-resourced adversaries who can craft specific SCAs against the various EaaS simulation components employing TEEs. This would be the case even if our design utilised the ARM TrustZone TEE backend correctly. Since TEE manufacturers are typically uninterested in fixing the fundamentally flawed parts of their hardware design that can be exploited with SCAs [52], the only way to protect against these attacks is to conduct extensive in-house research on the possible SCAs against the used TEE system and then implement all the necessary software modifications to protect against the discovered vulnerabilities. Even so, it may be impossible to protect against specific SCAs, and regardless of how well internal research on SCAs is conducted, it is impossible to identify every possible defence. Therefore, on a fundamental level, the TEEs do not provide the design-based security guarantees in the real world that they theoretically provide. However, the use of TEEs to secure the implementation is not in vain, even if SCAs could theoretically be used to attack TEEs. This is due to the fact that utilising TEEs drastically reduces the system's attack surface by providing stronger protections for data confidentiality, integrity, and authenticity. As we could not find many instances of SCAs being utilised in the wild against TEE-enabled systems, the use of TEEs appears to deter all but the most highly skilled and resource-rich attackers, such as nation-state actors. This substantially decreases the attack surface. Consequently, the use of TEEs to strengthen system security remains highly advantageous.

It is also important to note that at no point in the simulation do we evaluate the statistical quality of the generated entropy. The NIST statistical suite [54] could be employed to evaluate the entropy of the EaaS simulation. Similarly, NIST-Lite [87] could be used to test the output randomness of IoT clients after generating randomness using entropy from EaaS. As explained in Subsection 2.5.5, these test suites cannot prove that the randomness is truly secure. However, they can be used to demonstrate that the entropy is at least not known to be insecure.

In general, it is essential to acknowledge that our current design **is not secure in the real world**. Due to this, the design is restricted to simulation and testing purposes only. For a truly secure implementation, each of the aforementioned biases and limitations must be addressed.

# 5. RESULTS FROM THE SIMULATION

In this chapter, we explain how our EaaS system, whose design was presented in Chapter 4, was implemented in practise, as well as the challenges and considerations we faced during the implementation. In addition, we discuss whether our EaaS implementation demonstrates that a secure distributed EaaS system utilising TEEs is feasible to implement and can it resolve the issues with IoT RNG presented in Subsection 2.5.6. Section 5.1 discusses the practicality of implementing an EaaS system, while Section 5.2 discusses the practicality of developing TAs to harden the security of an EaaS system.

## 5.1 Practicality of implementing an EaaS simulation

We used Rust programming language and Enarx for the development of the EaaS system. The related code and script files for actually running the simulation can be found from our GitLab repository [1], where we published them under the MIT open-source license.

For the entropy generation, we used the `ChaCha12` algorithm as the CSPRNG, and the `FromEntropy` function in Rust for seeding. `FromEntropy` first attempts to use `OsRng` (`/dev/urandom` of the ASCII Devuan Linux distribution [145] in our case) as the entropy source, and if that fails, `JitterRng` (which uses the jitter in the CPU execution time) as a backup. We used `ChaCha12` as opposed to the more common `ChaCha20` because, according to Aumasson [149], utilising ChaCha with significantly fewer than 20 rounds does not increase the security risk, but makes it significantly faster. In particular, according to Aumasson, ChaCha would be secure with only eight rounds, but we opt for twelve rounds as a prudent tradeoff between a reduced execution time and a larger security margin.

The actual code for the entropy generation function is as follows:

```
fn get_random_u32() -> u32 {
    let mut csprng = rand_chacha::ChaCha12Rng::from_entropy();
    let random_u32 = csprng.next_u32();
    return random_u32;
}
```

To get the networking to function, we utilised an asynchronous HTTP server `mini_http` [150]. For the asymmetric encryption of the data that the server provides to the IoT client, we used the client's public SSH key. Similarly, we used the TES's SSH keys (with Ed25519 algorithm) for the generation and validation of the digital signature.

Our simulation demonstrates that an EaaS system can be implemented in practise. Since Magi cluster was successfully utilised in the simulation, we can confidently answer **RQ3**: *implementation of an EaaS system utilising an embedded computer cluster is feasible*. Since we are able to use the strong entropy from the EaaS with the IoT clients, *EaaS seems to be a practical solution for solving IoT RNG problems related to entropy generation*. This gives an partial answer to **RQ6**.

## 5.2 Practicality of developing TAs for the EaaS simulation that utilise TEEs

As presented in Subsection 2.3.1 and Section 4.5, Enarx cannot be used to develop TAs that run on actual TrustZone hardware. However, it can be used to develop TAs that run on Intel SGX and AMD SEV, and with the "nil" backend, the same TAs can run on other hardware, such as the RPi 3b+'s used in the Magi cluster, albeit without the TEE hardware component.

When attempting to implement a simple TCP client networking using sockets, we encountered issues even with the nil backend. To make client-side networking function, we had to contribute additional development time and effort to the Enarx project. Also, Enarx does not currently support reading and writing files. This meant that the key values in our code had to be hardcoded. Moreover, we discovered that the documentation is frequently insufficient or out of date, necessitating direct contact with Enarx developers to resolve a number of our issues. These issues demonstrated that Enarx is not yet mature enough for all development aspects, and that the concept of "simply running an existing application binary within Enarx" is still a long way off.

In our SoK paper [2], we found that there do not appear to be any open-source development frameworks or containers for TrustZone; rather, developers rely on proprietary development frameworks to run their applications within TrustZone. To circumvent this limitation and actually run the EaaS system within a TEE, we would need to switch to hardware that supports Intel SGX or AMD SEV.

However, despite the aforementioned limitations, we were able to implement a working distributed EaaS system that utilises TAs. The related code and script files can be found from our GitLab repository [1]. In order for networking to function within Enarx, we had to preopen a socket and pass it to the TA in the `Enarx.toml` file. In the actual code, we differentiated between a WASI (Enarx) build and a non-WASI ("normal") build. In the case of a WASI build, the pre-opened TCP stream was passed to the TA by reading the existing pre-opened connection from the `Enarx.toml` file in the following line:

```
let stdstream = unsafe { std::net::TcpStream::from_raw_fd(3) };
```

In the case of a non-WASI build, we simply established a new TCP connection in the line below:

```
std::net::TcpStream::connect("127.0.0.1:3445")
```

Afterwards, the TCP streams were handled identically in the code.

On the basis of these observations, we can answer **RQ5**: *it is possible to develop TAs for an EaaS system, albeit with limitations*. In our case, we were unable to use the actual hardware protections of the TEE because we had to use the nil backend of Enarx. Furthermore, due to other limitations in Enarx, we had to use hardcoded public key values, and to get simple TCP networking to work between a client and server, we first had to do additional open-source development to the Enarx project.

# 6. CONCLUSIONS

After providing the necessary background information, we designed and built an embedded computer cluster and used it to simulate EaaS. The purpose of the simulation was to address the existing issues with IoT RNG. In addition, we utilised TEEs in our simulation to assess the viability of using TEEs in a distributed manner in the EaaS simulation and to improve the service's integrity and confidentiality.

In this chapter, we provide detailed responses to the research questions posed in Section 1.3, followed by a discussion of potential future research in the form of open questions based on the limitations of our EaaS simulation.

## 6.1 Answers to research questions

We presented the following research questions in Section 1.3:

**RQ1.** How did we implement a computer cluster to meet the needs of our research group?

**RQ2.** How does IoT RNG fail in practice?

**RQ3.** Is it feasible to create an EaaS system using an embedded computer cluster?

**RQ4.** How can TEEs be used to harden the security of an EaaS system?

**RQ5.** Is it feasible to develop TAs that utilise TEEs for the EaaS system?

**RQ6.** Is EaaS capable of resolving existing IoT RNG issues?

Chapter 3 provided a comprehensive response to **RQ1**, while Section 2.1 explained the theoretical context of the implementation. The key points were that the design must be based on the use cases of the specific cluster and the stakeholders' available resources. Consequently, each computer cluster is unique. When discussing computer clusters designed for specific research groups, the HPC use cases are typically not nearly as important as the research-specific use cases. In university research groups, learning about distributed computing principles and HPC practice are typically important motives for constructing and employing computer clusters [22, 23, 26]. In the case of NISEC, the requirements were primarily associated with embedded or distributed system research. Consequently, the Magi cluster was implemented as an embedded computer cluster comprised of 640 RPi 3s, and the EaaS simulation was performed to test the cluster with a practical use case in order to conduct research on both embedded and distributed systems.

**RQ2** was answered in detail in Subsection 2.5.6, while Section 2.2, Section 2.4 and Section 2.5 provided the needed background information to understand Subsection 2.5.6. The main findings were that IoT devices typically lack access to CSPRNG subsystems due to their lack of a modern OS, that developers frequently make incorrect calls to the HWRNG components, that handling HWRNG errors correctly is difficult, if not impossible, at times, that documentation for bare-metal function calls is frequently inadequate, which makes their proper use challenging, and that numerous IoT devices generate HWRNG of subpar quality. In addition, IoT devices are susceptible to weak entropy generation during boot process, even if they employ an OS that supports a proper CSPRNG subsystem.

**RQ3**, **RQ4**, and **RQ5** were answered in Chapter 4 where we presented the design for our EaaS simulation and in Chapter 5 where we presented the results from the simulation.

The results from the EaaS implementation presented in Section 5.1 gave a definite answer to **RQ3**. Implementing a working EaaS system is feasible on an embedded cluster, as was demonstrated by our EaaS implementation running on the Magi cluster. Furthermore, we published the code repository [1] under open-source (MIT) license, allowing other developers and researchers to freely use it for future EaaS implementations.

**RQ4** was answered in Section 4.4 where we explained the threat model for the EaaS simulation and how TEEs can be used to enhance the system's integrity and confidentiality against adversaries with root access to the Trusted Execution Server and the entropy sources, thus hardening the EaaS system.

In Section 5.2 we determined that the development of TAs for our EaaS system with Enarx is possible, but not straightforward. **RQ5** cannot therefore be answered with a simple yes/no response. In theory, there are tools for developing TAs for various TEE architectures; however, in practise, these tools may not be suitable for developing TAs that run on actual TEE hardware, and software developers with no prior experience working with TEEs may find them extremely difficult to use due to frequently outdated or lacking documentation. As was the case with Enarx, the available tools may also be immature in the sense that they cannot be used for things such as networking or file reading and writing.

For **RQ6**, Section 2.6 provided a theoretical foundation for solving the problem with IoT RNGs using either randomness beacons or EaaS, and in Chapter 4, we presented our own EaaS design aimed at resolving these issues. The simulation results presented in Chapter 5 seem to indicate that EaaS is indeed capable of resolving IoT RNG issues in a practical manner. However, our implementation of the EaaS is not secure for real-world implementations due to the limitations highlighted in Section 4.5. Therefore, additional research must be conducted on our EaaS system. The real-world industry EaaS implementations by various companies [111–115] seem to also suggest that EaaS is a good solution for providing IoT devices with strong entropy.

## 6.2 Future work and open questions

Our EaaS implementation is not secure in the real world, as described in Section 4.5. **To make our EaaS system truly secure, we would need to implement the following**:

- Utilise additional hardware entropy sources, such as QRNG devices [56–58] or TRNGs based on ring oscillator PUFs [61–66]. Currently, our simulation is comprised solely of entropy sources with identical hardware and software architectures, as well as identical entropy generation. In a truly secure implementation, multiple entropy sources should be combined to ensure that a bias in one source does not affect the generated entropy.
- Utilise decentralisation rather than a centralised TES/TTP server, to get rid of the single-point-of-trust problem.
- Implement proper remote attestation to harden the system integrity in a decentralised design.
- Evaluate the generated entropy statistically, so that we can be certain that the system is not absolutely insecure.
- Secure the EaaS system against SCAs. CPU manufacturers often exclude SCAs from their TEE threat model [52], and there are many examples of real-world TEE vulnerabilities [147, 148]. Since we employ no additional SCA defences, we must assume that our system is susceptible to SCA attacks.

Nevertheless, *our EaaS implementation demonstrates that a secure distributed EaaS system can be created using TEEs* by adhering to all the aforementioned steps.

The major unanswered question is **whether utilising DRBs would be preferable to EaaS**. With DRBs, we could collaborate with other open-source parties conducting research on the subject, such as the league of entropy [109]. Based on our findings of related projects in Section 2.6 and Section 4.1, DRBs and randomness beacons are being studied significantly more than EaaS; therefore, this may be a better direction overall.

Finally, we must study **how to more effectively utilise TEEs within an EaaS or DRB system**. Utilising a proper TEE backend is the first and most essential step. Based on our SoK paper [2], it appears that the TrustZone TEE included in the RPi3's ARM SoC does not permit easy open-source development, so we used Enarx's nil backend in our simulation. In the absence of open-source development tools for TrustZone, a truly secure implementation would require a switch to a different hardware architecture. Similarly, our EaaS design lacks a remote attestation mechanism. However, if we were to generalise our design into a decentralised version that could be utilised by multiple parties, it would be necessary to implement remote attestation to strengthen the system's integrity.

# REFERENCES

[1] Network and Information Security Group (NISEC) at Tampere University. *EaaS-TEE: Entropy as a Service Within Trusted Execution Environment*. 2022. URL: `https://gitlab.com/nisec/eaastee` (visited on 11/03/2022).

[2] Paju, A., Javed, M. O., Savimäki, J., Nurmi, J., and Brumley, B. B. *SoK: A Systematic Review of TEE Usage for Developing Trusted Applications*. Preprint. (can be found from: https://trepo.tuni.fi/). Nov. 2022.

[3] Gómez López, J., Villar, E., Molero, G., and Cama-Pinto, A. Evaluation of High Performance Clusters in Private Cloud Computing Environments. *Distributed Computing and Artificial Intelligence - 9th International Conference, DCAI 2012, Salamanca, Spain, 28-30th March, 2012*. Ed. by S. Omatu, J. F. De Paz Santana, S. Rodríguez-González, J. M. Molina, A. M. Bernardos, and J. M. Corchado Rodríguez. Vol. 151. Advances in Intelligent and Soft Computing. Springer, 2012, pp. 305–312. DOI: `10.1007/978-3-642-28765-7_36`. URL: `https://doi.org/10.1007/978-3-642-28765-7_36`.

[4] Cheung, A. L. and Reeves, A. P. High Performance Computing on a Cluster of Workstations. *Proceedings of the First International Symposium on High Performance Distributed Computing, HPDC '92, Syracuse, NY, USA, September 9-11, 1992*. IEEE, 1992, pp. 152–160. DOI: `10.1109/HPDC.1992.246477`. URL: `https://doi.org/10.1109/HPDC.1992.246477`.

[5] TOP500. *List Statistics*. URL: `https://www.top500.org/statistics/list/` (visited on 06/14/2022).

[6] Yeo, C. S., Buyya, R., Pourreza, H., Eskicioglu, M. R., Graham, P., and Sommers, F. Cluster Computing: High-Performance, High-Availability, and High-Throughput Processing on a Network of Computers. *Handbook of Nature-Inspired and Innovative Computing - Integrating Classical Models with Emerging Technologies*. Ed. by A. Y. Zomaya. Springer, 2006, pp. 521–551. DOI: `10.1007/0-387-27705-6_16`. URL: `https://doi.org/10.1007/0-387-27705-6_16`.

[7] IBM. *IBM® Power® AIX® Operating System*. URL: `https://www.ibm.com/it-infrastructure/power/os/aix` (visited on 11/01/2022).

[8] Khalidi, Y. Y. A., Bernabéu-Aubán, J. M., Matena, V., Shirriff, K., and Thadani, M. Solaris MC: A Multi Computer OS. *Proceedings of the USENIX Annual Technical Conference, San Diego, California, USA, January 22-26, 1996*. USENIX Association, 1996, pp. 191–204.

[9] Hoffmann, G.-R. High-Performance Computing and Networking for Numerical Weather Prediction. *High-Performance Computing and Networking, International Conference and Exhibition, HPCN Europe 1994, Munich, Germany, April 18-20, 1994, Proceedings, Volume II: Networking and Tools*. Ed. by W. Gentzsch and U. Harms. Vol. 797. Lecture Notes in Computer Science. Springer, 1994, pp. 1–4. DOI: 10.1007/3-540-57981-8_86. URL: https://doi.org/10.1007/3-540-57981-8_86.

[10] Faeldon, J., Espana, K., and Sabido, D. J. Data-Centric HPC for Numerical Weather Forecasting. *43rd International Conference on Parallel Processing Workshops, ICPPW 2014, Minneapolis, MN, USA, September 9-12, 2014*. IEEE Computer Society, 2014, pp. 79–84. DOI: 10.1109/ICPPW.2014.23. URL: https://doi.org/10.1109/ICPPW.2014.23.

[11] Monteiro, A., Teixeira, C., and Pinto, J. S. HPC in Weather Forecast: Moving to the Cloud. *Int. J. Cloud Appl. Comput.* 5.1 (2015), pp. 14–31. DOI: 10.4018/ijcac.2015010102. URL: https://doi.org/10.4018/ijcac.2015010102.

[12] Hoyle, D. C., Delderfield, M., Kitching, L., Smith, G., and Buchan, I. E. Shared Genomics: High Performance Computing for Distributed Insights in Genomic Medical Research. *Healthgrid Research, Innovation and Business Case - Proceedings of HealthGrid 2009, Berlin, Germany, 29 June - 1 July 2009*. Ed. by T. Solomonides, M. Hofmann-Apitius, M. Freudigmann, S. C. Semler, Y. Legré, and M. Kratz. Vol. 147. Studies in Health Technology and Informatics. IOS Press, 2009, pp. 232–241. DOI: 10.3233/978-1-60750-027-8-232. URL: https://doi.org/10.3233/978-1-60750-027-8-232.

[13] Zhou, L., Rekik, I., Yan, C., and Wu, G. Special Issue on High Performance Computing in Bio-Medical Informatics. *Neuroinformatics* 16.3-4 (2018), p. 283. DOI: 10.1007/s12021-018-9393-x. URL: https://doi.org/10.1007/s12021-018-9393-x.

[14] Filatova, V. and Pestov, L. Medical Ultrasound Tomography Problem: Experimental Data Processing with High-Performance Computing. *Proceedings of the Workshop on Mathematical Modeling and Scientific Computing: Focus on Complex Processes and Systems - dedicated to the memory of Nikolai Botkin, Munich, Germany, November 19-20, 2020*. Ed. by V. L. Turova, A. E. Kovtanyuk, H. G. Bock, F. Holzapfel, and E. A. Kostina. Vol. 2783. CEUR Workshop Proceedings. CEUR-WS.org, 2020, pp. 57–67. URL: http://ceur-ws.org/Vol-2783/paper05.pdf.

[15] Davé, R. Monstrous Galaxies Unmasked. *Nature* 525.7570 (Sept. 2015), pp. 465–466. ISSN: 1476-4687. DOI: 10.1038/525465a. URL: https://doi.org/10.1038/525465a.

[16] Almeida, F., Mediavilla, E., Oscoz, A., and De Sande, F. Applying High Performance Computing Techniques in Astrophysics. *Applied Parallel Computing, State*

*of the Art in Scientific Computing, 7th International Workshop, PARA 2004, Lyngby, Denmark, June 20-23, 2004, Revised Selected Papers*. Ed. by J. J. Dongarra, K. Madsen, and J. Wasniewski. Vol. 3732. Lecture Notes in Computer Science. Springer, 2004, pp. 530–537. DOI: 10.1007/11558958_63. URL: https://doi.org/10.1007/11558958_63.

[17] Stoilov, T., Stoilova, K., and Vladimirov, M. Modeling and Assessment of Financial Investments by Portfolio Optimization on Stock Exchange. *Advances in High Performance Computing - Results of the International Conference on "High Performance Computing", HPC 2019, Borovets, Bulgaria, September 2-6, 2019*. Ed. by I. Dimov and S. Fidanova. Vol. 902. Studies in Computational Intelligence. Springer, 2019, pp. 340–356. DOI: 10.1007/978-3-030-55347-0_29. URL: https://doi.org/10.1007/978-3-030-55347-0_29.

[18] Somasekaram, P., Calinescu, R., and Buyya, R. High-Availability Clusters: A Taxonomy, Survey, and Future Directions. *J. Syst. Softw.* 187 (2022), p. 111208. DOI: 10.1016/j.jss.2021.111208. URL: https://doi.org/10.1016/j.jss.2021.111208.

[19] Furuya, S. and Ueda, K. Load Balancing Method for Data Management Using High Availability Distributed Clusters. *23rd Asia-Pacific Conference on Communications, APCC 2017, Perth, Australia, December 11-13, 2017*. IEEE, 2017, pp. 1–6. DOI: 10.23919/APCC.2017.8303970. URL: https://doi.org/10.23919/APCC.2017.8303970.

[20] Yang, P., Gao, H., Xu, H., Bian, M., and Chu, D. A Load Balancing Method Based on Node Features in a Heterogeneous Hadoop Cluster. *Collaborative Computing: Networking, Applications and Worksharing - 13th International Conference, CollaborateCom 2017, Edinburgh, UK, December 11-13, 2017, Proceedings*. Ed. by I. Romdhani, L. Shu, T. Hara, Z. Zhou, T. J. Gordon, and D. Zeng. Vol. 252. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer, 2017, pp. 344–354. DOI: 10.1007/978-3-030-00916-8_32. URL: https://doi.org/10.1007/978-3-030-00916-8_32.

[21] Ding, L., Zheng, W., Liu, S., and Han, Z. Research and Optimization of the Cluster Server Load Balancing Technology Based on Centos 7. *Human Centered Computing - Third International Conference, HCC 2017, Kazan, Russia, August 7-9, 2017, Revised Selected Papers*. Ed. by Q. Zu and B. Hu. Vol. 10745. Lecture Notes in Computer Science. Springer, 2017, pp. 201–207. DOI: 10.1007/978-3-319-74521-3_23. URL: https://doi.org/10.1007/978-3-319-74521-3_23.

[22] Ludin, M., Weeden, A., Houchins, J., Thompson, S., Peck, C., Babic, I., Muterspaw, K., and Sergienko, E. LittleFe: The High Performance Computing Education Appliance. *2013 IEEE International Conference on Cluster Computing, CLUSTER 2013, Indianapolis, IN, USA, September 23-27, 2013*. IEEE Computer Society,

2013, p. 1. DOI: 10.1109/CLUSTER.2013.6702649. URL: https://doi.org/10.1109/CLUSTER.2013.6702649.

[23] López, P. and Baydal, E. Teaching High-Performance Service in a Cluster Computing Course. *J. Parallel Distributed Comput.* 117 (2018), pp. 138–147. DOI: 10.1016/j.jpdc.2018.02.027. URL: https://doi.org/10.1016/j.jpdc.2018.02.027.

[24] Doucet, K. and Zhang, J. The Creation of a Low-Cost Raspberry Pi Cluster for Teaching. *Proceedings of the 24th Western Canadian Conference on Computing Education, WCCCE '19, Calgary, AB, Canada, May 3-4, 2019*. Ed. by B. Stephenson. ACM, 2019, 7:1–7:5. DOI: 10.1145/3314994.3325088. URL: https://doi.org/10.1145/3314994.3325088.

[25] Penyala, H., Ibrahim, S., and El Mesalami, A. M. The Raspberry Pi Education Mine: For Teaching Engineering and Computer Science Students Concepts Like, Computer Clusters, Parallel Computing, and Distributed Computing. *2020 IEEE International Conference on Electro Information Technology, EIT 2020, Chicago, IL, USA, July 31 - August 1, 2020*. IEEE, 2020, pp. 624–628. DOI: 10.1109/EIT48999.2020.9208242. URL: https://doi.org/10.1109/EIT48999.2020.9208242.

[26] Shoop, E., Brown, R. A., Adams, J. C., and Matthews, S. J. Teaching Distributed Computing Fundamentals using Raspberry Pi Clusters. *SIGCSE 2022: The 53rd ACM Technical Symposium on Computer Science Education, Providence, RI, USA, March 3-5, 2022, Volume 2*. Ed. by L. Merkle, M. Doyle, J. Sheard, L.-K. Soh, and B. Dorn. ACM, 2022, p. 1201. DOI: 10.1145/3478432.3499161. URL: https://doi.org/10.1145/3478432.3499161.

[27] Noergaard, T. *Embedded Systems Architecture - A Comprehensive Guide for Engineers and Programmers*. Elsevier, 2005. ISBN: 978-0-7506-7792-9.

[28] Gupta, B. B. and Quamara, M. *Internet of Things Security: Principles, Applications, Attacks, and Countermeasures*. Feb. 2020. ISBN: 9780429353529. DOI: 10.1201/9780429353529.

[29] National Institute of Standards and Technology. *Networks of 'Things'*. Tech. rep. Special Publication 800-183, July 28, 2016. Washington, D.C.: U.S. Department of Commerce, 2016. DOI: 10.6028/NIST.SP.800-183.

[30] Shaukat, K., Alam, T. M., Hameed, I. A., Khan, W. A., Abbas, N., and Luo, S. A Review on Security Challenges in Internet of Things (IoT). *26th International Conference on Automation and Computing, ICAC 2021, Portsmouth, United Kingdom, September 2-4, 2021*. IEEE, 2021, pp. 1–6. DOI: 10.23919/ICAC50006.2021.9594183. URL: https://doi.org/10.23919/ICAC50006.2021.9594183.

[31] Zahra, A. and Shah, M. A. IoT Based Ransomware Growth Rate Evaluation and Detection Using Command and Control Blacklisting. *23rd International Conference on Automation and Computing, ICAC 2017, Huddersfield, United Kingdom,*

*September 7-8, 2017*. IEEE, 2017, pp. 1–6. DOI: 10.23919/IConAC.2017.8082013. URL: https://doi.org/10.23919/IConAC.2017.8082013.

[32] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K., and Zhou, Y. Understanding the Mirai Botnet. *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*. Ed. by E. Kirda and T. Ristenpart. USENIX Association, 2017, pp. 1093–1110. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis.

[33] Butun, I., Österberg, P., and Song, H. Security of the Internet of Things: Vulnerabilities, Attacks, and Countermeasures. *IEEE Commun. Surv. Tutorials* 22.1 (2020), pp. 616–644. DOI: 10.1109/COMST.2019.2953364. URL: https://doi.org/10.1109/COMST.2019.2953364.

[34] Kietzmann, P., Schmidt, T. C., and Wählisch, M. A Guideline on Pseudorandom Number Generation (PRNG) in the IoT. *ACM Comput. Surv.* 54.6 (2021), 112:1–112:38. DOI: 10.1145/3453159. URL: https://doi.org/10.1145/3453159.

[35] The Confidential Computing Consortium. *Confidential Computing: Hardware-Based Trusted Execution for Applications and Data*. White paper. Jan. 2021. URL: https://confidentialcomputing.io/wp-content/uploads/sites/85/2021/03/confidentialcomputing_outreach_whitepaper-8-5x11-1.pdf (visited on 02/09/2022).

[36] Sabt, M., Achemlal, M., and Bouabdallah, A. Trusted Execution Environment: What It is, and What It is Not. *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 1*. IEEE, 2015, pp. 57–64. DOI: 10.1109/Trustcom.2015.357. URL: https://doi.org/10.1109/Trustcom.2015.357.

[37] ARM. *ARM Security Technology — Building a Secure System using TrustZone® Technology*. Tech. rep. 2009. URL: https://documentation-service.arm.com/static/5f212796500e883ab8e74531?token= (visited on 05/04/2022).

[38] Enarx. *Getting Started > Enarx*. URL: https://enarx.dev/docs/Start/Enarx (visited on 07/27/2022).

[39] Hoyer, H. *From MacOS to Raspberry Pi — Extending the Enarx Development Platforms*. URL: https://blog.enarx.dev/backend-nil/ (visited on 07/27/2022).

[40] Paar, C. and Pelzl, J. *Understanding Cryptography - A Textbook for Students and Practitioners*. Springer, 2010. ISBN: 978-3-642-04100-6. DOI: 10.1007/978-3-642-04101-3. URL: https://doi.org/10.1007/978-3-642-04101-3.

[41] Smart, N. P. Physical Side-Channel Attacks on Cryptographic Systems. *Softw. Focus* 1.2 (2000), pp. 6–13. DOI: 10.1002/1529-7950(200012)1:2<6::AID-SWF10>3.0.CO;2-W. URL: https://doi.org/10.1002/1529-7950(200012)1:2%3C6::AID-SWF10%3E3.0.CO;2-W.

[42] Bhasin, S. and Mukhopadhyay, D. Fault Injection Attacks: Attack Methodologies, Injection Techniques and Protection Mechanisms - A Tutorial. *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*. Ed. by C. Carlet, M. A. Hasan, and V. Saraswat. Vol. 10076. Lecture Notes in Computer Science. Springer, 2016, pp. 415–418. DOI: 10.1007/978-3-319-49445-6_24. URL: https://doi.org/10.1007/978-3-319-49445-6_24.

[43] Boneh, D., DeMillo, R. A., and Lipton, R. J. On the Importance of Checking Cryptographic Protocols for Faults. *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*. Ed. by W. Fumy. Vol. 1233. Lecture Notes in Computer Science. Springer, 1997, pp. 37–51. DOI: 10.1007/3-540-69053-0_4. URL: https://doi.org/10.1007/3-540-69053-0_4.

[44] Aumüller, C., Bier, P., Fischer, W., Hofreiter, P., and Seifert, J.-P. Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures. *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*. Ed. by B. S. Kaliski Jr., Ç. K. Koç, and C. Paar. Vol. 2523. Lecture Notes in Computer Science. Springer, 2002, pp. 260–275. DOI: 10.1007/3-540-36400-5_20. URL: https://doi.org/10.1007/3-540-36400-5_20.

[45] Durumeric, Z., Kasten, J., Adrian, D., Halderman, J. A., Bailey, M., Li, F., Weaver, N., Amann, J., Beekman, J., Payer, M., and Paxson, V. The Matter of Heartbleed. *Proceedings of the 2014 Internet Measurement Conference, IMC 2014, Vancouver, BC, Canada, November 5-7, 2014*. Ed. by C. Williamson, A. Akella, and N. Taft. ACM, 2014, pp. 475–488. DOI: 10.1145/2663716.2663755. URL: https://doi.org/10.1145/2663716.2663755.

[46] The OpenSSL Project Authors. *OpenSSL Cryptography and SSL/TLS Toolkit*. URL: https://www.openssl.org/ (visited on 02/08/2022).

[47] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., and Hamburg, M. Meltdown: Reading Kernel Memory from User Space. *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. Ed. by W. Enck and A. P. Felt. USENIX Association, 2018, pp. 973–990. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp.

[48] Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., and Yarom, Y. Spectre Attacks: Exploiting Speculative Execution. *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1–19. DOI: 10.1109/SP.2019.00002. URL: https://doi.org/10.1109/SP.2019.00002.

[49] Aldaya, A. C., Brumley, B. B., ul Hassan, S., Pereida García, C., and Tuveri, N. Port Contention for Fun and Profit. *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 870–887. DOI: 10.1109/SP.2019.00066. URL: https://doi.org/10.1109/SP.2019.00066 .

[50] Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T. F., Yarom, Y., and Strackx, R. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. Ed. by W. Enck and A. P. Felt. USENIX Association, 2018, pp. 991–1008. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck.

[51] Lou, X., Zhang, T., Jiang, J., and Zhang, Y. A Survey of Microarchitectural Side-Channel Vulnerabilities, Attacks, and Defenses in Cryptography. *ACM Comput. Surv.* 54.6 (2021), 122:1–122:37. DOI: 10.1145/3456629. URL: https://doi.org/10.1145/3456629.

[52] Johnson, S. P. *Intel® SGX and Side-Channels*. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sgx-and-side-channels.html (visited on 02/17/2022).

[53] Aumasson, J.-P. *Serious Cryptography: A Practical Introduction to Modern Encryption*. USA: No Starch Press, 2017. ISBN: 1593278268.

[54] National Institute of Standards and Technology. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. Tech. rep. Special Publication 800-22, Revision 1a, April 30, 2010. Washington, D.C.: U.S. Department of Commerce, 2010. DOI: 10.6028/NIST.SP.800-22r1a.

[55] Khrennikov, A. Y. and Svozil, K. Quantum Probability and Randomness. *Entropy* 21.1 (2019), p. 35. DOI: 10.3390/e21010035. URL: https://doi.org/10.3390/e21010035.

[56] Shaw, G., Sivaram, S. R., and Prabhakar, A. Quantum Random Number Generator with One and Two Entropy Sources. *National Conference on Communications, NCC 2019, Bangalore, India, February 20-23, 2019*. IEEE, 2019, pp. 1–4. DOI: 10.1109/NCC.2019.8732222. URL: https://doi.org/10.1109/NCC.2019.8732222.

[57] Vokic, N., Milovancev, D., Pacher, C., Achleitner, M., Hübel, H., and Schrenk, B. Quantum RNG Integration in an NG-PON2 Transceiver. *Optical Fiber Communications Conference and Exhibition, OFC 2021, San Francisco, CA, USA, June 6-10, 2021*. IEEE, 2021, pp. 1–3. URL: https://ieeexplore.ieee.org/document/9489917.

[58] Kim, H.-I. and Jeon, J.-C. Quantum LFSR Structure for Random Number Generation Using QCA Multilayered Shift Register for Cryptographic Purposes. *Sensors*

22.9 (2022), p. 3541. DOI: 10.3390/s22093541. URL: https://doi.org/10.3390/s22093541.

[59] Abbott, A. A. and Calude, C. S. Von Neumann Normalisation of a Quantum Random Number Generator. *Comput.* 1.1 (2012), pp. 59–83. DOI: 10.3233/COM-2012-001. URL: https://doi.org/10.3233/COM-2012-001.

[60] Martínez, A. C., Solís, A., Díaz Hernández Rojas, R., U'Ren, A. B., Hirsch, J. G., and Pérez Castillo, I. Advanced Statistical Testing of Quantum Random Number Generators. *Entropy* 20.11 (2018), p. 886. DOI: 10.3390/e20110886. URL: https://doi.org/10.3390/e20110886.

[61] Maiti, A., Nagesh, R., Reddy, A., and Schaumont, P. Physical Unclonable Function and True Random Number Generator: A Compact and Scalable Implementation. *Proceedings of the 19th ACM Great Lakes Symposium on VLSI 2009, Boston Area, MA, USA, May 10-12 2009*. Ed. by F. Lombardi, S. Bhanja, Y. Massoud, and R. I. Bahar. ACM, 2009, pp. 425–428. DOI: 10.1145/1531542.1531639. URL: https://doi.org/10.1145/1531542.1531639.

[62] Amaki, T., Hashimoto, M., and Onoye, T. An Oscillator-Based True Random Number Generator with Jitter Amplifier. *International Symposium on Circuits and Systems (ISCAS 2011), May 15-19 2011, Rio de Janeiro, Brazil*. IEEE, 2011, pp. 725–728. DOI: 10.1109/ISCAS.2011.5937668. URL: https://doi.org/10.1109/ISCAS.2011.5937668.

[63] Ghandali, S., Holcomb, D. E., and Paar, C. Temperature-Based Hardware Trojan For Ring-Oscillator-Based TRNGs. *CoRR* abs/1910.00735 (2019). arXiv: 1910.00735. URL: http://arxiv.org/abs/1910.00735.

[64] Derlecki, M., Siwiec, K., Narczyk, P., and Pleskacz, W. A. Design of a True Random Number Generator Based on Low Power Oscillator with Increased Jitter. *22nd IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems, DDECS 2019, Cluj-Napoca, Romania, April 24-26, 2019*. IEEE, 2019, pp. 1–4. DOI: 10.1109/DDECS.2019.8724643. URL: https://doi.org/10.1109/DDECS.2019.8724643.

[65] Kumar, D., Anand, R., Singh, S. V., Misra, P. K., Srivastava, A., and Goswami, M. 0.4 mW, 0.27 pJ/bit True Random Number Generator Using Jitter, Metastability and Current Starved Topology. *IET Circuits Devices Syst.* 14.7 (2020), pp. 1001–1011. DOI: 10.1049/iet-cds.2019.0318. URL: https://doi.org/10.1049/iet-cds.2019.0318.

[66] Wang, X., Liang, H., Wang, Y., Yao, L., Guo, Y., Yi, M., Huang, Z., Qi, H., and Lu, Y. High-Throughput Portable True Random Number Generator Based on Jitter-Latch Structure. *IEEE Trans. Circuits Syst. I Regul. Pap.* 68.2 (2021), pp. 741–750. DOI: 10.1109/TCSI.2020.3037173. URL: https://doi.org/10.1109/TCSI.2020.3037173.

[67] National Institute of Standards and Technology. *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. Tech. rep. Special Publication 800-90A, Revision 1, June 24, 2015. Washington, D.C.: U.S. Department of Commerce, 2015. DOI: `10.6028/NIST.SP.800-90Ar1`.

[68] Matsumoto, M. and Nishimura, T. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Trans. Model. Comput. Simul.* 8.1 (1998), pp. 3–30. DOI: `10.1145/272991.272995`. URL: `https://doi.org/10.1145/272991.272995`.

[69] *random(4) — Linux Manual Page*. Mar. 2021. URL: `https://man7.org/linux/man-pages/man4/random.4.html` (visited on 04/19/2022).

[70] Fischer, V., Bernard, F., Bochard, N., and Varchola, M. Enhancing Security of Ring Oscillator-Based TRNG Implemented in FPGA. *FPL 2008, International Conference on Field Programmable Logic and Applications, Heidelberg, Germany, 8-10 September 2008*. IEEE, 2008, pp. 245–250. DOI: `10.1109/FPL.2008.4629939`. URL: `https://doi.org/10.1109/FPL.2008.4629939`.

[71] Yoo, S.-K., Karakoyunlu, D., Birand, B., and Sunar, B. Improving the Robustness of Ring Oscillator TRNGs. *ACM Trans. Reconfigurable Technol. Syst.* 3.2 (2010), 9:1–9:30. DOI: `10.1145/1754386.1754390`. URL: `https://doi.org/10.1145/1754386.1754390`.

[72] Markettos, A. T. and Moore, S. W. The Frequency Injection Attack on Ring-Oscillator-Based True Random Number Generators. *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*. Ed. by C. Clavier and K. Gaj. Vol. 5747. Lecture Notes in Computer Science. Springer, 2009, pp. 317–331. DOI: `10.1007/978-3-642-04138-9_23`. URL: `https://doi.org/10.1007/978-3-642-04138-9_23`.

[73] Bayon, P., Bossuet, L., Aubert, A., and Fischer, V. Fault Model of Electromagnetic Attacks Targeting Ring Oscillator-Based True Random Number Generators. *J. Cryptogr. Eng.* 6.1 (2016), pp. 61–74. DOI: `10.1007/s13389-015-0113-2`. URL: `https://doi.org/10.1007/s13389-015-0113-2`.

[74] *urandom(4) — Linux Manual Page*. URL: `https://linux.die.net/man/4/urandom` (visited on 05/01/2022).

[75] Kelsey, J., Schneier, B., Wagner, D. A., and Hall, C. Cryptanalytic Attacks on Pseudorandom Number Generators. *Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings*. Ed. by S. Vaudenay. Vol. 1372. Lecture Notes in Computer Science. Springer, 1998, pp. 168–188. DOI: `10.1007/3-540-69710-1_12`. URL: `https://doi.org/10.1007/3-540-69710-1_12`.

[76] Nitaj, A. and Rachidi, T. Factoring RSA Moduli with Weak Prime Factors. *Codes, Cryptology, and Information Security - First International Conference, C2SI 2015,*

*Rabat, Morocco, May 26-28, 2015, Proceedings - In Honor of Thierry Berger*. Ed. by S. El Hajji, A. Nitaj, C. Carlet, and E. M. Souidi. Vol. 9084. Lecture Notes in Computer Science. Springer, 2015, pp. 361–374. DOI: `10.1007/978-3-319-18681-8_29`. URL: `https://doi.org/10.1007/978-3-319-18681-8_29`.

[77] Nemec, M., Sýs, M., Svenda, P., Klinec, D., and Matyas, V. The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Ed. by B. Thuraisingham, D. Evans, T. Malkin, and D. Xu. ACM, 2017, pp. 1631–1648. DOI: `10.1145/3133956.3133969`. URL: `https://doi.org/10.1145/3133956.3133969`.

[78] Hegadi, R. S. and Patil, A. P. A Statistical Analysis on In-Built Pseudo Random Number Generators Using NIST Test Suite. *5th International Conference on Computing, Communication and Security, ICCCS 2020, Patna, India, October 14-16, 2020*. IEEE, 2020, pp. 1–6. DOI: `10.1109/ICCCS49678.2020.9276849`. URL: `https://doi.org/10.1109/ICCCS49678.2020.9276849`.

[79] Jenkins, B. *ISAAC: A Fast Cryptographic Random Number Generator*. URL: `http://www.burtleburtle.net/bob/rand/isaacafa.html` (visited on 06/21/2022).

[80] Kelsey, J., Schneier, B., and Ferguson, N. Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator. *Selected Areas in Cryptography, 6th Annual International Workshop, SAC'99, Kingston, Ontario, Canada, August 9-10, 1999, Proceedings*. Ed. by H. M. Heys and C. M. Adams. Vol. 1758. Lecture Notes in Computer Science. Springer, 1999, pp. 13–33. DOI: `10.1007/3-540-46513-8_2`. URL: `https://doi.org/10.1007/3-540-46513-8_2`.

[81] Ferguson, N., Schneier, B., and Kohno, T. *Cryptography Engineering - Design Principles and Practical Applications*. Wiley, 2010. ISBN: 978-0-470-47424-2. URL: `http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470474246.html`.

[82] IANIX. *ChaCha Usage & Deployment*. URL: `https://ianix.com/pub/chacha-deployment.html` (visited on 06/21/2022).

[83] Brown, R. G. *Dieharder: A Random Number Test Suite*. URL: `https://webhome.phy.duke.edu/~rgb/General/dieharder.php` (visited on 06/21/2022).

[84] L'Ecuyer, P. and Simard, R. J. TestU01: A C Library for Empirical Testing of Random Number Generators. *ACM Trans. Math. Softw.* 33.4 (2007), 22:1–22:40. DOI: `10.1145/1268776.1268777`. URL: `https://doi.org/10.1145/1268776.1268777`.

[85] Hurley-Smith, D. and Hernandez-Castro, J. Great Expectations: A Critique of Current Approaches to Random Number Generation Testing & Certification. *Security Standardisation Research - 4th International Conference, SSR 2018, Darm-*

*stadt, Germany, November 26-27, 2018, Proceedings*. Ed. by C. Cremers and A. Lehmann. Vol. 11322. Lecture Notes in Computer Science. Springer, 2018, pp. 143–163. DOI: 10.1007/978-3-030-04762-7_8. URL: https://doi.org/10.1007/978-3-030-04762-7_8.

[86] Soto, J. Statistical Testing of Random Number Generators. *In: Proceedings of the 22nd National Information Systems Security Conference*. 1999.

[87] Lee, C.-Y., Bharathi, K., Lansford, J., and Khatri, S. P. NIST-Lite: Randomness Testing of RNGs on an Energy-Constrained Platform. *39th IEEE International Conference on Computer Design, ICCD 2021, Storrs, CT, USA, October 24-27, 2021*. IEEE, 2021, pp. 41–48. DOI: 10.1109/ICCD53106.2021.00019. URL: https://doi.org/10.1109/ICCD53106.2021.00019.

[88] Petro, D. and Cecil, A. You're Doing IoT RNG. DEF CON 29. 2021. URL: https://bishopfox.com/blog/youre-doing-iot-rng.

[89] Arm Limited. *Mbed OS*. URL: https://os.mbed.com/mbed-os/ (visited on 06/22/2022).

[90] Zephyr Project. *Zephyr – An Operating System for IoT*. URL: https://www.zephyrproject.org/zephyr-an-operating-system-for-iot/ (visited on 06/22/2022).

[91] Contiki-NG. *Contiki-NG: The OS for Next Generation IoT Devices*. 2017. URL: https://github.com/contiki-ng/contiki-ng (visited on 06/22/2022).

[92] Heninger, N., Durumeric, Z., Wustrow, E., and Halderman, J. A. Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices. *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*. Ed. by T. Kohno. USENIX Association, 2012, pp. 205–220. URL: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/heninger.

[93] Vassilev, A. and Staples, R. Entropy as a Service: Unlocking Cryptography's Full Potential. *Computer* 49.9 (2016), pp. 98–102. DOI: 10.1109/MC.2016.275. URL: https://doi.org/10.1109/MC.2016.275.

[94] Kelsey, J. The New Randomness Beacon Format Standard: An Exercise in Limiting the Power of a Trusted Third Party. *Security Standardisation Research - 4th International Conference, SSR 2018, Darmstadt, Germany, November 26-27, 2018, Proceedings*. Ed. by C. Cremers and A. Lehmann. Vol. 11322. Lecture Notes in Computer Science. Springer, 2018, pp. 164–184. DOI: 10.1007/978-3-030-04762-7_9. URL: https://doi.org/10.1007/978-3-030-04762-7_9.

[95] Raikwar, M. and Gligoroski, D. SoK: Decentralized Randomness Beacon Protocols. *CoRR* abs/2205.13333 (2022). DOI: 10.48550/arXiv.2205.13333. arXiv: 2205.13333. URL: https://doi.org/10.48550/arXiv.2205.13333.

[96] Shumow, D. and Ferguson, N. On the Possibility of a Back Door in the NIST SP800-90 Dual EC PRNG. *Advances in Cryptology - CRYPTO 2007, 27th An-*

*nual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings, Rump session*. 2007. URL: `https://rump2007.cr.yp.to/15-shumow.pdf` (visited on 06/30/2022).

[97] Bernstein, D. J., Lange, T., and Niederhagen, R. Dual EC: A Standardized Back Door. *The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*. Ed. by P. Y. A. Ryan, D. Naccache, and J.-J. Quisquater. Vol. 9100. Lecture Notes in Computer Science. Springer, 2016, pp. 256–281. DOI: `10.1007/978-3-662-49301-4_17`. URL: `https://doi.org/10.1007/978-3-662-49301-4_17`.

[98] National Institute of Standards and Technology. *Entropy as a Service*. 2016. URL: `https://csrc.nist.gov/Projects/Entropy-as-a-Service` (visited on 07/01/2022).

[99] Rabin, M. O. Transaction Protection by Beacons. *J. Comput. Syst. Sci.* 27.2 (1983), pp. 256–267. DOI: `10.1016/0022-0000(83)90042-9`. URL: `https://doi.org/10.1016/0022-0000(83)90042-9`.

[100] National Institute of Standards and Technology. *Interoperable Randomness Beacons*. 2011. URL: `https://csrc.nist.gov/projects/interoperable-randomness-beacons` (visited on 06/28/2022).

[101] Bierhorst, P., Knill, E., Glancy, S., Zhang, Y., Mink, A., Jordan, S. P., Rommal, A., Liu, Y.-K., Christensen, B., Nam, S. W., Stevens, M. J., and Shalm, L. K. Experimentally Generated Randomness Certified by the Impossibility of Superluminal Signals. *Nat.* 556.7700 (2018), pp. 223–226. DOI: `10.1038/s41586-018-0019-0`. URL: `https://doi.org/10.1038/s41586-018-0019-0`.

[102] Kiayias, A., Russell, A., David, B., and Oliynykov, R. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*. Ed. by J. Katz and H. Shacham. Vol. 10401. Lecture Notes in Computer Science. Springer, 2017, pp. 357–388. DOI: `10.1007/978-3-319-63688-7_12`. URL: `https://doi.org/10.1007/978-3-319-63688-7_12`.

[103] Syta, E., Jovanovic, P., Kokoris-Kogias, E., Gailly, N., Gasser, L., Khoffi, I., Fischer, M. J., and Ford, B. Scalable Bias-Resistant Distributed Randomness. *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 444–460. DOI: `10.1109/SP.2017.45`. URL: `https://doi.org/10.1109/SP.2017.45`.

[104] Cascudo, I. and David, B. SCRAPE: Scalable Randomness Attested by Public Entities. *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings*. Ed. by D. Gollmann, A. Miyaji, and H. Kikuchi. Vol. 10355. Lecture Notes in Computer Sci-

ence. Springer, 2017, pp. 537–556. DOI: 10.1007/978-3-319-61204-1_27. URL: https://doi.org/10.1007/978-3-319-61204-1_27.

[105] Schindler, P., Judmayer, A., Stifter, N., and Weippl, E. R. HydRand: Efficient Continuous Distributed Randomness. *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 2020, pp. 73–89. DOI: 10.1109/SP40000.2020.00003. URL: https://doi.org/10.1109/SP40000.2020.00003.

[106] Cascudo, I. and David, B. ALBATROSS: Publicly AttestabLe BATched Randomness Based On Secret Sharing. *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III*. Ed. by S. Moriai and H. Wang. Vol. 12493. Lecture Notes in Computer Science. Springer, 2020, pp. 311–341. DOI: 10.1007/978-3-030-64840-4_11. URL: https://doi.org/10.1007/978-3-030-64840-4_11.

[107] Bhat, A., Shrestha, N., Luo, Z., Kate, A., and Nayak, K. RandPiper - Reconfiguration-Friendly Random Beacons with Quadratic Communication. *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*. Ed. by Y. Kim, J. Kim, G. Vigna, and E. Shi. ACM, 2021, pp. 3502–3524. DOI: 10.1145/3460120.3484574. URL: https://doi.org/10.1145/3460120.3484574.

[108] Das, S., Krishnan, V., Isaac, I. M., and Ren, L. Spurt: Scalable Distributed Randomness Beacon with Transparent Setup. *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 2502–2517. DOI: 10.1109/SP46214.2022.9833580. URL: https://doi.org/10.1109/SP46214.2022.9833580.

[109] League of Entropy. *Distributed Randomness Beacon*. 2019. URL: https://drand.love/ (visited on 07/01/2022).

[110] Ullah, I., Meratnia, N., and Havinga, P. J. M. Entropy as a Service: A Lightweight Random Number Generator for Decentralized IoT Applications. *2020 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2020, Austin, TX, USA, March 23-27, 2020*. IEEE, 2020, pp. 1–6. DOI: 10.1109/PerComWorkshops48775.2020.9156205. URL: https://doi.org/10.1109/PerComWorkshops48775.2020.9156205.

[111] Crypto4A. *Root of QAOS*. 2019. URL: https://crypto4a.com/products/root-of-qaos/ (visited on 07/26/2022).

[112] QNU Labs. *QOSMOS (Entropy as a Service)*. 2021. URL: https://www.qnulabs.com/qosmos-entropy-as-a-service/ (visited on 07/01/2022).

[113] Qrypt. *Entropy as a Service*. 2021. URL: https://docs.qrypt.com/eaas/ (visited on 07/01/2022).

[114] Quantropi. *SEQUR Quantum Entropy Services*. 2022. URL: https://www.quantropi.com/qispace/sequr/ (visited on 07/26/2022).

[115] QuintessenceLabs. *qRand 100 Quantum Entropy Enhancer*. 2021. URL: https://www.quintessencelabs.com/products#qrand-100 (visited on 10/03/2022).

[116] Haahr, M. *RANDOM.ORG: True Random Number Service*. 1998. URL: https://www.random.org (visited on 06/28/2022).

[117] Huttunen, H. *TUNI Narvi Cluster*. URL: https://tuni-itc.github.io/wiki/Technical-Notes/tuni-narvi-cluster/ (visited on 09/15/2021).

[118] Aalto Science-IT. *Triton Cluster*. URL: https://scicomp.aalto.fi/triton/ (visited on 01/31/2022).

[119] Helin, J. A. and Maisala, S. *Helsinki University HPC Environment User Guide*. URL: https://wiki.helsinki.fi/display/it4sci/HPC+Environment+User+Guide (visited on 01/31/2022).

[120] CSC – IT Center for Science. *LUMI*. URL: https://research.csc.fi/-/lumi (visited on 09/09/2022).

[121] CSC – IT Center for Science. *Puhti*. URL: https://research.csc.fi/-/puhti (visited on 09/09/2022).

[122] CSC – IT Center for Science. *Mahti*. URL: https://research.csc.fi/-/mahti (visited on 09/09/2022).

[123] CSC – IT Center for Science. *Pouta*. URL: https://docs.csc.fi/cloud/pouta/ (visited on 09/09/2022).

[124] CSC – IT Center for Science. *Rahti*. URL: https://rahti.csc.fi/ (visited on 09/09/2022).

[125] CSC – IT Center for Science. *Allas*. URL: https://research.csc.fi/-/allas (visited on 09/09/2022).

[126] Jahanzeb, M., Oh, S., and Fox, G. C. Evaluating ARM HPC Clusters for Scientific Workloads. *Concurr. Comput. Pract. Exp.* 27.17 (2015), pp. 5390–5410. DOI: 10.1002/cpe.3602. URL: https://doi.org/10.1002/cpe.3602.

[127] Yokoyama, D., Schulze, B., Borges, F., and Mc Evoy, G. V. The Survey on ARM Processors for HPC. *J. Supercomput.* 75.10 (2019), pp. 7003–7036. DOI: 10.1007/s11227-019-02911-9. URL: https://doi.org/10.1007/s11227-019-02911-9.

[128] Armejach, A., Brank, B., Cortina, J., Dolique, F., Hayes, T., Ho, N., Lagadec, P.-A., Lemaire, R., López-Paradís, G., Marliac, L., Moretó, M., Marcuello, P., Pleiter, D., Tan, X., and Derradji, S. Mont-Blanc 2020: Towards Scalable and Power Efficient European HPC Processors. *Design, Automation & Test in Europe Conference & Exhibition, Grenoble, France, February 1-5, 2021*. IEEE, 2021, pp. 136–141. DOI: 10.23919/DATE51398.2021.9474093. URL: https://doi.org/10.23919/DATE51398.2021.9474093.

[129] Mantovani, F., Garcia-Gasulla, M., Gracia, J., Stafford, E., Banchelli, F., Josep-Fabrego, M., Criado-Ledesma, J., and Nachtmann, M. Performance and Energy Consumption of HPC Workloads on a Cluster Based on ARM ThunderX2 CPU. *Future Gener. Comput. Syst.* 112 (2020), pp. 800–818. DOI: `10.1016/j.future.2020.06.033`. URL: `https://doi.org/10.1016/j.future.2020.06.033`.

[130] Pardos, V. S., Armejach, A., Suárez Gracia, D., and Moretó, M. On the Use of Many-Core Marvell ThunderX2 Processor for HPC workloads. *J. Supercomput.* 77.4 (2021), pp. 3315–3338. DOI: `10.1007/s11227-020-03397-6`. URL: `https://doi.org/10.1007/s11227-020-03397-6`.

[131] Abrahamsson, P., Helmer, S., Phaphoom, N., Nicolodi, L., Preda, N., Miori, L., Angriman, M., Rikkilä, J., Wang, X., Hamily, K., and Bugoloni, S. Affordable and Energy-Efficient Cloud Computing Clusters: The Bolzano Raspberry Pi Cloud Cluster Experiment. *IEEE 5th International Conference on Cloud Computing Technology and Science, CloudCom 2013, Bristol, United Kingdom, December 2-5, 2013, Volume 2*. IEEE Computer Society, 2013, pp. 170–175. DOI: `10.1109/CloudCom.2013.121`. URL: `https://doi.org/10.1109/CloudCom.2013.121`.

[132] D'Amore, M., Baggio, R., and Valdani, E. A Practical Approach to Big Data in Tourism: A Low Cost Raspberry Pi Cluster. *Information and Communication Technologies in Tourism 2015, ENTER 2015, Proceedings of the International Conference in Lugano, Switzerland, February 3 - 6, 2015*. Ed. by I. Tussyadiah and A. Inversini. Springer, 2015, pp. 169–181. DOI: `10.1007/978-3-319-14343-9_13`. URL: `https://doi.org/10.1007/978-3-319-14343-9_13`.

[133] Saffran, J., Garcia, G., Souza, M. A., Penna, P. H., Castro, M., Góes, L. F. W., and Freitas, H. C. A Low-Cost Energy-Efficient Raspberry Pi Cluster for Data Mining Algorithms. *Euro-Par 2016: Parallel Processing Workshops - Euro-Par 2016 International Workshops, Grenoble, France, August 24-26, 2016, Revised Selected Papers*. Ed. by F. Desprez, P.-F. Dutot, C. Kaklamanis, L. Marchal, K. Molitorisz, L. Ricci, V. Scarano, M. A. Vega-Rodríguez, A. L. Varbanescu, S. Hunold, S. L. Scott, S. Lankes, and J. Weidendorfer. Vol. 10104. Lecture Notes in Computer Science. Springer, 2016, pp. 788–799. DOI: `10.1007/978-3-319-58943-5_63`. URL: `https://doi.org/10.1007/978-3-319-58943-5_63`.

[134] Srinivasan, K., Chang, C.-Y., Huang, C.-H., Chang, M.-H., Sharma, A., and Ankur, A. An Efficient Implementation of Mobile Raspberry Pi Hadoop Clusters for Robust and Augmented Computing Performance. *J. Inf. Process. Syst.* 14.4 (2018), pp. 989–1009. URL: `http://www.jips-k.org/q.jips?cp=pp%5C&pn=588`.

[135] An, J., Park, S., and Ihm, I. Construction of a Flexible and Scalable 4D Light Field Camera Array Using Raspberry Pi Clusters. *Vis. Comput.* 35.10 (2019), pp. 1475–1488. DOI: `10.1007/s00371-018-1512-z`. URL: `https://doi.org/10.1007/s00371-018-1512-z`.

[136] Hosny, K. M., Magdi, A., Lashin, N. A., El-Komy, O., and Salah, A. Robust Color Image Watermarking Using Multi-Core Raspberry Pi Cluster. *Multim. Tools Appl.* 81.12 (2022), pp. 17185–17204. DOI: 10.1007/s11042-022-12037-5. URL: https://doi.org/10.1007/s11042-022-12037-5.

[137] Mappuji, A., Effendy, N., Mustaghfirin, M., Sondok, F., Yuniar, R. P., and Pangesti, S. P. Study of Raspberry Pi 2 Quad-core Cortex A7 CPU Cluster as a Mini Supercomputer. *CoRR* abs/1612.07128 (2016). arXiv: 1612.07128. URL: http://arxiv.org/abs/1612.07128.

[138] Cicirello, V. A. Design, Configuration, Implementation, and Performance of a Simple 32 Core Raspberry Pi Cluster. *CoRR* abs/1708.05264 (2017). arXiv: 1708.05264. URL: http://arxiv.org/abs/1708.05264.

[139] Gupta, N., Brandt, S. R., Wagle, B., Wu, N., Kheirkhahan, A., Diehl, P., Baumann, F. W., and Kaiser, H. Deploying a Task-Based Runtime System on Raspberry Pi Clusters. *5th IEEE/ACM International Workshop on Extreme Scale Programming Models and Middleware, ESPM2@SC 2020, Atlanta, GA, USA, November 11, 2020*. IEEE, 2020, pp. 11–20. DOI: 10.1109/ESPM251964.2020.00007. URL: https://doi.org/10.1109/ESPM251964.2020.00007.

[140] Hawthorne, D., Kapralos, M. P., Blaine, R. W., and Matthews, S. J. Evaluating Cryptographic Performance of Raspberry Pi Clusters. *2020 IEEE High Performance Extreme Computing Conference, HPEC 2020, Waltham, MA, USA, September 22-24, 2020*. IEEE, 2020, pp. 1–9. DOI: 10.1109/HPEC43674.2020.9286247. URL: https://doi.org/10.1109/HPEC43674.2020.9286247.

[141] Cilloni, T., Cai, X., Fleming, C., and Li, J. Understanding and Detecting Majority Attacks. *2nd IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS 2020, Oxford, UK, August 3-6, 2020*. Ed. by J. Xu, S. Schulte, P. Ruppel, A. Küpper, and D. Jadav. IEEE, 2020, pp. 11–21. DOI: 10.1109/DAPPS49028.2020.00002. URL: https://doi.org/10.1109/DAPPS49028.2020.00002.

[142] Höller, T., Roland, M., and Mayrhofer, R. Analyzing Inconsistencies in the Tor Consensus. *iiWAS2021: The 23rd International Conference on Information Integration and Web Intelligence, Linz, Austria, 29 November 2021 - 1 December 2021*. Ed. by E. Pardede, M. Indrawan-Santiago, P. D. Haghighi, M. Steinbauer, I. Khalil, and G. Kotsis. ACM, 2021, pp. 485–494. DOI: 10.1145/3487664.3487793. URL: https://doi.org/10.1145/3487664.3487793.

[143] Raspberry Pi Foundation. *Raspberry Pi Documentation*. URL: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html (visited on 07/28/2022).

[144] *systemd(1) — Linux Manual Page*. URL: https://man7.org/linux/man-pages/man1/init.1.html (visited on 09/07/2022).

[145] Devuan. *Devuan ASCII 2.0.0 Stable Release*. URL: `https://www.devuan.org/os/announce/ascii-stable-announce-060818` (visited on 11/04/2022).

[146] Perrin, G. *Download FreeBSD*. URL: `https://www.freebsd.org/where/` (visited on 10/10/2022).

[147] Cerdeira, D., Santos, N., Fonseca, P., and Pinto, S. SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-Assisted TEE Systems. *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 2020, pp. 1416–1432. DOI: `10.1109/SP40000.2020.00061`. URL: `https://doi.org/10.1109/SP40000.2020.00061`.

[148] Fei, S., Yan, Z., Ding, W., and Xie, H. Security Vulnerabilities of SGX and Countermeasures: A Survey. *ACM Comput. Surv.* 54.6 (2021), 126:1–126:36. DOI: `10.1145/3456631`. URL: `https://doi.org/10.1145/3456631`.

[149] Aumasson, J.-P. Too Much Crypto. *IACR Cryptol. ePrint Arch.* (2019), p. 1492. URL: `https://eprint.iacr.org/2019/1492`.

[150] Hoyer, H. *mini_http*. 2017. URL: `https://github.com/haraldh/mini_http` (visited on 11/03/2022).