Dissertation Type: enterprise

DEPARTMENT OF COMPUTER SCIENCE

"Laboratory free" side-channel analysis

James Webb

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Thursday 9th May, 2019

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MEng in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

James Webb, Thursday 9th May, 2019

Contents

List of Figures

Executive Summary

In recent years, IoT devices have become increasingly popular in both consumer and corporate markets. Many of these IoT devices communicate sensitive data over the Internet through the use of cryptographic algorithms. However, side-channel attacks on these systems present a significant threat if the implementation of a cryptographic algorithm is not resistant to such attacks. Testing resilience against side-channels is often a time consuming and costly process due to the knowledge and equipment required to perform analysis in a laboratory environment. This project presents a "laboratory free" approach to side-channel analysis and provides a way for anyone from lone developers through to large scale corporations to perform side-channel analysis on cryptographic software, quickly and easily with very little cost.

Below is an itemisation of the main deliverables and achievements surrounding this project:

- I successfully applied for $2000 in AWS research credits which fully funded the cloud-based element of the project.
- I spent over 250 hours designing and implementing a highly available and performant cloud-based system specialised in handling side-channel related tasks.
- I spent over 70 hours integrating an external identity provider into a cloud-based storage system to facilitate the ability for segregated storage between users.
- I developed an API and web server written in Python which was designed to manage the interaction between users and the cloud-based environment.
- I assessed the effect of increasing machine resources and implementing distributed multiprocessing in relation to the runtime of side-channel analysis tasks in a cloud-based environment.
- I designed and implemented a way for laboratory-based acquisition equipment to dynamically receive side-channel jobs requested by users and created a way for results to be securely uploaded to the aforementioned storage system, allowing for further analysis at a later date.

Supporting Technologies

Infrastructure

- Amazon Web Services was used to provide the entire cloud-based environment for the project. https://aws.amazon.com/

  The following services within AWS were utilised:

  - **DynamoDB** - provided the core database infrastructure for both acquisition and analysis jobs.
  - **S3** - acted as the storage system for all results obtained from side-channel related tasks.
  - **EC2** - provided the infrastructure required for analysis tasks.
  - **ECS** - handled the scaling and management of Docker containers which formed the API and web server.
  - **ALB** - managed routing of web requests between available downstream servers and performed functionality such as TLS.
  - **IAM** and **Cognito** - facilitated the ability to create a segregated storage system.
  - **Route 53** - acted as the authoritative DNS nameserver for project resources.
  - **SQS** and **Lambda** - handled the post processing of side-channel acquisition results.
  - **SES** - provided the ability to send out automated emails to users.

- Auth0 was used as the identity provider which handled authentication related tasks for the entire platform. https://auth0.com/

Hardware

- The side-channel acquisition servers comprising of a PicoScope, SCALE development board and a connected workstation provided the ability to capture power traces.
  - **PicoScope** - extracts power consumption data from the SCALE development board. https://www.picotech.com/products/oscilloscope
  - **SCALE** - executes target cryptographic implementation. https://github.com/danpage/scale

Software

- The API and web server were written using Python Flask along with certain third-party libraries:
  - **Flask** - formed the base web server acting as the framework enabling the website and API server to operate. http://flask.pocoo.org/
  - **Waitress** - used as the WSGI server for the Flask application. https://pypi.org/project/waitress/
  - **PyJWT** - package used to assist with the verification of JSON Web Tokens returned by Auth0. https://pypi.org/project/PyJWT/
  - **Bootstrap** - open source CSS framework used to provide an elegant website. https://getbootstrap.com/
  - **Boto 3** - AWS SDK for Python. https://pypi.org/project/boto3/
  - **jsonschema** - used to validate the payload of API requests submitted by users. https://pypi.org/project/jsonschema/

Notation and Acronyms

AES	:	Advanced Encryption Standard
IoT	:	Internet of Things
CAPS	:	Certified Assisted Products
NCSC	:	National Cyber Security Centre
SME	:	Small/Medium Enterprise
SCAasS	:	Side-Channel Analysis as a Service
IaaS	:	Infrastructure as a Service
PaaS	:	Platform as a Service
SaaS	:	Software as a Service
CI	:	Continuous Integration
CD	:	Continuous Deployment
NIST	:	National Institute of Standards and Technology
CTF	:	Capture The Flag
API	:	Application Programming Interface
DPA	:	Differential Power Analysis
CPA	:	Correlation Power Analysis
CSV	:	Comma Separated Values
DNS	:	Domain Name System
OS	:	Operating System
SAN	:	Storage Area Network
CPU	:	Central Processing Unit
RAM	:	Random Access Memory
AWS	:	Amazon Web Services
SQS	:	Simple Queue Service
RDBMS	:	Relational Database Management System
ACID	:	Atomicity, Consistency, Isolation and Durability
TCP	:	Transmission Control Protocol
HTTP	:	Hypertext Transfer Protocol
TLS	:	Transport Layer Security
ECR	:	Elastic Container Registry
ECS	:	Elastic Container Service
SCALE	:	Side-Channel Attack Lab. Exercises
USB	:	Universal Serial Bus
SMA	:	SubMiniature version A
S3	:	Simple Storage Service
SCARV	:	Side-Channel Hardened RISC-V
WSGI	:	Web Server Gateway Interface
FQDN	:	Fully Qualified Domain Name
ALB	:	Application Load Balancer
OSI	:	Open Systems Interconnection
ACM	:	AWS Certificate Manager
IP	:	Internet Protocol
JSON	:	JavaScript Object Notation
SES	:	Simple Email Service
SMTP	:	Simple Mail Transfer Protocol
HTML	:	Hypertext Markup Language

DKIM	:	DomainKeys Identified Mail
SPF	:	Sender Policy Framework
DMARC	:	Domain Message Authentication Reporting and Conformance
IAM	:	Identity and Access Management
IDP	:	Identity Pool
EFS	:	Elastic File System
SSH	:	Secure Shell
SCP	:	Secure Copy Protocol
	:	
$GF(q)$	:	A finite field with q elements
XOR	:	Bitwise exclusive-or function
$HW(x)$	:	the Hamming weight of x

Acknowledgements

I would like to thank my supervisor, Dr Dan Page, for providing advice and support throughout the process, and Amazon Web Services for funding the project in the form of research credits on their platform. In addition, I would like to thank Luke Mather from Cerberus Laboratories for testing out the platform and everyone who provided feedback on the project at the poster day.

Chapter 1

Contextual Background

1.1 Introduction

Side-channel analysis describes any attack vector aimed towards an implementation of an algorithm as opposed to the algorithm specification itself. For example, a side-channel attack on the well known Advanced Encryption Standard (AES) [6] would target vulnerabilities pertaining to a physical device running an implementation of AES as opposed to focusing on the AES specification itself.

The premise behind performing such attacks focuses on extracting information from a device that is inadvertently leaking sensitive data. These leaks could arise in a variety of formats which include; power consumption data, electromagnetic radiation and time taken to perform specific operations. It is important to also understand that this data is often not directly leaked, but carefully extracted by an adversary via the use of specialist tools or techniques.

Upon obtaining sets of sensitive data, a variety of analysis techniques can then be performed to partially or fully recover critical pieces of information relating to the security of a cryptosystem. For example, a symmetric key used to encrypt a message.

1.2 Problem Definition

Within the security sector, side-channels are extremely important, more often than not there aren't enough resources dedicated to reducing the possibility of side-channel attacks. Instead, much more emphasis is placed on ensuring that the underlying specification of a system is secure rather than a particular implementation of it. From a developer standpoint, this can often result in one adopting a secure specification, but implementing it in an insecure manner due to either lack of knowledge or awareness of the consequences. Subsequently, if a product or service with these characteristics is released to the world with a large uptake, the ramifications can quickly become severe and far more resources then have to be spent addressing the issue retrospectively as opposed to initially tackling the issue head-on during the development phase.

If an easy to use system were to exist that allowed for the detection of certain vulnerabilities within the codebase prior to the release, issues could be fixed during the development phase for a fraction of the cost of doing so retrospectively.

1.2.1 Trends in Technology

Technology is shifting towards systems with increased connectivity and complexity. These systems heavily rely on the ability of devices within the system to perform security and identity critical tasks. Problems can often arise when such systems include Internet of Things (IoT) devices which are notorious for security issues as they are often built on embedded devices that don't cater well for cryptographic operations [7]. Consequently, such systems can easily be exposed to side-channel attacks if the IoT device is placed in an adversarial environment. Therefore, the security of the whole system can be compromised if the IoT device is not secure against side-channel attacks.

1.2.2 Certification

One of the ways to prove that a device is resistant to side-channel analysis is by going through a certification process. Certifications such as the Certified Assisted Products (CAPS) [8] offered by the UK National Cyber Security Centre (NCSC) is one of many. However, such certification processes can be lengthy and often only review a small number of products meeting strict criteria with respect to their real-world application.

There is a growing need throughout the security sector to certify cryptographic software and hardware. Both consumers and businesses using cryptography are becoming more aware of the dangers associated with insecure systems. It is for this reason that many privately run analysis companies have evolved to tackle this issue. One of which is Riscure Labs [9], they offer side-channel analysis services to businesses wishing to certify their hardware and software to ensure prospective clients that a system conforms to a specific well-known standard. However, like the CAPS certification process, Riscure's system is by no means instant. At the time of writing this document, no system exists to allow developers or small teams to perform side-channel analysis on software and hardware quickly.

1.2.3 Total Cost of Ownership

Arguably one of the reasons why so many systems are vulnerable to side-channel attacks is due to the cost of performing side-channel analysis. If one wishes to set up a laboratory for side-channel analysis, the monetary cost of doing so could be high. Specialist equipment needs to be procured such as oscilloscopes, device boards and all of the necessary components to integrate a side-channel testing environment into an existing system. Needless to say, this integration task is a project in itself. Given this fact, it is no wonder that such a vast number of companies and individuals often ignore this essential process of development.

1.3 Solution

1.3.1 Laboratory Free Approach

The drawback of a high total cost of ownership primarily stems from the fact that a laboratory setup is imperative for side-channel analysis. If one is able to eliminate the laboratory setup from the side-channel analysis and still obtain the same results, then the cost of performing such testing instantly becomes more affordable for developers and small or medium-sized enterprises (SMEs).

An appropriate analogy is that of the evolution of cloud computing. Traditionally, companies and individuals had to go through the process of co-location within a data centre in order to run production level IT systems. This meant that any entity wishing to do this suffered yet again the barrier of cost of ownership, this time in the form of purchasing servers and renting space within a data centre to access vital resources. These resources include services such as reliable power and stable networking. This barrier limits the development of novel technologies to large established companies who can afford to allocate resources to new projects. However, with the advent of cloud computing, this has changed. Cloud services at their most basic form offer a service denoted as Infrastructure as a Service (IaaS). This is the renting of bare metal hardware upon which various applications can be run. This removes the upfront cost of purchasing servers and also means that truly anyone can have a production level system setup in minutes.

Whilst this sounds like a great advance in technology, it is important to understand how cloud providers make money. The simple way to demonstrate this is to scale things down to one machine. If one assumes that a cloud provider has purchased one machine for £1095 and that standard depreciation rules apply meaning that the product will be worthless after 3 years, the machine has cost the cloud provider £1 per day for 3 years. Now assume that a prospective customer wishes to run an application on such a machine for a limited time of 1 hour, but does not wish to purchase the machine up front. They subsequently approach the cloud provider declaring interest in renting the machine for an hour. The cloud provider calculates that it costs approximately £0.05 per hour and therefore decides to charge the customer £0.10 per hour. The customer is thrilled that they are able to perform their development tasks for £0.10 and the cloud provider is also thrilled as they have made over 100% profit!

Therefore if one applies the same principles of cloud computing to side-channel analysis, it may be possible to provide a service that boasts the same results as traditional laboratory style analysis, but instead with a zero total cost of ownership and near instant results. This "side-channel analysis as a service" (SCAaaS) paradigm would be far more attractive to developers and SMEs for this reason. Furthermore, one could argue that if such a cost-effective service existed, there would be no need for

developers of cryptographic software to omit this vital analysis phase. Consequently, this type of service could vastly improve the security of many implementations of cryptographic algorithms deployed in the world.

1.3.2 Use Cases

Continuous Integration

In recent years, continuous integration (CI) services have become popular. They provide a platform for the developer to perform automated testing on a code base in response to certain events such as new Git commits to the code base repository. During CI testing, any aberrant behaviour is detected and flagged to the developer. An example of such a service is Travis CI [10], which is hosted in the cloud and is a perfect example of CI as a service.

"Laboratory free" side-channel analysis could be morphed into a solution which provides a form of outsourced CI, but is solely geared towards side-channel analysis tasks. Such a solution would be suitable for a lone cryptographic developer without access to equipment or the necessary expertise to perform side-channel analysis by themselves. Further to this, having a system which reports on the overall security of a piece of software at regular intervals would undoubtedly be useful for teams as they can directly see the impact certain code changes have on the overall security of the software.

Cryptographic Design Processes

In cryptography, when a new standard is drafted, various cryptographic algorithms are offered up by companies, academic institutions and individuals to compete against each other for the prize of their algorithm being adopted as the new standard. A notable example is that of AES; to which many submissions were received and tested against each other until eventually a cipher called Rijndael [11] was chosen as the winner.

A side-channel analysis service such as the one proposed may be extremely useful in this context as an effort to provide experimental consistency and openness in cryptographic competitions. A service such as this would provide a fair and unbiased comparison of all entries and help to inform an eventual decision as to which entry is regarded as the winner.

Recently, the National Institute for Standards and Technology (NIST) held a competition for lightweight cryptography algorithms [12]. In Section 4.2 of the submission requirements [13], it states that in reference to the resistance of side-channel attacks, "the ability to provide it easily and at low cost is highly desired" and "side-channel resistance may be necessary in some applications". Both of these statements reinforce the fact that a laboratory free side-channel analysis system could prove useful when testing submissions for such competitions.

Education

In many companies and academic institutions around the world, competitions are held for software development. The security-related events usually come under the category of capture the flag (CTF), whereby the goal is for a competitor to extract sensitive information usually in the form of a key or phrase from a series of challenges set by the event holder. These types of games would suit this particular form of side-channel analysis extremely well. One can imagine competitors designing a side-channel attack against a pre-set cryptographic software implementation. The goal could then be to use the cloud-based side-channel analysis framework to attack the pre-defined software implementation. The winner would then be regarded as the group or individual that is able to recover the largest amount of sensitive data from the target implementation. This holds two advantages; firstly, it allows for greater awareness to be spread with respect to the importance of side-channel analysis. Secondly, the best attacks produced by the competition can be added to the analysis system to test real application software candidates against.

It is also possible to invert the perspective of such a challenge. Instead of attacking a pre-set implementation of an algorithm, one could instead design a competition which asks competitors to produce a piece of cryptographic software that is resilient to the battery of attacks and analysis tools that will be run against it. The group or individual with the highest level of resilience to all attacks would be declared as the winner.

Both of these styles of competition would not be possible in a traditional laboratory-based side-channel analysis environment due to the time-sensitive nature of such competitions and more simply, the lack of such a laboratory at most venues.

Independent Standard

A slightly more indirect use case of this particular side-channel analysis system is the ability of it to be considered as an independent testing body. If a service is created that is both quick and affordable as well as accurate then there is no reason why such a service could not be used as a recommended certification body for cryptographic applications prior to their deployment in the real world. For example, a security score could be assigned to each application tested, thus illustrating neatly the severity of any vulnerabilities that may exist. This greatly helps end users understand how a particular device or application fairs against a similar competitor without having to have an in-depth knowledge of side-channel analysis. With enough uptake, a system such as this could become the de facto security standard whereby the security score of a potential cryptographic application needs to exceed a particular threshold for it to be deployed by a vendor into their ecosystem.

1.3.3 Existing Work

eshard esDynamic Platform

A company called eshard has released a product called the esDynamic Platform [14]. This is a unified interface for performing side-channel analysis tasks. It allows one to perform a variety of attacks such as DPA and CPA attacks. Further to this, the product also integrates with writing software such as LaTeX, allowing academics to integrate results neatly into reports. However, the product does suffer some drawbacks. The first of which is that it does not seem to offer a side-channel analysis as a service approach whereby the product only integrates with existing laboratory equipment. This means that the total cost of ownership issue still remains. Further to this, the openness with respect to some of the data formats used could be questionable. For example, there is no reference on the site to any format used for trace sets which could indicate that it is proprietary. Unfortunately, this could raise the issue of vendor lock-in from an end user perspective.

NewAE Technology Challenges

CHES [15] is a CTF competition, but geared solely towards side-channel analysis. This system offers competitors the opportunity to submit both cryptographic implementations and attacks. It has a similar high-level goal to that of "laboratory free" side-channel analysis, but fails to offer any automation. Instead it relies on users attacking the submitted implementations using their own equipment in a laboratory. Subsequently, due to this, it fails to mitigate the issue of total cost of ownership.

DPA Contest

The current edition of DPA Contest [16] instructs participants to submit attacks against AES implementations targeted at smart cards. The general premise of such a contest is to collect a set of attacks against a specific cryptographic implementation and use the results to improve the security of future implementations. Competitors are given trace sets to analyse and recover sensitive information from. As such, it is somewhat linked to "laboratory free" side-channel analysis, but is solely geared towards DPA style attacks. Further to this, the service offers no ability for users to generate trace sets of their own cryptographic implementations or the option to run the attacks collected on generated trace sets.

1.4 Project Definition

1.4.1 Overall Flow

Figure 1.1 illustrates the intended operation of the proposed service as a high-level flowchart. Clients will fundamentally be split into two types; acquisition and analysis. Acquisition clients are only concerned with retrieving power consumption traces from cryptographic software executing on specific pieces of hardware. On the other hand, analysis clients are focused on analysing power consumption data using DPA or CPA style analysis tools.

As the project is explained in more detail in subsequent sections, it should become clear as to which components of the system are involved in each section of Figure 1.1.

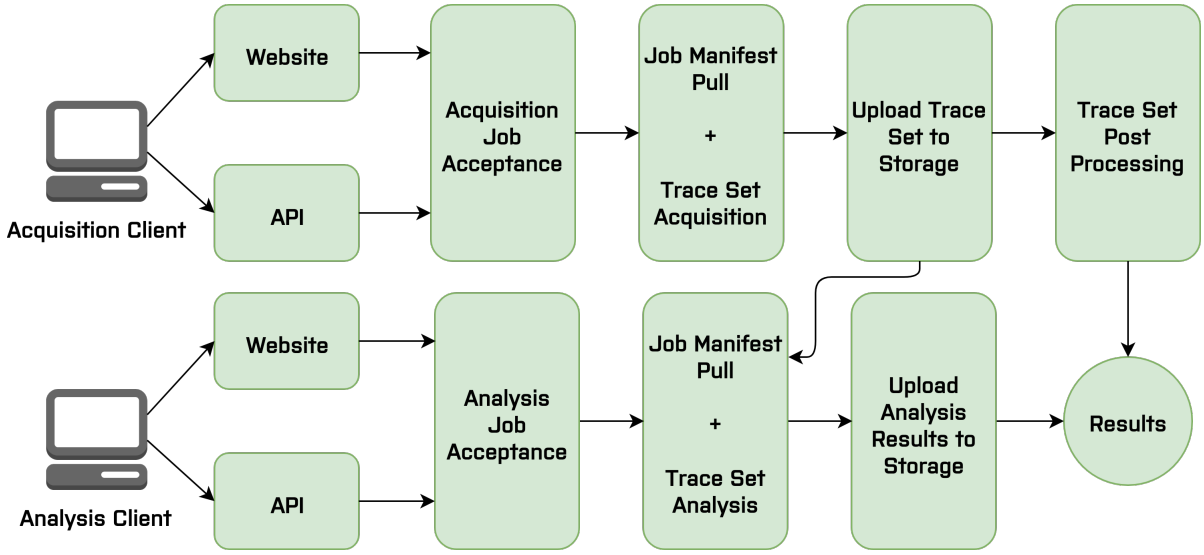

Figure 1.1: Flowchart depicting high level service operation.

1.4.2 Aims and Objectives

Development

D1. Develop an auto-scaling, highly available cloud-based framework which includes features suitable for side-channel analysis tasks.

D2. Extend this framework further with a website and API system allowing for the submission of side-channel acquisition jobs. These jobs will allow for leakage data to be captured from a target board running a particular piece of software submitted by a user to the system.

D3. Integrate a queuing system into an existing side-channel analysis laboratory environment to allow for the provisioning of acquisition jobs to hardware. Within this task, the results of the capture should be uploaded to a secure cloud environment to allow for further analysis.

D4. Extend the framework further again by adding the ability to submit analysis jobs to the system. These analysis jobs should reference some acquisition data previously captured.

D5. Develop the ability for side-channel acquisition data to be analysed within a cloud based environment.

Research

R1. Explore possibilities of improving the efficiency of side-channel analysis tasks within a cloud based environment.

R2. Evaluate the economic cost of side-channel analysis within a cloud based environment.

1.4.3 Challenges

Building a Suitable Cloud Framework

One of the core challenges around this task will be creating a framework that can successfully support all of the core tasks that surround side-channel analysis.

One of the key aspects will be a unified storage system that can store both acquisition data in the form of trace sets and analysis data in the form of graphs and output logs. The system will need to ensure that this data is readily accessible, but only to authorised users. Exploring how to do this in an efficient and cost-effective way will be required.

In some of the existing solutions, the lack of openness with respect to acquisition data formats is concerning. Therefore, the system should have the ability to support and convert data between an array of widely accepted formats. This would include formats like Python pickle and CSV, but also an effort

should be made to include a common trace set format. One such format is called TRS [17] and was designed by Riscure Labs. Implementing such a conversion mechanism between these formats will be difficult, but an attractive feature nonetheless.

In addition to the above, the entire system must be able to scale with demand automatically. Computing and storage resources must be dynamically provisioned to ensure that the service does not degrade at any point in time.

Interaction with Traditional Laboratory Hardware

Clearly certain aspects of the system cannot be ported to the cloud. For example, cloud providers do not provide an oscilloscope service to connect to hosted machines to collect power consumption data. Due to factors such as these, a laboratory-based component must still be included to collect side-channel acquisition data. However, it will need to be controlled by the cloud-based infrastructure in order to allow users to submit their jobs and have the results available in real-time. Creating a decoupling mechanism between these two environments will be both essential and challenging to ensure a smooth service operation. It is critical that the laboratory equipment is not overloaded with user tasks so middleware must be created to dispatch work to the remote laboratory in a sensible manner.

Integration of an Analysis Framework

One integral part of the side-channel analysis as a service framework is the ability to perform side-channel analysis tasks quickly in the cloud. One should have the option to reference an acquisition task as an input into some attack within the cloud framework's analysis database to get results for a particular cryptographic implementation. In order to support a wide array of analysis tasks, it makes sense to integrate an existing fully featured analysis framework. One such framework is Jlsca [18], which is a side-channel analysis toolkit written in Julia based off of a project called Pysca [19]. Jlsca contains a plethora of analysis tooling making it perfect as an analysis engine for this project. The challenge will be integrating this toolkit with the cloud framework such that it will accept results from completed acquisition jobs as input data. Further to this, in order to ensure efficient computation of analysis tasks, research will need to be done on parallelising this toolkit across multiple host machines.

1.4.4 Scope

The overall aim of this project is to demonstrate the possibility of a solution in the spirit of side-channel analysis as a service. That being said, certain embellishments will be omitted as they are deemed somewhat trivial additions to the core framework that will be created. Subsequent paragraphs below explain the remit of the project at a high level.

The first and most obvious element of the project is the API and website design. Users must have the ability to submit jobs for both acquisition and analysis. Further to this, these submissions must be kept private to each user and results must be readily available upon completion. To facilitate this, a federated storage system will be created that stores results in a user segregated fashion whilst enabling availability at all times. Clearly, in order to provide the segregation, an identity provider will have to be integrated into the system which is responsible for authenticating users onto the platform.

The overall functionality of the platform will heavily rely on its underlying ability to auto scale on demand. Essential services will have this embedded from the ground up to ensure the overall service is highly available. In addition, the middleware mentioned earlier will provide a buffer between any constrained lab-based resources.

Only the acquisition based tasks will be performed in the remote laboratory as they require specific boards and equipment such as oscilloscopes. Therefore, the specification of this component in relation to the cloud-based analysis runtime will be more fully featured. This will be done to prove that the concept of remote acquisitions is entirely feasible. Consequently, there will be a variety of customisations available in this stage to allow users to take advantage of a remote trace set acquisition system. Whereas, the analysis system will likely be non-customisable, with a fixed set of analysis tools which can be extended easily in the future.

To ensure that the analysis segment of the project is successful, it is essential that time is spent on researching ways to maximise performance for the least monetary cost. As analysis computation will be performed solely in the cloud, a variety of techniques should be demonstrated including differences in hosted machine type through to multiprocessing techniques across a cluster of machines.

1.4.5 Organisation

This report is divided into three main topics which at each successive stage introduce areas of the project in more detail before finally giving an overall evaluation on the system in full based on the original aims and objectives listed in Section 1.4.2.

1. **Technical Background**

   This section outlines the technical systems and frameworks used in the design and implementation of this project. This section assumes that the reader has a well rounded knowledge of both cloud computing and side-channel analysis. Technologies integral to the operation of the project are introduced and explained in detail.

2. **Project Execution**

   This section details explicitly how each of the technologies mentioned in the technical background inter-operate to enable the project to function. This includes the design, implementation and testing of various aspects of the overall system. In relation to Section 1.4.2, this chapter focuses purely on the developmental goals.

3. **Project Evaluation**

   Here the project is evaluated retrospectively from a functional and economic standpoint. Furthermore, user testing data is also shown to provide a real world account of the system created. In relation to Section 1.4.2, this chapter focuses on the research aspects of the project by evaluating the effectiveness of different approaches towards side-channel analysis in the cloud.

Chapter 2

Technical Background

2.1 Cloud Computing Technologies

Cloud computing is at the heart of this project. Cloud computing platforms allow any developer or large enterprise to develop and deploy production-level services quickly, cheaply and easily. In order to understand some of the fundamentals of cloud computing, it is first important to define the three main cloud service offerings at a high level before inspecting lower level components at a finer granularity.

1. **Infrastructure as a Service**

   This service was loosely defined in the previous chapter and is responsible for providing clients with bare metal machine hardware that can be customised further. This is the most basic service offering from cloud providers due to the client managing everything from the operating system through to the application software. IaaS usually has many purchasing options available to clients which are priced based on the type of resource and the urgency of the request. For example, one of the most economical ways to purchase IaaS services from a cloud provider is by reserving a particular machine for a set period of time. In addition, one can reserve an entire machine or opt to share the tenancy with others for a cheaper rate. This is known as virtualisation and will be discussed further in a separate section. Common examples of IaaS deployed today include services such as AWS EC2 and Google Compute Engine.

2. **Platform as a Service**

   This offering is one level higher than IaaS from a client's perspective in that the cloud provider manages more of the underlying system. One no longer has to manage the operating system and common components required for application development as in IaaS. The idea behind platform as a service is that developers can have a single unified platform which contains all the necessary components required to build an application. A popular PaaS paradigm is that of a LAMP stack offering. This provides users with a Linux operating system, Apache Web Server, MySQL database and the PHP programming language. If a cloud provider makes this available out of the box for developers, then a large portion of time spent setting these components up can be better spent on core application development, thus improving productivity. Common examples of service offerings include AWS Elastic Beanstalk and Google App Engine.

3. **Software as a Service**

   This is the most sophisticated offering from cloud service providers whereby the client does not manage any part of the underlying system whatsoever. Instead, a client simply pays for a service which offers some functionality that is valuable to them. Common examples of such products include Office 365 and Gmail.

2.1.1 Virtual Servers

Virtualisation is at the core of IaaS in many cloud environments. It facilitates shared tenancy options on single pieces of hardware without sacrificing security or significantly reducing performance. Without virtualisation, clients would be forced to rent entire servers at high prices and cloud providers would need to allocate an entire machine for every client. Clearly this methodology becomes unfeasible to operate as

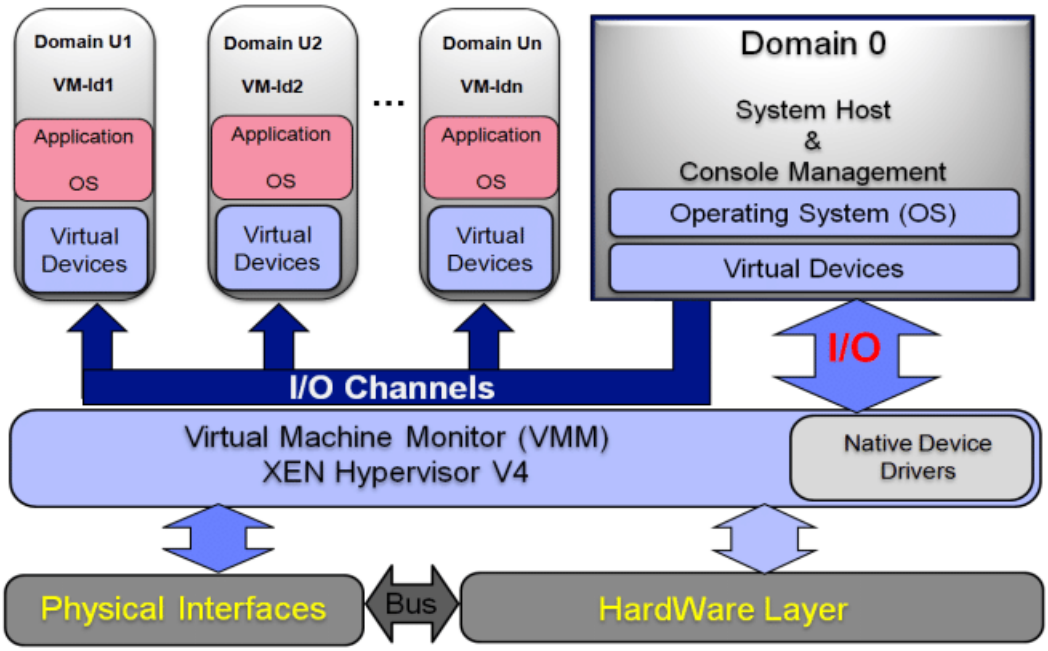

Figure 2.1: Xen hypervisor configuration [1].

client numbers grow. There are a variety of flavours to virtualisation, but the majority are not specific to a cloud-based architecture and thus will not be discussed in this document.

One type of virtualisation that is used in cloud architectures is called paravirtualisation. Here a software application called a hypervisor is installed onto the bare metal machine. This hypervisor allows operating systems to be installed on top of it by providing a set of low-level API calls that the OS can use. Consequently, the operating system makes system calls into the hypervisor which then manages low-level system resources such as the processor, memory, network interfaces and so on. As the hypervisor is managing the underlying physical resources, it is possible to install numerous operating systems onto the hypervisor which manages the overall resource allocation for all hosted operating systems. As long as the underlying hypervisor maintains segregation between each OS, every OS can cohabit the bare metal machine simultaneously without adverse effects. These hosted operating systems are denoted as virtual machines or virtual servers. As long as the underlying hypervisor is identical across all of the servers in the cloud infrastructure, virtual servers can be moved between physical machines trivially by utilising a network exclusively for storage transfer called a storage area network (SAN). This allows for clients to easily change parameters for their virtual server such as CPU type, networking bandwidth and RAM size all simply by either altering the resource allocation within the hypervisor or transferring the entire virtual server to a new physical machine.

Paravirtualisation does have some downsides. Chief among these is that the hosted operating system has to be rewritten to make use of the API provided by the hypervisor. This of course requires that the operating system's source code be open, which is often not the case.

One of the most popular paravirtualisation systems is called Xen [20]. It was originally developed by the University of Cambridge and is now in use on some of the largest cloud infrastructures in the world including AWS. Figure 2.1 explains the operation of hypervisors such as Xen diagrammatically. First, Xen is installed on top of the bare-metal machine and the hypervisor contains important pieces of software specific to the underlying machine such as device drivers. Then, Domain 0 is spawned which is the overall management system for all hosted operating systems. All hosted operating systems are identified by a naming scheme of "Domain n" where n represents an integer greater than 0. Each domain is then assigned virtual devices such as network interfaces, backing stores and so on. Subsequently, when a domain makes a request to a particular virtual device, it does this through the management controller, Domain 0. The controller then processes this request along with others it may receive and then multiplexes this to the real physical interfaces.

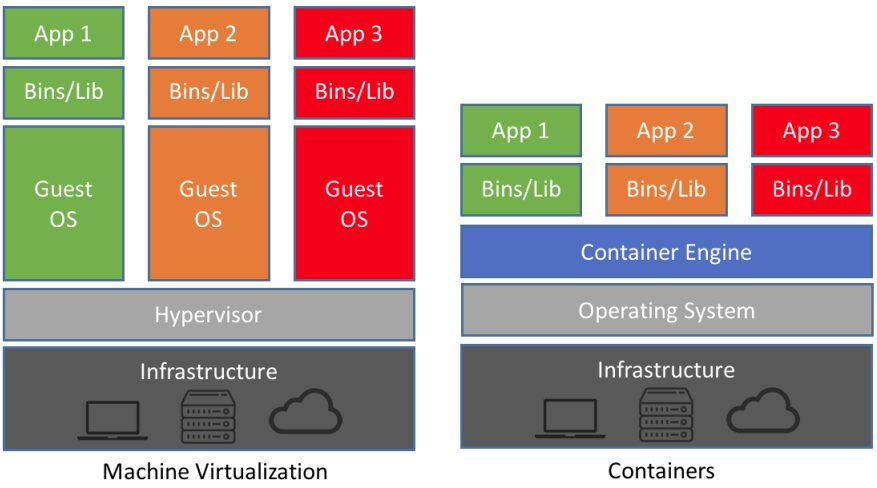

Figure 2.2: Container setup vs virtual machines [2].

2.1.2 Containers and Autoscaling

A critical aspect of a production level service is its ability to autoscale to meet customer demand. With an effective autoscaling system in place, the service should never suffer degradation at peak usage times and also allows a service to dynamically increase in size without having to initially overprovision resources, thus saving money from the service provider's perspective.

A popular method of autoscaling is through the use of containers. Containers are another form of virtualisation, but unlike a virtual machine where the operating system and application are bundled together, a container only houses the application software and all containers run on the same underlying operating system. This has advantages and disadvantages. Firstly, due to the lack of an OS, container sizes are much smaller than their virtual machine counterparts which means that they can be moved around a SAN fabric much faster than virtual machines. In addition to this, due to the fact that all containers run on an underlying operating system which is already booted, it means that the startup time is greatly reduced than that of a virtual machine which has to boot its own operating system. However, due to the fact that every container runs on the same underlying operating system, it means that all containers share the same kernel. This of course may raise some security concerns [21], which include kernel exploits and container breakouts. For example, if a single container were to cause a kernel panic, then all other containers on the physical host cease to function. This can present a particular issue in a shared tenancy environment.

Figure 2.2 contrasts the differences between traditional paravirtualisation and containerisation. Containers run on top of a container engine in a similar fashion to the way that virtual machines interact with a hypervisor. The container engine is a program that runs on the underlying OS installed on the physical machine and manages all containers that are running. An example of a popular container engine is the Docker Engine [22] and is widely supported across many cloud environments. Using this technology one can create microservices in the form of Docker containers that interact with each other over a network. Furthermore, when coupling this technology with a cloud-based container service such as AWS Elastic Container Service (ECS) [23], it is possible to scale these microservices independently of one another automatically to create a fully redundant, fault-tolerant and scalable production level service.

2.1.3 Queues

An often overlooked element of cloud architectures is that of queuing services. Queuing is essential in order to decouple elements of a large system to provide asynchronicity. For example, imagine a website that operates a simple image processing service which turns colour images to greyscale. A user would like the website to be responsive such that when one uploads an image for processing, the site remains responsive whilst the image processing is occurring in the background which may take a considerable amount of time. Without a queuing system, this would not occur and the user would be forced to wait until the processing is complete before they can have another web-page served. A queue can be employed such that when a user uploads an image, it is placed on the queue ready for further processing. The

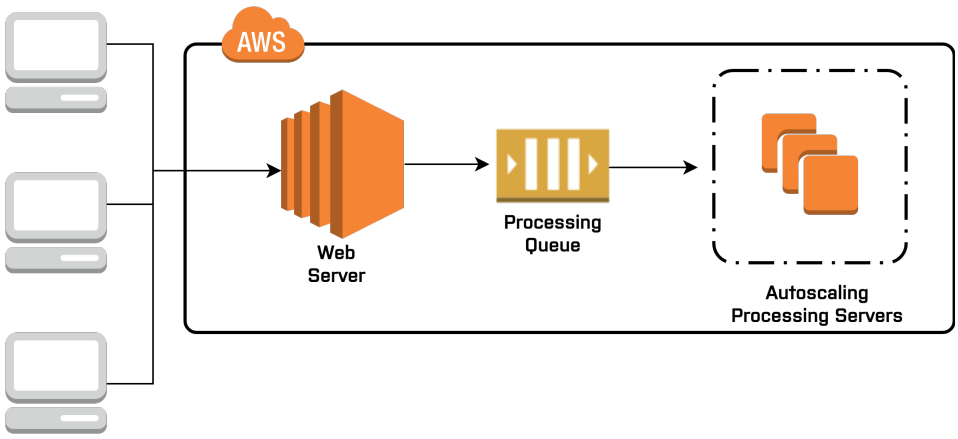

Figure 2.3: AWS SQS basic example.

action of appending a job to the queue is far quicker than waiting for the actual processing job to finish, thus the web server remains responsive.

Figure 2.3 illustrates the simple example above using AWS Simple Queue Service (SQS) [24]. Here web clients interact with a web server hosted within the AWS cloud. The web server then appends any processing requests to the SQS queue. This queue can then be directly fed into worker nodes which can be part of an auto-scaling group which process the requests in the queue. Ensuring that the processing servers are part of an auto-scaling group allows for the processing service as a whole to scale to meet demand very easily. The auto-scaling service can simply decide how many processing servers to spawn based on the number of items in the queue. De-coupling the processing component away from the web server component allows for the two components to scale independently of one another in a true microservice fashion, unlike a traditional monolithic system.

2.1.4 Databases

Databases are integral to many systems regardless of whether they are deployed in the cloud. However, most cloud providers in use will offer two types of database system; SQL and NoSQL. Both have advantages and disadvantages so it is important to compare both and choose the most appropriate option for a particular system.

SQL Databases

Structured Query Language (SQL) databases are arguably the most popular databases today. They make up the core of traditional relational database management systems (RDBMS). A multitude of implementations exist such as MySQL, SQLite, PostgreSQL and so on.

A relational database management system is any database where data is stored in tables and follows a relational model. A table represents structured data by utilising columns and rows which are set up prior to data being inputted into the table. Each row represents an entry in the table whilst a column represents the individual elements of each row and their associated types. For example, a table called "person" may exist and columns such as "name" and "age" may be defined with types `string` and `int` respectively. Each row within the table contains values representing each of the columns defined. The specification of columns within a table is denoted as the table schema and must be set up prior to data entry. For example, when using the aforementioned person table, one cannot insert a row with additional fields other than name and age without altering the schema beforehand.

Once multiple tables are defined, they can be linked together by using foreign keys. This declares a relation between two columns in different tables and allows for easy referencing of items programmatically. For example, imagine a situation where an SQL database has tables for users and subscriptions. Each subscription will have a unique ID. This unique ID could be a column in the user table and a foreign key could be set up between these two tables to establish a relation between a subscription ID in the user table and the corresponding subscription row in the subscription table.

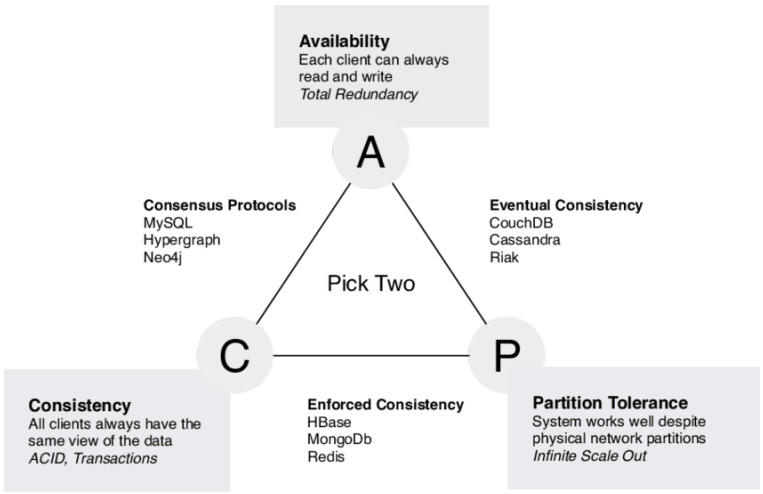

Figure 2.4: CAP theorem visualised [3].

NoSQL Databases

In contrast to SQL databases, NoSQL databases are much simpler. The most simple of which is a key-value store. These types of databases are suited best for unstructured data that is non-relational. There are numerous examples of these types of databases deployed today such as MongoDB, BigTable, HBase and AWS DynamoDB [25]. In the case of DynamoDB, data is stored as items which are JSON strings. This allows for arbitrary unstructured JSON data to be stored in the database as long as certain constraints are met such as a unique primary key.

CAP Theorem and Scalability

Now that both applicable database paradigms have been introduced, it is important to understand how each one fairs with respect to scalability in a cloud-based environment. This is centralised around the CAP Theorem [26], which is neatly illustrated in Figure 2.4. The theorem states that all database systems can only choose a maximum of two properties from availability, consistency and partition tolerance [27]. Availability is the concept that every request to the database is served at all times. This of course requires that the database system be operational 100% of the time. Consistency states that all database instances have the same view of data at all times. As such, if a request to read an element is served by one database, then the most recent write to that element across the database cluster should be returned. A partition tolerant system can operate throughout any amount of network failure that doesn't result in the collapse of the entire network. As such, a database system that is partition tolerant should be able to replicate database records across the cluster during intermittent network outages to maintain normal operation.

One of the leading issues with traditional relational databases when considering scalability is that they operate according to the ACID properties. ACID represents atomicity, consistency, isolation and durability. These properties are intended to guarantee validity in the case of errors, power failures and so on by using rollback techniques to return the database to a prior valid state if an operation is unsuccessful. Such an event satisfying these properties is called a transaction. However, one may observe that these ACID properties only satisfy availability and consistency in the CAP theorem. This means that such relational databases are not partition tolerant. This of course presents a large issue for scalability. If an RDBMS system can only scale vertically (more resources dedicated to a single machine) as opposed to horizontally (resources are spread out over multiples machines) then it limits the overall scalability and redundancy of such a system in a cloud environment. However, certain services like Amazon Aurora [28] offer scalable RDBMS by creating read-replicas. Unfortunately, write operations are still constrained by a single master node meaning that it is still not truly horizontally scalable.

As described earlier, NoSQL databases are much simpler by design and operate on a system known as BASE which stands for basically available, soft state, eventual consistency. This, in essence, relaxes the availability and consistency properties of a database in order for it to be fully partition tolerant. This relaxation allows such databases to be horizontally scalable in cloud environments. Of course with this partition tolerant property fixed, many implementations of NoSQL databases work on providing either

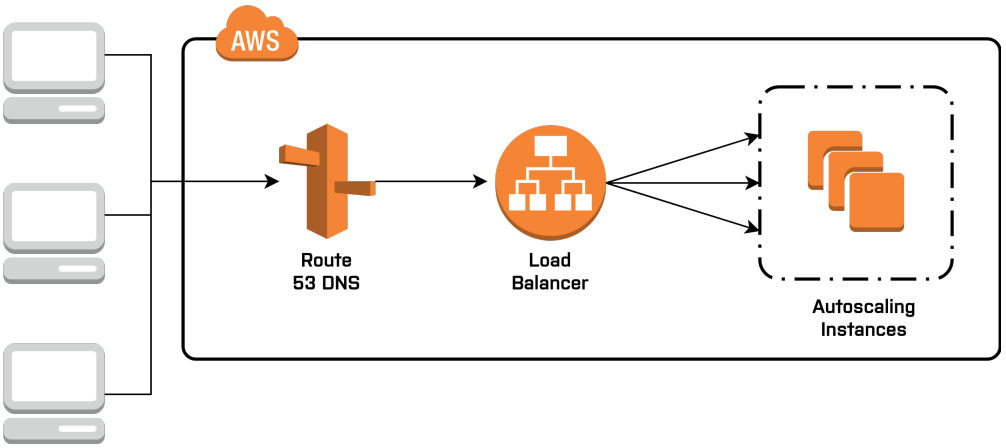

Figure 2.5: AWS load balancer example.

availability or consistency as a priority as indicated by Figure 2.4. Systems like AWS DynamoDB try to offer the best of both worlds in the form of different API calls. For example, "Eventual Consistent Reads" will always return data, but may not reflect the results of a recent write operation. However, "Strongly Consistent Reads" are not always available in the case of network failures, but will return data that reflects the results of the most recent and successful write operation [29].

2.1.5 Storage and Identity Federation

An efficient storage system is integral for services that need to host content on behalf of their users or deliver content to their users. Services such as Dropbox, BBC iPlayer, Netflix and so on all make use of storage systems to host content.

One of the most important aspects of a storage system is redundancy. Service providers will not tolerate data loss from cloud providers. In order to achieve this, cloud providers add certain levels of redundancy by copying the data throughout multiple physical locations. For example, AWS Simple Storage Service (S3) [30] provides API calls which synchronously store data across multiple locations before returning a success flag. Further to this, integrity checks are performed on data at regular intervals and corrupted data is repaired using coding schemes and redundant data [31].

Another important consideration for storage is both security and segregation. For example, a service provider may wish to encrypt a user's stored data and segregate it from other users such that it is only accessible by a single user (the owner of the data). Encryption is offered by encrypting the data stored at rest, usually with either the storage provider's key or a key of one's choosing. Segregation is done by employing an identity federation. An identity federation is specific to each cloud provider and allows external identity providers to integrate with cloud-based resources. For example, a cloud identity federation can integrate with an external authentication system in order to uniquely identify users in the form of federated identities. When users then wish to access a private resource, their federated identity is used to create a resource token which allows a user to access their private cloud-based resources.

2.1.6 Load Balancing

A critical aspect of any cloud-based scalable service is load balancing. Without it, requests will not be shared evenly between all available resources. Figure 2.5 represents a typical example of an API service in an AWS cloud environment. First, clients will use the DNS (Domain Name System) to resolve the hostname of a particular load balancer into an IP address. In the diagram above, AWS Route 53 [32] represents the nameserver for the load balancer. Communication then proceeds with the load balancer using the Transmission Control Protocol (TCP) which is the underlying transport protocol of HTTP and guarantees reliable, in-order delivery of information between hosts. In this initial stage of communication, various other ancillary tasks can be performed. Most notable among which is Transport Layer Security (TLS). The load balancer can act as a HTTPS endpoint for all communicating clients and communication with the downstream server instances can then be done over HTTP, making for a more lightweight application. The load balancer routes individual requests to autoscaling instances

(application servers) using a queuing algorithm such as round robin. This ensures that all requests to the load balancer are spread evenly across the server cluster rather than being concentrated on one particular instance.

2.1.7 Serverless

Serverless technologies are one of the newest additions to cloud services. Examples include AWS Lambda, Azure Functions and Google Cloud Functions. Instead of managing servers and scalability manually (IaaS), the cloud provider manages the deployment and scalability of application software completely. This provides developers with the opportunity to develop applications with the following characteristics known as the "four pillars of serverless computing" [33].

1. **No Server Management**

   The cloud provider manages the provision of application software to physical machines automatically through the use of containers. A developer writes application software and then upon application invocation, a container is created and transferred to a physical machine running a container engine which then subsequently executes the application software. This process happens automatically and more often than not, if the rate of invocation is frequent, the container is stored in RAM on the physical machine and so avoids this "cold start" process.

2. **Flexible Scaling**

   Another key feature of serverless technologies is the ability for applications to be scaled automatically to meet demand. Invocations for particular applications can be event-driven or scheduled. When the number of invocations reaches a certain threshold, more instances of the container hosting the application can be deployed to multiple physical machines. This is similar to autoscaling, but completely abstracted away from the developer.

3. **High Availability**

   Ensuring application and service availability is essential for any production level system. Serverless technologies manage this automatically by deploying application containers to multiple physical machines to create a highly available application with zero effort required from developers.

4. **Zero Cost at Idle**

   Arguably one of the most attractive qualities of serverless is the fact that one is only charged upon invocation of an application. As the containers are dynamically created, transported and executed on an event-driven basis, there is no need to purchase or rent portions of physical machines upfront as with IaaS. Instead, one is only charged based on factors such as execution time, number of invocations, RAM utilisation and so on. This can make serverless an attractive platform for certain applications, particularly those without latency requirements and with infrequent invocations.

Despite the attractive qualities of serverless, there are some disadvantages to take note of. In particular, high latency and cost. If the rate of invocation is infrequent, the time taken to perform a cold start and serve the request is much more than that of an already running application on a physical machine. Therefore, serverless is not well suited to latency critical applications. Another point to consider is monetary cost. Vendors charge based on the number of invocations of an application rather than the price of reserving a physical machine like IaaS. Therefore, if an application is heavily used, it can actually work out more expensive than an IaaS machine reservation.

2.1.8 Continuous Integration and Deployment

As introduced in Section 1.3.2, continuous integration is the process whereby developers commit code changes to a source code repository frequently and these changes are then analysed against automated tests. This phase acts as the first pass of review and allows for any breaking changes to be identified easily, thus preventing issues propagating into production code. Once this stage is finished, the changes are merged and the updated application is built into an image and deployed to either a staging or production server. This second phase is known as continuous deployment.

Figure 2.6 demonstrates a typical CI/CD setup within the AWS cloud. First, software developers commit to a particular branch on GitHub, this then activates a web-hook within AWS CodePipeline [34]. CodePipeline then archives the repository at that commit and passes the archive to AWS CodeBuild

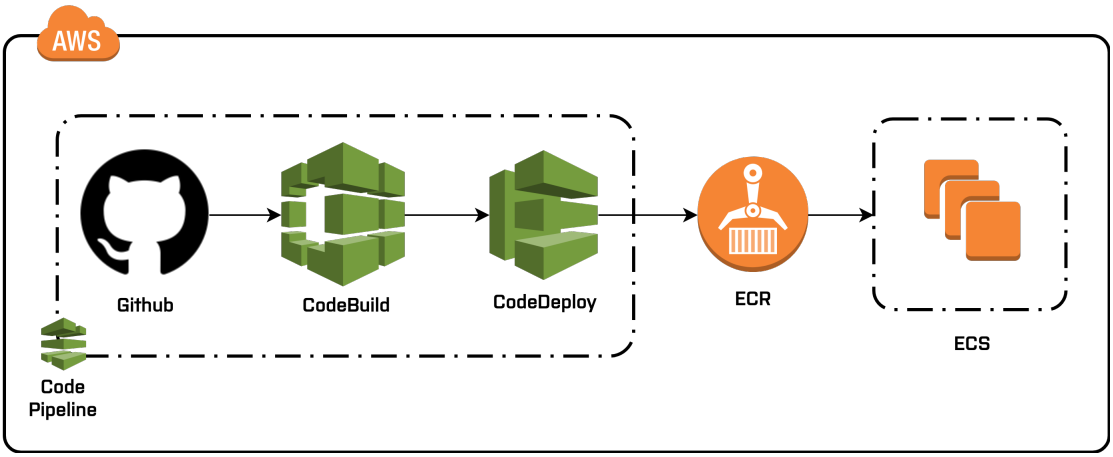

Figure 2.6: Typical example of CI/CD within AWS.

which creates a Docker image based on some ruleset held within the Git repository. This Docker image is then passed to AWS CodeDeploy which places it into the appropriate repository within AWS Elastic Container Registry (ECR) [35] and alerts AWS Elastic Container Service (ECS) that a new image is ready for deployment. ECR is effectively a private repository for Docker images and can assist ECS with deployment by referencing images via version numbers and tags. For example, ECS may request images from ECR with a tag such as "latest" to ensure that the latest Docker image is pulled and deployed to the production machines within the cluster.

2.2 Side-channel Analysis

Side-channel analysis is key to this project and therefore it is important to present information pertaining to the attack processes that will be used in the system architecture.

2.2.1 AES

The AES algorithm is a common target for side-channel analysis [36]. The next section demonstrates an attack process using AES as an example. Subsequently, a brief introduction into the format of this cryptographic algorithm is given.

AES is a substitution-permutation network and operates on blocks of data of 128 bits in length, usually denoted as the plaintext. In addition to this input, a key is also specified which can be one of three sizes; 128, 192 or 256 bits in length. The algorithm operates by performing operations on the data in a series of rounds. The number of rounds performed depends on the key length, but each round contains all of the operations below in the order listed. The only exception to this is the last round which does not perform the Mix Columns operation. In addition to this, the Add Round Key operation is performed prior to the first round commencing.

Sub Bytes

The sub bytes operation uses the AES S-box [6] which is a pre-calculated substitution table. It is a non-linear substitution function and operates on each byte independently of the current AES state and is invertible. The S-box is applied to each byte in the AES state matrix which is an internal 4×4 matrix where each element contains a byte.

Shift Rows

In this transformation, the elements of the rows within the internal matrix are shifted leftwards in a cyclic fashion. Elements of the first row remain unchanged, the second row shifts by 1, the third by 2 and so on.

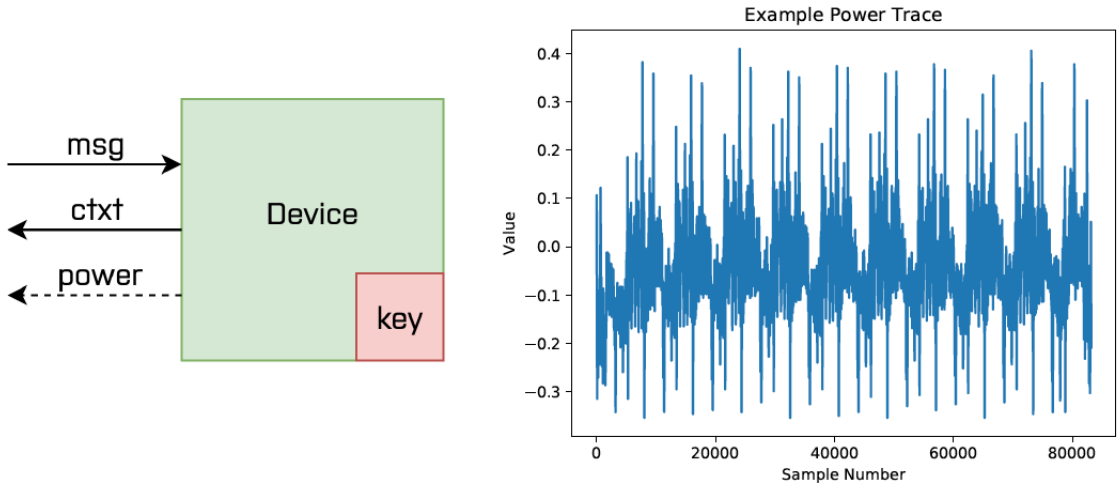

Figure 2.7: An example target device and power trace.

Mix Columns

This operation operates on the state in a column-wise fashion. Entire columns of the 4×4 matrix are treated as polynomials within $GF(2^8)$ and are multiplied by a fixed polynomial with modular reduction to produce an entirely new resultant column.

Add Round Key

In this operation, an AES round key from the AES key schedule [6] is added to each column of the state via bitwise XOR.

2.2.2 DPA/CPA and Attack Process

The system will have full support for power trace based acquisition and analysis tasks. Differential Power Analysis (DPA) [37] and Correlation Power Analysis (CPA) [38] are methods of extracting sensitive information out of power traces such as those illustrated in Figure 2.7.

In order to understand the inner workings behind these attacks, it is first essential that one is familiar with the attack model. Figure 2.7 depicts a high-level scenario of a target device. The device offers the adversary the ability to send it arbitrary plaintext messages and an encryption of the plaintext is computed and sent back. In addition to the ciphertext, the device also leaks its power consumption to the adversary. The goal of the game is to use this power consumption data to determine the secret key held within the device.

The fundamental principals behind both CPA and DPA attacks are to model the physical device under attack. First a series of messages are passed to the device under attack which holds a secret key. From this, real power consumption data is collected. With the model of the device, one feeds in the same data along with a key hypothesis. Hypothetical power consumption data is then extracted. Finally, statistical analysis is employed to compare the real power consumption data against the hypothetical power consumption data in order to reach a verdict about the key hypothesis.

DPA/CPA attacks generally follow a five-step process from a high-level perspective [39]. Below are these steps listed with an example given from an AES attack perspective.

1. **Choose Target Intermediate Variable**

   These values must be a function of the key and data fed into a known cryptographic algorithm. In practice, such variables are derived from functions which are non-linear in order to provide distinguishing features between key hypotheses.

   If one uses AES as an example algorithm, the output byte of the first S-box in round one is a good candidate as it's a function of the first byte of the plaintext and the first byte of the key.

2. **Measure Power Consumption**

   Power consumption is usually obtained through the use of equipment such as oscilloscopes which monitor the target device. For a series of encryptions, the power consumption is measured at regular sampling intervals to produce a power matrix of dimension $t \times s$ where t is denoted as the number of encryptions (each under a different plaintext) and s is the number of samples taken by the oscilloscope per encryption.

3. **Calculate Hypothetical Intermediate Values**

   Clearly when one guesses k bits of a key, then this leads to 2^k hypotheses. If one combines this fact with the t inputs to the target device then it is possible to produce a matrix of size $t \times 256$ for each key byte. These matrices enumerate all possible key hypotheses for all plaintext inputs.

4. **Map Hypothetical Values to Predicted Power Consumption Values**

   It is possible to map the hypothetical intermediate values to predicted power consumption values by utilising an appropriate power model. One such model used is a function called the Hamming Weight. This function returns the number of non-zero characters in a particular string. Therefore, in the case of a binary string, it represents how many occurrences of 1 are present in the string. One can take a key byte matrix of size $t \times 256$ matrix and apply the Hamming Weight function to it in order to produce a matrix representing power consumption for each key byte.

5. **Statistical Analysis of Power Values**

   At this point, one now has both the real power consumption matrix and a modelled power consumption matrix for each key byte. Therefore, a distinguisher is used to compare the two in a column-wise fashion. The general principle of the distinguisher here is to produce correlation values such that the higher the value, the more likely the associated key hypothesis is correct. Subsequently, a simple iteration over the resultant correlation matrix is able to determine which key hypothesis is the most likely outcome for each key byte.

Figure 2.8 illustrates the above process for AES on a per key byte basis. Firstly, the real power consumption matrix is calculated by sampling s samples over t different plaintext encryptions. Then for each key byte, a hypothetical power consumption matrix is generated based on the following formula:

$$HW(\text{S-box}(m_{t,b} \oplus \widetilde{k_j}))$$

where HW denotes the hamming weight function, S-box represents the standard AES S-box, $m_{t,b}$ represents the b^{th} byte of the t^{th} plaintext and $\widetilde{k_j}$ represents a particular key byte hypothesis.

This hypothetical power consumption matrix is then fed into a distinguisher along with the real power consumption matrix. Here the matrices are compared in a column-wise fashion in the following way:

$$C_{0,0} = D(col_0(H), col_0(R)), C_{0,1} = D(col_0(H), col_1(R)), C_{0,s-1} = D(col_0(H), col_{s-1}(R))$$

$$C_{1,0} = D(col_1(H), col_0(R)), C_{1,1} = D(col_1(H), col_1(R)), C_{1,s-1} = D(col_1(H), col_{s-1}(R))$$

where col_x denotes the x^{th} column of a matrix.

Calculation of correlation coefficients is often done via the use of the Pearson correlation coefficient equation [40][41]. Fundamentally, correlation assesses the relationship between two variables, in this case, a hypothetical power consumption matrix for a particular key byte and a real power consumption matrix. The equation below always produces a value, r such that $-1 < r < 1$, where -1 and $+1$ indicate perfect negative and positive linear correlation respectively.

$$r_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2(y_i - \bar{y})^2}}$$

where $\bar{x}$ and $\bar{y}$ denote the respective sample means.

This operation is performed such that a correlation matrix is produced for each key byte. Then one simply iterates over the correlation matrix to find the maximum value. Once this is calculated, the corresponding key byte value can be read off by reading the column index of the highest value. This is then done for each key byte correlation matrix to determine the entire AES key.

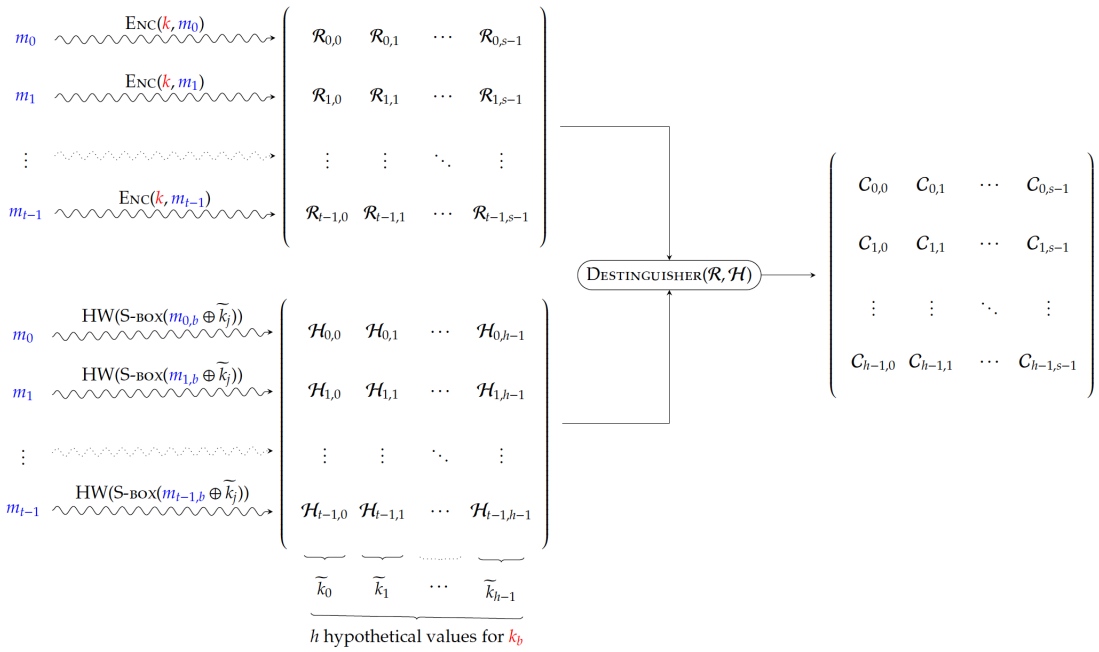

Figure 2.8: DPA attack visualised.

Ancillary Tasks

Side-channel analysis often requires the integration of pre-processing and post-processing tasks in order for attacks to be successful. One particular post-processing technique that is often required is key enumeration. The goal of this is to find a method to efficiently enumerate the most likely keys returned from a DPA attack. Key enumeration is used extensively because not all DPA attacks will return the correct key. For example, a particular key-byte may have been identified incorrectly and consequently, the key recovered is incorrect. Consider the case where a correct key-byte is actually the second highest value in a particular correlation matrix. In order to have the entire key enumerated and tested, one needs to assess key bytes other than the maximally correlated one from each matrix. One of the most important questions to this approach is how deep is the unknown key in the remaining key space, and second, how expensive is it to enumerate keys up to a certain depth? [42]. Some well-known methods such as "path count"[43] and a histogram based approach [44] are comparable in performance and aim to enumerate keys in an efficient manner to help solve this problem.

2.3 Third Party Technologies

Besides cloud vendors, third-party technologies play a large part in this project. This section explores each technology used in order to provide an overview of their functionality.

2.3.1 Auth0

Auth0 [45] is an identity provider that manages all authentication mechanisms that surround modern applications. It allows developers to save a considerable amount of time by providing a useful framework that represents a complete authentication system for a custom application. Instead of a developer having to create a protected password database, secure API endpoints and so on, Auth0 provides this out of the box so developers can instead focus on developing the application instead of an authentication system.

The core identity system is based around JSON Web Tokens (JWTs) [46]. Auth0 provides three types; ID tokens, access tokens and refresh tokens. ID tokens contain information specific to that user and are somewhat akin to a passport in the real world. Access tokens are used to grant access to certain resources like API endpoints and are ephemeral in nature such that they expire after a certain amount of time. A refresh token is allocated to a user when they successfully authenticate and is used to generate new access tokens after they expire. In order to guarantee the authenticity of these tokens, they are signed by a private key specific to the application and known only by Auth0 servers. Resource endpoints can then check the validity of these tokens by verifying the signature against the application's public key.

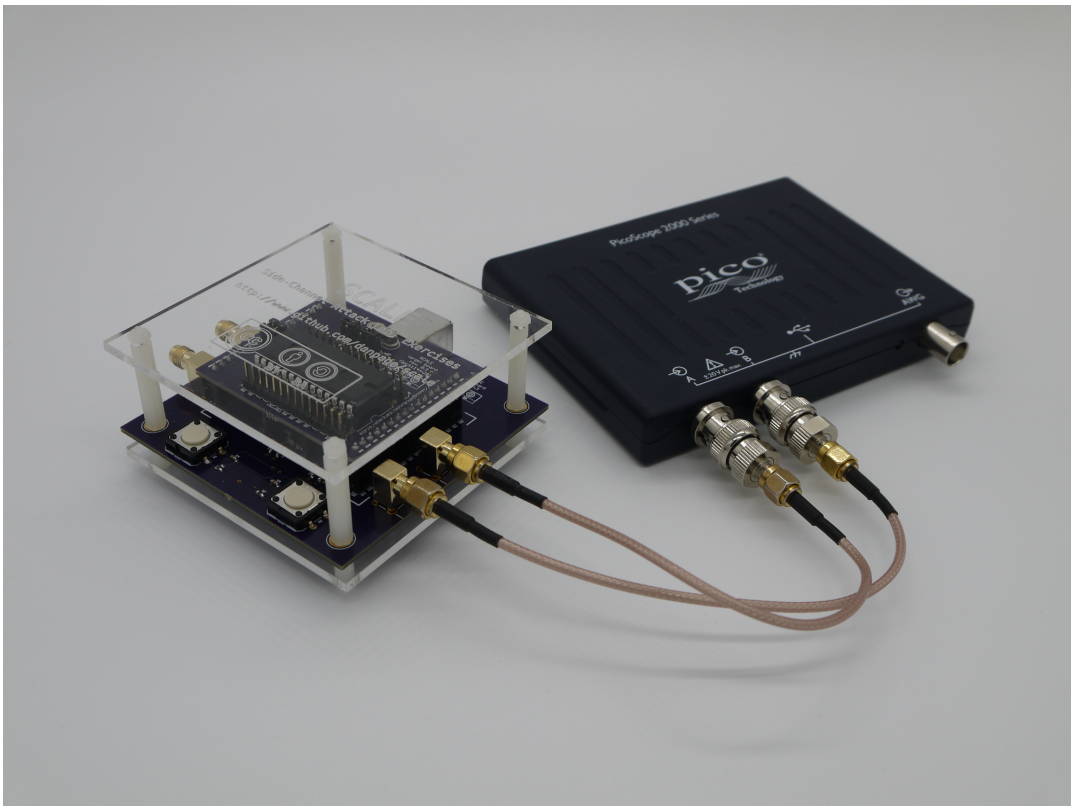

Figure 2.9: SCALE board with PicoScope setup [4].

2.3.2 Jlsca

Jlsca is a third party side-channel analysis library developed by Riscure Labs. It is written in Julia and provides a "plug and play" framework for side-channel analysis. It has support for many advanced features, but with respect to power analysis, it contains DPA and CPA engines with examples on how to perform attacks with sample trace sets. A library such as this is extremely useful in providing a rugged set of attacks which can be performed on pre-collected trace sets to provide an indication of how secure an implementation of a cryptographic algorithm really is. Using Jlsca removes the need for implementing many DPA/CPA attacks manually which saves a considerable amount of time.

A chief advantage of using the Jlsca library is that one is able to use the in-built multiprocessing support of the Julia language to speed up the attack process on multi-core machines. However, a big downside to the library is that the trace sets have to be in Riscure's TRS format. Consequently, pre-processor conversion functions have to be created in order to ensure that the trace sets are usable by the library.

2.3.3 PicoScope and SCALE

PicoScope is a brand name by Pico Technology [47]. PicoScopes are oscilloscopes and can be used to capture power consumption data and transmit this information directly to a connected workstation via USB. This facilitates the possibility to automate the acquisition of power consumption data during the analysis of a connected device performing some set of cryptographic operations.

Figure 2.9 depicts a setup of a PicoScope combined with a Side-Channel Attack Lab. Exercises (SCALE) [4] development board. The SCALE board comes in two parts, a host board and a target board. The host board acts as a motherboard into which one is able to insert a particular target board. One is able to swap between different target boards to reflect an interest in a particular target device such as a specific type of microprocessor. The SCALE boards also communicate with workstations over a USB connection which provides power and also allows for the boards to be programmed with an implementation of a cryptographic algorithm. One will notice that coaxial cables are connected to SMA connectors on the SCALE host board. This allows for the power consumption data to be extracted by the PicoScope.

Chapter 3

Project Execution

3.1 Website and API

3.1.1 Introduction and Overall Design

This project is part of the Side-Channel Hardened RISC-V (SCARV) [48] group. SCARV Lab represents the "laboratory free" side-channel analysis portion of the overall group and has been granted funding from AWS in the form of research credits. AWS was chosen for this reason along with the fact that AWS provides all of the functionality to make this project feasible in the limited time available.

Figure 3.1 depicts the overarching setup of the entire project within the AWS Cloud. The following subsections of this chapter define the functionality and design of each of the components in full along with the methods of interaction between relevant elements. The overall goal of this closely linked infrastructure is to provide an automated, highly available and scalable approach to "laboratory free" side-channel analysis.

At a high level the website and API allow for a user to perform the following actions:

1. Sign up/login to their SCARV Lab account.
2. View the status of submitted acquisition or analysis jobs.
3. Submit new acquisition or analysis jobs for processing.
4. Download the results of completed acquisition or analysis tasks for further processing or inspection.

3.1.2 Web Frameworks

In order to provide an automated system offering side-channel analysis as a service, it is imperative that an easy to use web interface and API system is designed to facilitate this. The API server and web server are HTTP servers bundled together and written entirely in Python Flask [49]. Flask is a micro-framework for Python based on the Werkzeug web server gateway interface (WSGI) web application library and the Jinja 2 templating engine. Flask serves as the core for the entire project as it handles all incoming requests and dispatches them to the appropriate backend services. Werkzeug is not suitable for a production level application as it performs poorly under high loads. Therefore, the popular and lightweight waitress WSGI server [50] was used instead.

The website utilises Bootstrap [51] which is a free and open source CSS framework designed to aid with web development by utilising templates for forms, buttons, navigation and so on. The front end website acts as a wrapper interface for the underlying API by using custom JavaScript.

3.1.3 Route 53

In order to provide a web-based service, it is essential that DNS records are created so that users are able to access the service from a fully qualified domain name (FQDN). As the project is a subsidiary of the SCARV organisation, it was allocated the FQDN of `lab.scarv.org`. Due to the fact that the project is hosted within the AWS cloud, the DNS records associated with this subdomain were chosen to be managed using AWS Route 53. One of the reasons for this is that Route 53 supports `ALIAS` records,

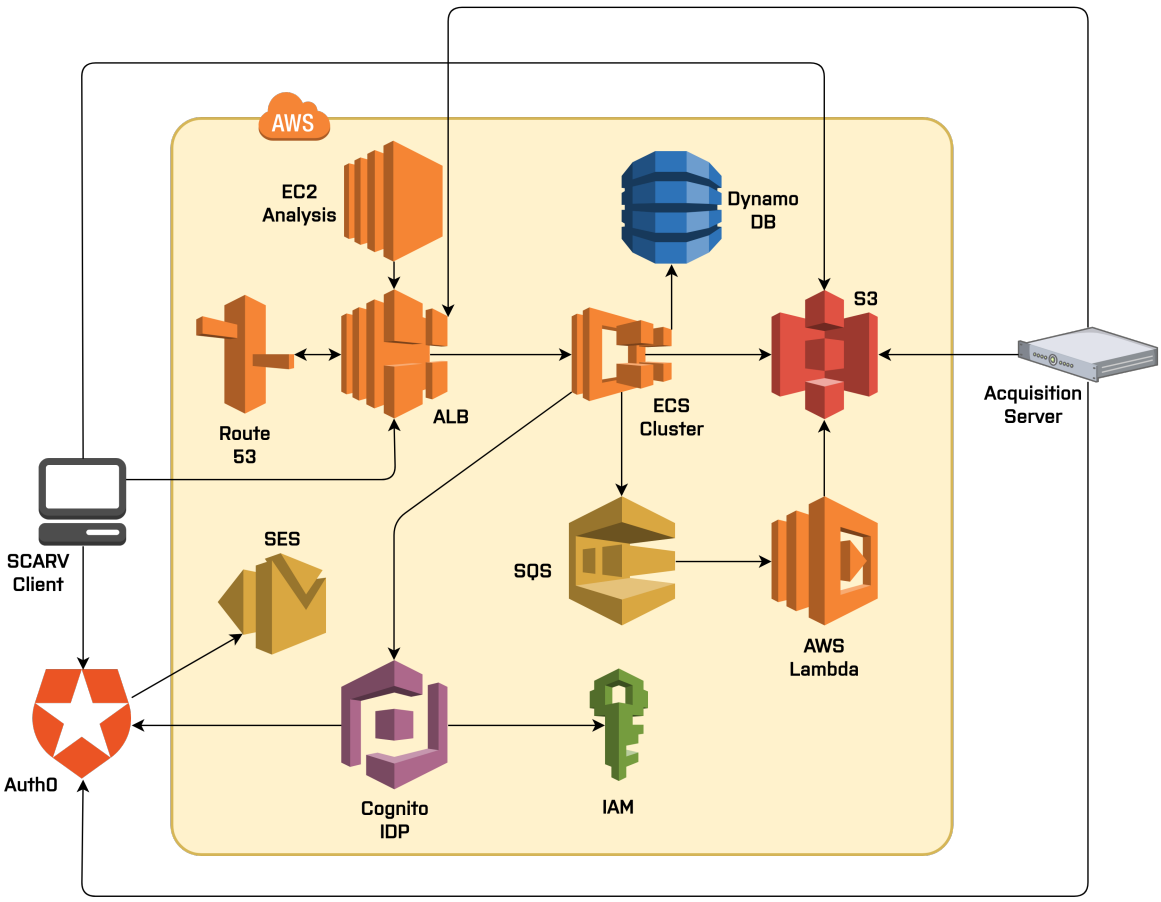

Figure 3.1: Overall AWS infrastructure.

which are very useful for load balancers held within AWS. Such load balancers change IPv4 and IPv6 addresses regularly for scaling purposes. Therefore, a static record such as `A` or `AAAA` is unsuitable and can even result in downtime of services if IP changes do occur. Instead, load balancers can be referenced by a hostname. However, it is a violation of DNS rules to place a `CNAME` record at the apex of a zone. For example, `lab.scarv.org CNAME lb_hostname` transfers control of the entire allocated subdomain to the load balancer's nameservers meaning that queries to subdomains of `lab.scarv.org.` may be returned as `unbound`. Therefore, to overcome this issue, `ALIAS` records re-resolve the hostname held within the record at query time to ensure that querying clients are returned the correct IP address for the designated load balancer.

3.1.4 Application Load Balancers

Load balancers are an incredibly important aspect of any scalable service. Without them, load cannot be shared evenly between all available servers. SCARV Lab employs AWS Application Load Balancers (ALB). These load balancers operate at Layer 7 of the OSI model [52]. As such, they are able to perform high-level functions such as HTTP redirection, URL inspection, TLS operations and health checks to downstream servers based on HTTP response codes.

HTTPS

Application load balancers support HTTPS via the use of AWS Certificate Manager (ACM). ACM handles the distribution of public certificates which are signed by the widely trusted Amazon Certificate Authority. One is able to obtain a certificate by proving ownership of a domain. ACM then installs an issued certificate onto the load balancer, which is then served to connecting clients. One of the core benefits of using ACM is the fact that certificates are free as long as they are utilised within AWS.

The application load balancers in use within SCARV Lab accept HTTPS connections from connecting

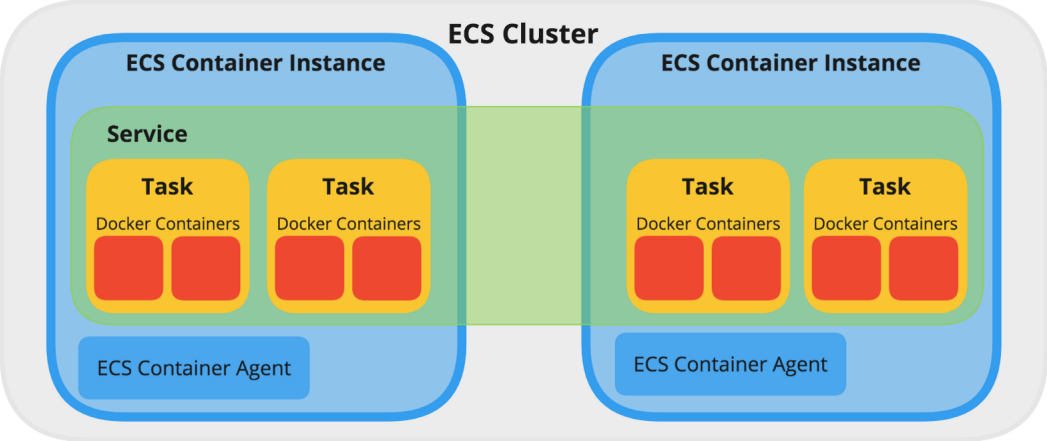

Figure 3.2: SCARV Lab ECS layout [5].

clients. A certificate was created via DNS verification for the hostname `lab.scarv.org`. Once a secure TLS connection is established with a client, the load balancers then communicate with the application servers via plain HTTP. This methodology greatly reduces the load and complexity of the underlying application software by offloading cryptographic operations to the load balancers. For security purposes, connecting clients are never served over HTTP. Instead, the load balancers are set up to return a HTTP 301 redirect to the HTTPS endpoint for both the website and API, thus preventing any sensitive information travelling over an insecure channel.

Downstream Routing

The ALB sends requests evenly to available downstream servers which are members of a target group. The routing decision is based on metrics which include the health of specific application servers and the individual load of application servers.

Dual Stack Addressing

One of the most influential and recent developments to the Internet as a whole is the introduction of IPv6. IPv6 is the latest version of the Internet Protocol and attempts to mitigate the address shortage issue faced with IPv4. SCARV Lab ensures connectivity to both IPv4 and IPv6 based clients by providing dual stack addresses to the application load balancers.

3.1.5 ECS Cluster

The ECS cluster, as depicted in Figure 3.1, represents the heart of the project. All application software running the website and API service is hosted within ECS.

SCARV Lab utilises ECS in Fargate [53] mode. Fargate is a compute engine for ECS that manages the provision, configuration and scaling of clusters of servers automatically. Consequently, one does not need to specify parameters such as EC2 server types. Instead, Fargate allows one to simply input the CPU and network requirements for each application. Fargate will then provision the containerised application to the appropriate hardware. This approach allows developers to focus on designing and building applications instead of managing the infrastructure that runs them [53].

Figure 3.2 illustrates the way in which ECS is configured. Firstly, task definitions are blueprints representing an application. They define the CPU, memory and network requirements for a set of containers along with any environment variables that need to be present. A task definition is related to a task in the same way that a class is related to an object in object-oriented programming. A task definition is instantiated to create a running task. A service directly maps to a task definition and controls important aspects that influence scalability such as the minimum and the maximum number of running instances of the task and the underlying scaling policy. A Fargate cluster can contain multiple services that scale independently of one another. Further to this, services can be spread over multiple physical machines for high availability as depicted in Figure 3.2.

SCARV Lab's website and API service scaling policy mandates that a minimum of two tasks must be running at any given time and that a scaling event occurs whenever the overall cluster CPU load is raised above 60% for 1 minute. This policy provides high availability and dynamic scalability.

3.1.6 DynamoDB

Choice of Database

Within the previous chapter, the merits of NoSQL databases were described in relation to scalability and their ability to handle unstructured data. The job manifests which define various parameters for both side-channel analysis and acquisition are written in JSON and are detailed in later sections. JSON data is not an ideal fit for the columnar database format of traditional relational database systems. One has to iterate over the data systematically and extract keys and values and place them in the corresponding columns. This of course represents an issue in terms of computational effort and the ability to expand the schema in the future. For example, adding a new item in the JSON job syntax requires a database schema change, API server change and so on. Even if one does this, then scalability issues still remain.

DynamoDB is a NoSQL based key-value store with JSON item syntax. This means that JSON data can be directly inserted, queried, manipulated and removed directly from DynamoDB providing that the correct data types are used. This presents a large advantage given the fact that SCARV Lab job manifests are based on JSON.

Upon data upload to DynamoDB, the API server sanitises the data and converts the types within a JSON document to DynamoDB accepted types. For example, `int` and `float` must be converted to the Python type of `Decimal` prior to upload. This process is of course reversed when retrieving data. Nevertheless, this process required a fraction of the development effort compared to a solution fit for a relational database system.

Scalability has been integrated in from the foundation of this project and the database system is no exception to this. DynamoDB is set up to scale automatically based upon usage metrics. Like ECS, scalability management can be done solely by AWS in an abstracted fashion. DynamoDB marks load in terms of read request and write request units [54]. If DynamoDB detects a large number of read or writes to the database cluster in the form of a total load greater than 70% for either read or write operations, more capacity is provisioned to counter this and reduce total load dynamically.

Database Queue

One will notice that certain elements of the overall architecture in Figure 3.1 are outside of the AWS Cloud. The most relevant of these in terms of the database system is the acquisition server. The operation of this is not within the control of AWS and side-channel acquisition is by no means an instant operation. Therefore, conventional queueing systems utilising SQS are not applicable for this scenario. For example, items on queues have a fixed maximum lifetime. If the acquisition server system were to fail for a considerable amount of time, the loss of job manifests would likely not be tolerated by end users. It is for this reason that DynamoDB also acts as a queue for side-channel tasks via the use of secondary indexes. Within the JSON manifest format there exists two fields that allow for this functionality, namely `remark` and `submit_time`. The `remark` field indicates the current status of the job that the manifest represents and can be in one of 4 states: `pending`, `processing`, `archiving` or `complete`. `submit_time` indicates the submission time of the job. Using this information, the API server can allow services to request the oldest pending job listed in the database.

Queries such as this can be made very efficient. Normally, one would have to scan the entire database for these parameters which is a costly and slow operation as it returns all items within the database. Instead, one can define a secondary index within DynamoDB based on the two aforementioned fields. `remark` is selected as a partition key and `submit_time` is selected as a sort key to form a composite key. Every item within a database needs a unique primary key and on DynamoDB a primary key is made up of a partition key and an optional sort key [55]. Jobs with the same partition key are stored in the same physical location within the database and are then sorted by the sort key in ascending order. Therefore, efficient queries can be done on the SCARV Lab job manifest tables which retrieve pending jobs in order of submit time, whilst also retaining the ability to query jobs via their ID through the primary index.

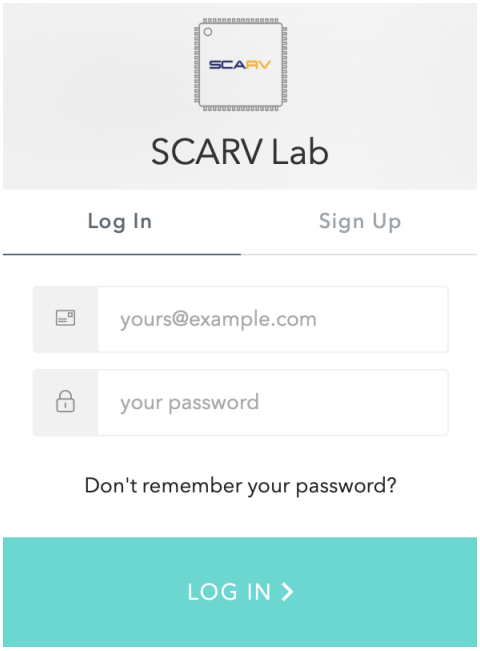

Figure 3.3: SCARV Lab login form.

3.1.7 Auth0 and SES

Auth0 is the SCARV Lab identity and authentication provider. It is responsible for handling sensitive data pertaining to user authentication. Auth0 hosts a username password database for all SCARV Lab users and manages user authentication via API calls and secure hosted pages with embedded forms such as those shown in Figure 3.3.

Auth0 provides a Python Flask SDK meaning that integration into the project was fairly seamless as long as one is familiar with the OAuth 2.0 specification and concepts [56]. Users are authenticated via the Authorization Code and Refresh Token grant types for the website and API respectively.

Authorization Code

The authorization code grant is specific to web-based clients and follows a lengthy procedure [57] which is described below:

1. The SCARV Lab website login button is displayed for users who aren't authenticated. If a user clicks this button they are redirected to the Auth0 authorization page for SCARV Lab.
2. The login prompt as per Figure 3.3 appears. They will proceed to authenticate and Auth0 redirects the user back to a specific endpoint on the SCARV Lab web server with an authorization code embedded within the URL.
3. The Auth0 Flask SDK sends this code to the Auth0 token servers along with secret parameters which identify the SCARV Lab web server to Auth0.
4. Auth0 verifies the code along with the secret parameters to ensure that the request is genuine and from the SCARV Lab web server.
5. Auth0 responds with an ID token and an access token. These are then written into the session cookie for easy verification when future requests are made. The web server then responds with this session cookie and the content of the web page relevant to the original request.

Refresh Token

With respect to the API, the authentication flows are much simpler. The most notable one to mention is the refresh token flow. The refresh token authentication process is far more intuitive than the authorization code flow. One simply submits a refresh token along with some other identifying parameters to the

Auth0 token endpoint in the form of a JSON body within a HTTP POST request. This is then validated by Auth0 and if successful, an access token and ID token are returned.

Accessing Resources

Once authentication has succeeded, both an ID token and access token are returned. For web-based clients, this is stored in the form of a session cookie which is signed with a secret key from the web server to prevent modification by end users. Upon subsequent requests, this cookie is inspected and if verification fails or the cookie has expired, the user is prompted to re-authenticate. For API based clients, access tokens are placed within the HTTP Authorization header. These tokens are then inspected and verified when accessing restricted resources.

Scopes

One of the most important aspects of any authentication system is to establish privilege levels for users. For example, administrative endpoints should only be accessible by administrators and not by regular users. To achieve this, OAuth 2.0 implements a concept called scopes. These are strings attached to an access token which define privileges for that user. For example, in SCARV Lab, three resource scopes are defined; `user`, `infrastructure` and `admin`. The presence or absence of these scopes impacts which resources a user is able to access on the SCARV Lab API or website.

New Users

In order for side-channel analysis as a service to be successful, new users must be able to sign up to the system. Auth0 handles this from an identity standpoint, but another important element is email verification. To perform email verification, one needs to have the ability to send automated emails. AWS Simple Email Service (SES) [58] allows for this functionality. Auth0 directly integrates into SES via the use of SMTP credentials and HTML templates. When a new user signs up, Auth0 instructs SES to send out a verification email to the relevant user who can then click a link within the email to verify their email address. A key part of doing this successfully is ensuring that emails are not marked as SPAM by receiving clients. One is able to mitigate this possibility by implementing concepts such as Domain Key Identified Mail (DKIM), Sender Policy Framework (SPF) and Domain Message Authentication Reporting and Conformance (DMARC). SPF declares publicly via DNS records which servers are permitted to send email from the domain. DKIM cryptographically signs messages sent from a mail server and publishes public keys via DNS records for the relevant domain which can be used by receiving clients to check the authenticity of the message. Finally, DMARC is a policy system informing receiving clients what to do if SPF or DKIM verification checks fail. SCARV Lab uses all three of these technologies to help mitigate the possibility of phishing attacks, thus reducing the possibility of the domain gaining bad reputation from email providers.

3.1.8 S3, Cognito IDP and IAM

In order for users to access results from side-channel acquisition and analysis, it is imperative that a storage system is used to capture these results. Without such a system in place, results from side-channel tasks would be unavailable. A very simple solution would be to create one global shared storage system whereby users are able to access their trace sets through a shared space. However, this raises a privacy concern in the sense that certain results may be sensitive and users may not want their information shared with other users of the system. Therefore segregated storage access is essential.

AWS S3 is a low-cost, large scale, object storage platform with support for arbitrary file types. S3 has a concept called buckets which are unique throughout the entirety of S3 and are used to reference the storage location of data. SCARV Lab has separate buckets for both acquisition data and analysis data. Results from all tasks are stored in these two buckets, but is segregated between users to maintain privacy.

Segregation of Storage

Segregation within buckets is a complicated procedure and requires the interaction of many AWS components as well as external systems. Auth0 supports the OpenID Connect specification [59] which is a layer on top of the OAuth 2.0 protocol allowing verification of the identity of an end user based on responses from an authentication server. SCARV Lab registers Auth0 with AWS Identity and Access

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:List*",
                "s3:Get*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::scarv-lab-traces/${cognito-identity.amazonaws.com:sub}",
                "arn:aws:s3:::scarv-lab-traces/${cognito-identity.amazonaws.com:sub}/*",
                "arn:aws:s3:::scarv-lab-analysis/${cognito-identity.amazonaws.com:sub}",
                "arn:aws:s3:::scarv-lab-analysis/${cognito-identity.amazonaws.com:sub}/*"
            ]
        }
    ]
}
```

Figure 3.4: SCARV Lab storage policy.

Management (IAM) as an identity provider supporting OpenID Connect. This allows AWS to uniquely identify SCARV Lab users and create a mapping between Auth0 identities and internal AWS identities. In order to assign permissions to these internal identities, they need to be federated. AWS Cognito Identity Pools (IDP) does this by referencing the Auth0 identity provider within IAM and creating an AWS identity pool with access roles setup for unauthenticated and authenticated users.

Figure 3.4 defines the access role for authenticated identities. This allows clients to download and list all items within their private area. In order to segregate user storage, a user's root directory must be unique throughout the entire bucket and must only accessible by themselves. This is accomplished through the uniqueness property of Cognito identity pool subjects (usernames). The root directory of a user's data in S3 is their AWS federated identity subject name, denoted as `sub` in the JSON document of Figure 3.4. One will notice that upload of documents is not permitted by authenticated users. This is done intentionally to prevent the SCARV Lab storage area being misused and instead being turned into a file hosting service for users.

Accessing Private Data

Clearly with a storage system that is segregated, access to user files becomes more complicated as authentication procedures have to be introduced for data retrieval. This is handled through a custom API endpoint within SCARV Lab and is divided into two stages as follows:

1. A user requests the download of a specific object within the SCARV Lab storage system. Their ID token is extracted from the request and sent to AWS Cognito IDP for verification.

2. Cognito IDP contacts Auth0 requesting verification of the submitted ID token. If successful, parameters such as AWS federated identity name, AWS access tokens, and token expiry times are returned.

3. The returned credentials are then used to generate an ephemeral download link for an S3 object. If a user is requesting something that is out of their designated private area, this process will fail. Subsequently, if successful, the URL is then returned as a HTTP 302 redirect and the user is prompted to download the requested object from the SCARV Lab storage system.

The above process is entirely abstracted away from the user as long as they are logged into the SCARV Lab web interface.

3.1.9 SQS and Lambda

In order for results to be downloaded by users through a web interface, they must be in an appropriate format that represents the results as a single file such as `zip`, `tar` and so on. Therefore, there needs to be a post-processing facility which converts the results of side-channel acquisition or analysis into one of the aforementioned formats.

AWS SQS and Lambda perform this "archiving" process. When a job is complete, the results are uploaded to S3 by the relevant pieces of infrastructure. This action places the `remark` field of the job manifest within DynamoDB into an `archiving` state. This subsequently, appends elements of the particular job manifest into an SQS queue. These elements include the job identifier and the federated identity subject name from AWS.

SCARV Lab employs a Lambda function to consume items from the SQS queue. This function will then recursively zip up the results from the particular job and place the zip file into the user's private area within S3. Once this process is complete, Lambda changes the `remark` field of the job manifest within DynamoDB to `complete`. This completed state then allows a user to request a download link for the job results through the web interface.

Utilising SQS as a decoupling mechanism between side-channel result upload and Lambda prevents vital pieces of infrastructure having to wait for Lambda to finish archiving a set of results when instead they could be performing more side-channel related tasks. Furthermore, if lots of items are on the queue at once, this delay could be significant. Allowing Lambda to directly feed from the SQS queue means that scalability of the Lambda functions is handled easily based on the number of items within the queue. Serverless technologies such as Lambda suit this particular requirement well because the results are not subject to strict latency requirements and are uploaded infrequently enough such that Lambda is more cost effective than a traditional IaaS archiving system.

3.1.10 Development Practice

As with almost all projects, source code management is vital if a project is to be maintainable. SCARV Lab uses Git for version control and all repositories are hosted on Github. Appropriate code management techniques are also used which include development within feature branches before making pull requests into master branches. In addition to this, cloud-based configurations are versioned internally within AWS to facilitate easy roll-back if the need should arise.

Continuous Integration and Continuous Deployment

Continuous integration and continuous deployment are extremely important in order to manage updates to the entire SCARV Lab infrastructure. Without it, one would have to coordinate updates between individual components manually which takes time and also risks the availability of the overall service if errors occur. SCARV Lab manages CI/CD using Github and AWS Code Pipeline.

SCARV Lab's CI/CD strategy operates in much the same way as the typical example demonstrated in Figure 2.6 of the previous chapter. The website and API are hosted collectively in a single Git repository on Github. Within this repository, a protected master branch exists which serves as the deployment branch such that it represents the code running on the production servers within AWS. To update this branch the following sequence of steps must occur:

1. One attempts to update the master branch by opening a pull request from another feature branch. The code changes are then reviewed by an administrator and merged into master if approved.
2. This action of merging into the master branch activates a web-hook within AWS Code Pipeline.
3. The code within the master branch is then built into a Docker container based upon the Dockerfile hosted within the repository.
4. Upon build completion, the Docker container is securely uploaded to AWS ECR and tagged as "latest" by following a build specification file within the repository.
5. The relevant AWS ECS service is informed that a new Docker image is ready for deployment. Subsequently, the service is updated to run a new task definition based upon the new container. This then triggers an incremental rollout of the new container by creating new tasks and destroying old ones.

3.2 Acquisition

Acquisition of power traces represents a core element of the side-channel as a service infrastructure. As Figure 3.1 details, it is one of the few elements outside of the AWS cloud. SCARV Lab currently hosts the acquisition server within Bristol University and is comprised of a PicoScope and a SCALE development

```
{
    "depo-id": "s3",
    "depo-spec": {
        "identity_id": "eu-west-1:a886be37-7bdd-4ea4-90a8-525de62b3ef0",
        "verify": true
    },
    "device-id": "scale-m3/ps2206b",
    "driver-id": "block/enc",
    "driver-spec": {},
    "id": "d69724c5643e0fd8055a124fc62fe7fdeed6c79e54156efcf27ad0147888c2dc",
    "remark": "complete",
    "repo-id": "git",
    "repo-spec": {
        "tag": "master",
        "url": "git@danpage_lab-target:danpage/lab-target.git"
    },
    "status": 0,
    "submit_time": 1553429570,
    "trace-spec": {
        "compress": true,
        "content": [
            "signal",
            "m",
            "c"
        ],
        "count": 1500,
        "crop": true,
        "format": "trs",
        "period-id": "auto",
        "period-spec": 0,
        "resolution-id": "auto",
        "resolution-spec": 0
    },
    "user": "auth0|5ba6c4aecbe8bc6b34270a62",
    "version": "0.1.0"
}
```


Figure 3.5: Sample acquisition job syntax.

kit as in Figure 2.9 of the previous chapter. This system is directly connected to a workstation running a Python server. This server periodically requests pending acquisition jobs from the SCARV Lab API. This request also includes query parameters so that the type of jobs available for acceptance can be indicated to the API by the acquisition server.

Polling for jobs is more effective than pushing in a scenario such as this due to network restrictions. For example, to push a job, one would need to set up either a public IP address for the server or perform port forwarding which is non-trivial in an enterprise network with a large administration overhead. Even when this setup is complete, the SCARV Lab API still has to register this server and then push jobs to it which creates additional complications. With a polling system, networking is handled easily due to states automatically created within firewalls for egress traffic from hosts. Consequently, the API only has to dispatch jobs to servers who make a request. Further to this, the expansion of the acquisition system is easy. One can simply set up more acquisition servers to poll the SCARV Lab API and jobs will be dispatched evenly between them, providing there are enough jobs for all servers.

Once a pending job has been dispatched to the acquisition server it transitions into the `processing` state. This is then reflected to the user through the web interface. The acquisition server then reads the job manifest and programs the SCALE development kit with the software detailed in the job manifest. The PicoScope is then set up and the acquisition of power traces begin. When power traces have been collected, the results are submitted directly to S3 and the SCARV Lab API is notified, which then begins the archiving process detailed in Section 3.1.9.

Job Syntax

Critical to the success of the acquisition system is an expressive and concise job manifest format. Figure 3.5 details a sample acquisition job manifest which contains the following important elements:

- **version** : Specifies the job manifest version to facilitate future changes easily.
- **id** : Job id. This is a unique string specific to the particular acquisition job in question and is calculated via hashing the JSON document.
- **device-id** and **driver-id** : Represents information specific to the board type one wishes to execute on and details pertaining to the operation of the cryptographic software. For example, `block/enc`

indicates a block cipher running in the encryption direction.

- **repo-id** and **repo-spec** : Details the repository information for the cryptographic software one wishes to test. Here a Git URL can be specified for cloning along with other parameters such as deploy keys.
- **depo-id** and **depo-spec** : Details information as to where the results should be uploaded. The `identity_id` field indicates the user directory within S3 so that results can be uploaded to the segregated storage system.
- **trace-spec** : Defines relevant operations relating to the capture of traces and the format they are stored in. Options pertaining to the resolution, period and number of traces to collect can be specified. In addition, the raw content of traces can be itemised in an ordered list to include elements such as messages, ciphertexts and the trigger signal within the results.

3.3 Analysis

Like the acquisition component, a side-channel analysis system is critical to SCARV Lab's service offering as a whole. The analysis is done within AWS using the somewhat well-known Jlsca library introduced in the previous chapter.

Users can submit jobs to SCARV Lab for analysis through the web interface by specifying an identifier for a completed acquisition job. At regular intervals, EC2 machines poll the SCARV Lab API in a similar fashion to that of the acquisition servers. The EC2 machines run a Python server which fetches pending analysis job manifests from the SCARV Lab API. This job is then parsed and the acquisition job identifier is used to fetch the acquisition job data from S3 for analysis. This acquisition data can come in a variety of formats from CSV, Python pickle or Riscure's TRS format. In order to make use of Riscure's Jlsca library, it is imperative that a pre-processing function is employed to convert the data into the TRS format which can be accepted by Jlsca.

As the analysis servers are written in Python, but Jlsca is written in Julia, the DPA engines cannot be trivially invoked as a Python function. Instead, Python's subprocess module is utilised to invoke an analysis task by specifying an attack framework file and the converted acquisition job data. The attack framework file is custom and one has to be created for every attack. The following parameters must always be specified:

- **Attack Type** : This parameter indicates the attack type to Jlsca. Possible values include DPA and CPA. It effectively selects the appropriate analysis engine within Jlsca.
- **Intermediary Value Choice** : If a DPA attack is chosen, one must specify target intermediate values. For example, the AES S-box.
- **Analysis Method** : This parameter effectively controls the functionality of the distinguisher in Figure 2.8. Values relate to metrics such as the Pearson correlation coefficient.
- **Attack Direction** : This controls whether the attack is focused on encryption or decryption.
- **Key Length** : This specifies the length of the key one is attempting to recover. For example, in AES the valid options would be 16, 24 and 32 bytes which reflects the available key lengths in the AES standard.
- **Multiprocessing Options** : This will be discussed later in the research section. However, such parameters control how the work is split between multiple machines networked together if one wishes to do so.

Given the vast array of options available when specifying an attack file in Jlsca, it is easy to see how many possible variations exist. Currently, due to the prototyping nature of the analysis system, these values are fixed and only a certain number of attacks are available. However, the API has been engineered with future expansion in mind by already allowing users to select such options even though at present, the analysis system will ignore them.

Once Jlsca is running in a separate thread, the results of the analysis are captured from `stdout` and written to a log which details information pertaining to the correlation of individual key bytes, and the recovered key if found. In addition to this, graphs indicating the correlation for all key hypotheses are included by plotting intermediary outputs from Jlsca with matplotlib [60]. This data set is then bundled and submitted to the appropriate location within S3 ready for user retrieval.

```
{
    "acquisition_id": "4a8e8d616915aca35a7b36892ff08a3d8067cf77857b06057d1d2babc167e40f",
    "depo-id": "s3",
    "depo-spec": {
        "identity_id": "eu-west-1:a886be37-7bdd-4ea4-90a8-525de62b3ef0"
    },
    "id": "40225f650733d2913037a6bf969b332f42a748543dc4d7fcb8194694d5e59b87",
    "remark": "complete",
    "status": 0,
    "submit_time": 1553449223,
    "user": "auth0|5ba6c4aecbe8bc6b34270a62",
    "version": "0.1.0"
}
```


Figure 3.6: Sample analysis job syntax.

Job Syntax

Figure 3.6 shows an example analysis job manifest. It includes these key elements that are crucial to the operation of an analysis task:

- **acquisition_id** : Acquisition job id. This references data from a previously completed acquisition job.
- **depo-id** and **depo-spec** : Details information as to where the results should be uploaded. The `identity_id` field indicates the user directory within S3 so that results can be uploaded to the segregated storage system.
- **id** : Job id. This is a unique string specific to the particular job in question and is calculated via hashing the JSON document.

Potential to scale

As with all elements of SCARV Lab, scalability is crucial to long term success in order to handle multiple analysis requests in a reasonable amount of time. The analysis system can be scaled in a similar way to that of the acquisition system. As the EC2 servers simply poll the SCARV Lab API for pending jobs, one simply needs to instantiate more servers to increase the rate at which analysis jobs are completed. This process is actually very simple; an image of the analysis server's operating system is saved and can be easily booted onto any physical machine within AWS in a matter of minutes. This coupled with an auto scaling group based on the number of pending items in the analysis queue allows for dynamic scalability to take place automatically.

3.4 Testing

As with any production level system deployed it is imperative that a reasonable level of testing has been performed in order to verify some of the aforementioned claims with respect to scalability and resilience. In addition to this, it is important that one is able to assert a degree of confidence in the operation of SCARV Lab as a system as a whole.

3.4.1 Scalability

In order to test the scalability of the website and API systems of SCARV Lab a series of tests were devised:

1. Continuous API and web requests.
2. Simultaneous downloads for hosted resources.
3. Authentication requests.

Continuous API and web requests

All subsequent tests involved creating a set of EC2 servers and one of the most important factors for network stress testing is a high bandwidth network connection. Most EC2 servers offer a base rate and a burstable rate bandwidth. One can then utilise the burstable rate for a short period of time. A `t3.medium` instance on EC2 is relatively low cost and provides good CPU performance with a burstable network bandwidth of up to 5Gbps [61], thus making it a suitable choice for stress testing.

To analyse request based traffic, two tests were performed; requests to the website root page and a database request to the API to gather user jobs. The absence of a local web cache is essential. If one was present, `HTTP 304 NOT MODIFIED` responses would be seen from the API and website, indicating to a client that the locally stored version is up to date. This obviously translates to very little load from a server standpoint and thus is not an accurate test of scalability. In both cases, TLS sessions are established between the client and the AWS load balancers. The scaling of these load balancers is handled by AWS internally and out of SCARV Lab's control. Therefore, testing was done by establishing a continuous TLS session to improve the throughput of requests to the website and API by avoiding extra time spent in session negotiation for subsequent requests.

When performing successive requests to the website root page from the EC2 machines, the overall ECS cluster CPU load began to increase. As mentioned in Section 3.1.5, the CPU threshold for the cluster is 60%. At minute intervals a task was spawned onto physical hardware until the overall cluster CPU load fell below the 60% threshold. When the stress test was stopped, the reverse happened and tasks were stopped every minute as the overall load fell well below 60%. As the web server and API are hosted within the same container, the exact same findings occurred when testing the API endpoint requesting all jobs for a specific user. The only difference with the API endpoint test was that a scaling event was triggered for DynamoDB in which the read capacity had to be increased for a short period of time to handle the increase in database requests from the API server.

The scalability of the acquisition and analysis systems are somewhat different from that of the website. The analysis system can be scaled very effectively within AWS according to the method described in Section 3.3. However, the acquisition system is external and thus cannot benefit from dynamic scaling like components within AWS due to lack of physical resources. Nevertheless, the acquisition system scaling methodology was described in Section 3.2 and simply involves the provision of more acquisition servers and linking them to the SCARV Lab API. In order to protect these services from a denial of service attack by malicious users creating many jobs per second, a limit of 3 pending jobs has been created per user. As such, a user is prevented from submitting new jobs via both the API and website when 3 outstanding jobs are held in the queue under their username. This not only helps from a scalability standpoint, but also establishes a fair use policy by preventing overactive users from starving other users who have a small number of jobs in the queue. The system effectively ensures every user has *equal priority* for both the analysis and acquisition system.

Continuous download requests for hosted resources

SCARV Lab's storage system is designed such that file transfers happen independently of the website and API servers, thus eliminating a potential bandwidth bottleneck. It does this by generating ephemeral URLs as described in Section 3.1.8. The previous test demonstrates that API endpoints are able to scale with continuous requests. Therefore, once a URL is distributed, the scalability of downloads relies solely on the scalability of S3 which is managed automatically by AWS internally. One small limitation to the current architecture is that the entirety of SCARV Lab is hosted within the Ireland region of AWS's cloud infrastructure. Thus, slower download speeds may be incurred in other locations around the world due to network peering limitations. However, this could easily be improved by using the content delivery network CloudFront [62]. AWS CloudFront could be used to replicate SCARV Lab resources around the AWS worldwide network to enable high bandwidth downloads anywhere in the world.

Authentication requests

Authentication is a core component of all systems surrounding the SCARV Lab API. Without it, elements involved in aspects such as segregated storage and private data management would cease to function. Therefore, the scalability of this service is key to ensuring overall availability as a whole. The endpoints of the SCARV Lab API have been shown to scale due to the results of the previous test. Therefore, the scalability of the authentication system rests on Auth0's API system. Currently, SCARV Lab is set up as a free tenant whereby the following restrictions are in place:

- Hosted login page is restricted to 300 requests per minute.
- Requests to JWT related endpoints are limited to 30 requests per second.

This is more than adequate for a small scale application. Furthermore, in future, if SCARV Lab decided to purchase a subscription from Auth0, the login page rate limit rises to 500 requests per minute restricted on a per IP address basis. However, in practice, even on the current free tenant plan, as soon as one authenticates, the web session is valid for one hour. Therefore, in order to reach such a limit, one would have to have a service which averaged 5 new logins per second. In addition to this, the SCARV Lab API communicates directly with Auth0's JWT endpoints and subsequently is only limited to 30 authentication requests per second.

3.4.2 Resilience

Error recovery and the ability to handle exceptions is of paramount importance for any production level system. Without controlled exception handling, a malicious user has the potential to disrupt the availability of the service as a whole by sending illicit requests.

One of the most well known malicious attacks is cross-site scripting (XSS) [63]. An attacker may attempt such an attack through the submission of a malicious job manifest via the SCARV Lab API which can then be shared to unsuspecting users. However, upon upload of a job manifest, its content is parsed and it must be a valid JSON job manifest document before being stored in the database. For example the upload of arbitrary text such as:

```
<script>doSomethingEvil()</script>
```

simply is not allowed and will instead return a `HTTP 400 BAD REQUEST`, thus mitigating the possibility of XSS attacks via submitted data.

All components within SCARV Lab use exception handling to indicate to the user the status of their request. This is communicated via the use of HTTP response codes and a `status` field within returned JSON documents. Responses always fall into one of two categories:

- **HTTP response code** : This case is reserved for requests that are unauthenticated, contain invalid payload or are directed at an endpoint which doesn't exist.
- **200 HTTP response code and JSON status code** : This is utilised when users have successfully authenticated and have a genuine request in the form of a valid payload to an existing endpoint. The outcome of the request is communicated by a `status` field within a returned JSON document. In the case of success, a 0 is always returned. However, if the request failed, then an error code is returned which can be mapped to a string via an integer enumeration allowing for causes of failure to be easily communicated to users.

Clearly in certain cases, exceptions may arise which cannot be controlled or are unforeseen. These are commonly known as Internal Server Errors and are communicated to clients via the use of the HTTP response code; 500. If not properly managed, these errors can cause an entire application to crash. SCARV Lab API mitigates this issue completely using two approaches. Firstly, Flask is designed to recover from internal errors in order to keep the server running. Secondly, AWS ECS regularly monitors the health of instantiated tasks within the cluster. If one is unresponsive or deemed unhealthy, then it is swiftly replaced by a new one. These two approaches ensure that the availability of SCARV Lab is never compromised.

3.4.3 Correctness

An important part of any testing ritual is validity testing. This verifies that the system performs correctly under normal parameters.

Acquisition Tasks

For acquisition tasks, a template repository [64] hosted on Github was set up. This serves as an open-source blueprint allowing developers to access a framework to develop cryptographic software for supported acquisition server hardware. For correctness testing, this setup was forked and a simple AES encryption kernel was written. This forked Github repository was then referenced within the job manifest file as shown in Figure 3.5. A large variety of job manifests were submitted based around this

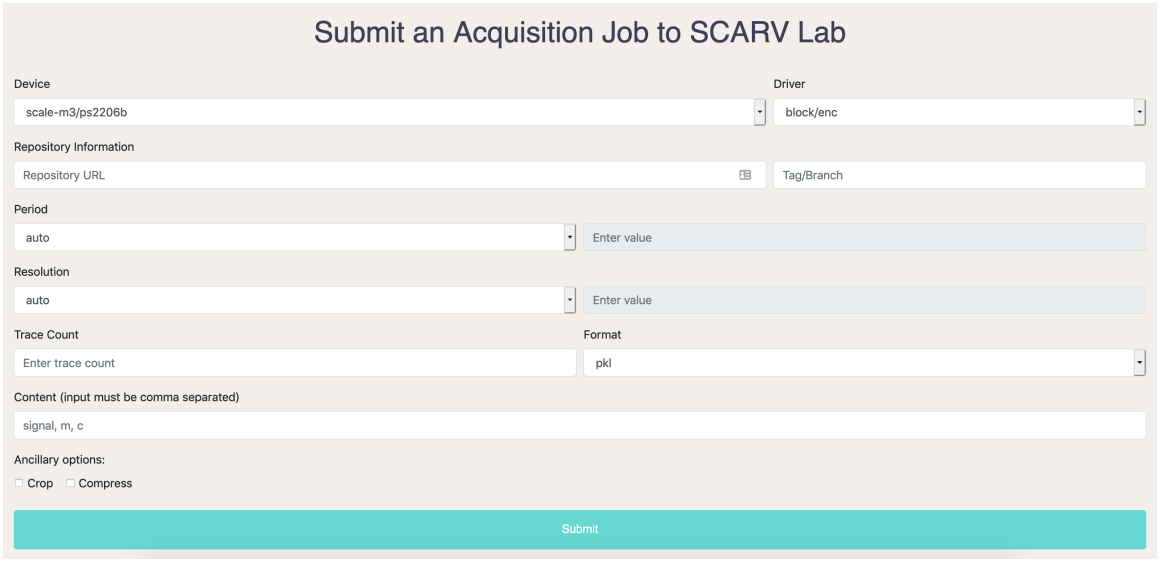

Figure 3.7: Acquisition job submission form.

Figure 3.8: Acquisition jobs status page.

single kernel. The differences being the options held within the job manifest. For example, resolution, compression, cropping and so on. These parameters were specified via the online web interface as shown in Figure 3.7. However, this could of course have been performed through the API instead. These jobs were then polled for by the acquisition servers and the results successfully uploaded to S3. A user is able to track the process of jobs through the web interface as shown in Figure 3.8. When a job has completed, trace sets relating to the job manifest along with an execution log can be downloaded from the website. A sample execution log can be seen in Appendix A.

Analysis Tasks

In order to check the functionality of the analysis tasks on real data, the trace sets from the sample kernel developed for acquisition testing were used. The submission of an analysis job can be done in much the same way as an acquisition job. However, the difference is that an acquisition job ID must be specified to indicate the data one is wishing to perform side-channel analysis on. Once submitted, the job manifests are downloaded by the analysis servers and analysis begins according to the process specified in Section 3.3. A user is then able to track the progress of their tasks akin to the method shown in Figure 3.8 for acquisition. Once the analysis is complete, results are uploaded to S3 and a user is able to download a job execution log as shown in Appendix B and plots indicating correlation of key bytes as shown in Figure 3.9.

Acquisition data sharing was also tested. As any previously completed acquisition job can be submitted for analysis, a user is free to share specific jobs with colleagues and friends to enable analysis to

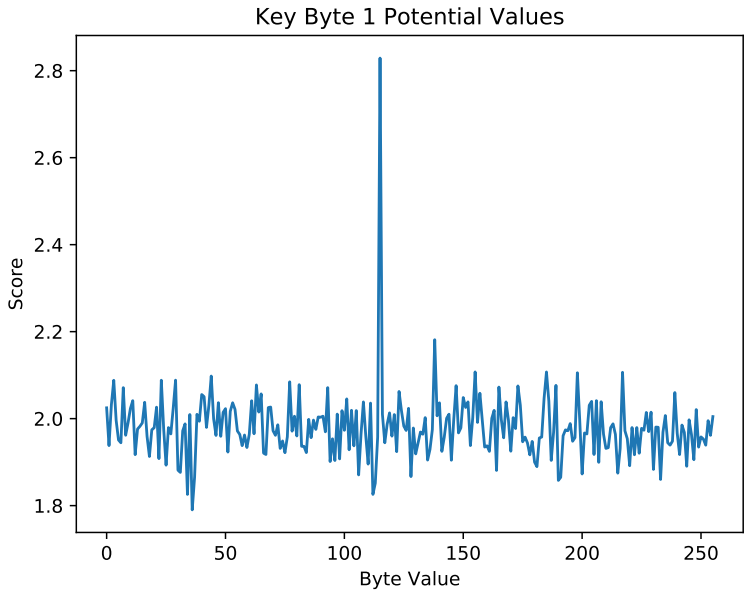

Figure 3.9: A correlation graph for a particular key byte on an analysis job.

be performed on specific trace sets. This is particularly ideal for environments such as competitions and assessments. Furthermore, the analysis results for an arbitrary acquisition data set is actually stored under the username of the user invoking the analysis task. Thus, analysis results can be kept private from the original owner of the data set if required.

Chapter 4

Critical Evaluation

4.1 Improving the efficiency of the analysis system

The analysis system is a key element of SCARV Lab as a whole. As described in the previous sections, all analysis tasks are performed using the Jlsca library running on infrastructure within the AWS cloud environment. This section aims to explore potential optimisations that can be achieved and evaluate their effectiveness in terms of execution speed and monetary cost.

At the time of writing this document, the author of the Jlsca library states the following:

> *"I parallelized components in Jlsca because I wanted to run and split over multiple machines, but so far I haven't found the time to experiment with that" [18].*

Before advancing into this topic, it is first important to give an insight into how distributed multi-processing works at a high level within the Julia language. Julia provides a module called "Distributed" [65]. This is included in the standard Julia library and provides an implementation of distributed memory parallel computing. Julia provides this multi-processing environment based on message passing. This facilitates the ability for programs to run on multiple processes in separate memory domains at once. This message passing is based upon two primitives; remote references and remote calls. Delving into these concepts is out of the scope of this document, but simplistically a remote reference refers to an object stored on a particular process whilst a remote call requests the invocation of a function on a process.

Whilst there are a variety of potential optimisations that can be performed on the analysis system, a large portion of these would represent projects in their own right. Therefore, only two optimisations will be discussed, AWS machine types and distributed multiprocessing. These optimisations are possible to evaluate within a cloud-based environment due to the large amount of computing resources available. In addition, both of these optimisations tie in neatly with respect to economic analysis. For example, a more powerful machine with more resources may decrease the total execution time for an analysis task, but is it worth the increase in cost? These types of questions will be evaluated in Section 4.2 based on the data available from these optimisations.

4.1.1 AWS Machine Types

Introduction

Within AWS there exists a plethora of machine types available. Most shared tenancy machines vary by two factors; CPU burst time and RAM size. Most of the modern and affordable instance types available on AWS provide two virtual CPU cores with a limit often enforced on computation. These limits provide a baseline of performance with the opportunity to burst up to the full CPU capacity for a short period of time. These machines are cheaper and are often suited to workloads which vary in intensity. The second differing option between machines is that of available RAM. RAM provides potential speedups if the computer has to work on large amounts of data. For example, if one is able to store an entire data set in RAM, computation upon that set will be much faster than if the CPU has to fetch parts of the set from the backing store. Both of these factors may play a large role in the overall runtime of analysis tasks.

The AWS instances selected for the experiment are listed below:

- **t3.micro** - General Purpose : 1GB RAM, 2vCPUs @ 2hr 24min burst, $8

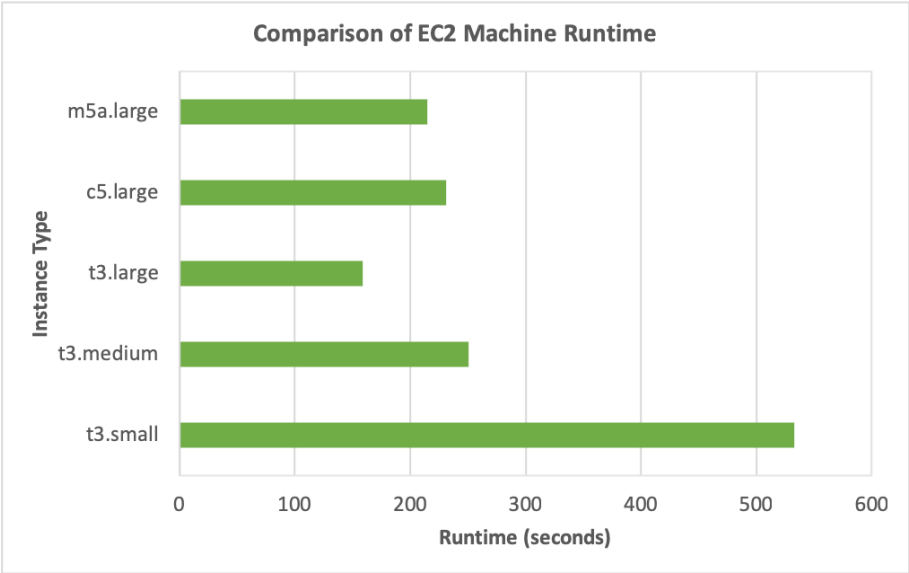

Figure 4.1: A graph showing the difference in runtime for comparable EC2 instance types.

- **t3.small** - General Purpose : 2GB RAM, 2vCPUs @ 4hr 48min burst, $16
- **t3.medium** - General Purpose : 4GB RAM, 2vCPUs @ 4hr 48min burst, $33
- **t3.large** - General Purpose : 8GB RAM, 2vCPUs @ 7hr 12min burst, $67
- **c5.large** - Compute Optimised : 4GB RAM, 2vCPUs, $70
- **m5a.large** - General Purpose : 8GB RAM, 2vCPUs, $70

where pricing is approximate and is based upon purchasing a Linux on-demand instance for one month.

Clearly, the test above focuses on a small subset of the overall machine types available within EC2. The first reason for doing this is due to monetary cost. It is important to understand that the primary goal of these optimisations is to provide the most efficient analysis system for the least amount of money. Machine types more powerful than this tend to range into the region of hundreds of dollars per month which is infeasible for a startup service such as SCARV Lab. Secondly, some cheaper or similarly priced machines are older generation EC2 instance types. For example, there isn't much point in testing a t2.small machine when it's direct successor, t3.small is available with more compute resources and a lower monthly price. Thirdly, the machines chosen above all have the same number of virtual CPUs. This means that the test accurately compares the impact of RAM, burst time and the difference between compute optimised instances.

Hypothesis

An increase in physical machine resources will result in a decrease in total execution time for an analysis task.

Results

The analysis task used to perform the test contained 1000 traces and was based around an AES kernel performing encryption with a 128 bit key. The log for the analysis job can be seen in appendix B.

Figure 4.1 details the runtime of the analysis task across the different machine types. The t3.micro result is omitted from the chart because the runtime was in excess of 2 hours which is an unacceptable amount of time for a single analysis job. From the results, it is clear to see that as RAM resources increase, the runtime of the analysis task decreases, thus validating the original hypothesis. When performing side-channel analysis tasks using Jlsca, the RAM utilisation of an EC2 machine increases to maximum capacity. In fact, in all cases, the amount of RAM wasn't enough and swap space had to be created to avoid the analysis task being killed by the kernel. EC2 file systems are network based and are accessed through a SAN fabric and swap space is no exception to this. This produces a significant communication

bottleneck when the system accesses the swap file, thus intensifying the effect of limited RAM on total execution time. For the particular analysis task in question, the amount of memory required was roughly 11GB at any one time. This meant that the swap space had to be used in all scenarios.

Interestingly, the difference in runtime between a t3.medium instance and a c5.large instance was marginal. This indicates that a compute optimised instance may not necessarily be the most attractive choice given the cost increase. However, one should note that the CPU computation on the cheaper general purpose machine was set at a burstable rate. Thus, would not be suitable if the analysis system was performing continuous analysis tasks, as the performance would significantly deteriorate due to this factor. Nevertheless, the comparison does show that one is able to burst up to the full capacity of two virtual CPUs for a limited amount of time.

Overall, from the results, it is clear to see that the t3.large instance provided the best performance for the given analysis task. This is an interesting contrast to that of the m5a.large instance which is comparable, but with no burstable limits. The main difference between these two instance types is the type of processor on the underlying bare metal machine. The t3.large instances use the Intel Xeon Platinum 8000 series processor whilst the m5a.large instances use the AMD EPYC 7000 series processor. This indicates that the difference in processor type also has a significant impact on the overall computation time of the analysis task. However, the limiting factor of the t3.large instance is the burstable compute time. This may make it infeasible for use when the execution of successive analysis tasks is required.

4.1.2 Distributed Multiprocessing

Introduction

Multiprocessing is the concept of splitting a task up between all available CPU cores on a machine by the use of threads. These threads can be executed independently of one another on separate execution units within different cores on a CPU. This concept gives rise to significant speedups for programs which have a large amount of independent computation that can be performed in parallel. This independent computation can be split into threads, executed on the multi-core CPU and the results collated together when execution is finished for all threads. If the program is able to be split into a number of threads equal to the number of cores on a CPU and if these threads operate truly independently of one another, then one can achieve a potential speedup of n times, where n represents the number of cores on the CPU in use. Distributed multiprocessing expands the above concept to a cluster of multi-core machines so that threads are executed on remote machine cores as well as local cores.

For the experiment, clusters consisting of three nodes were created in EC2 to enable distributed multiprocessing. This was done for each of the aforementioned machine types to provide a comparison between each of the clusters. In order for work to be distributed between each of the nodes within a particular cluster the following concepts were used:

- **Elastic IP addresses** : In order for each cluster member to communicate with other nodes, each node must know how to contact the other cluster members. Obviously, this can be done trivially with software that supports communication over a broadcast network. However, Julia requires each cluster member to know every other member's IP address. In order to prevent IP addresses changing between EC2 instance reboots and when switching instance types to test different clusters, AWS elastic IP addresses were used which are statically assigned to instances.

- **Elastic File System (EFS)** : When worker nodes within a cluster are instructed to perform computation on a particular data set, it is obvious that each worker node needs the underlying data set or a portion of it to work on. When using Jlsca, each worker must have the same underlying trace set present before work is split up between the nodes. To achieve this, EFS is used which creates a shared filesystem between worker nodes to ensure that each one always has the same view of the data set prior to computation commencing.

- **Julia's Machine File Command Line Argument** : In order to dispatch jobs to the worker nodes, Julia uses a command line argument `--machine-file`. This references a file which contains information for each of the nodes within the cluster such as username, IP address and SSH credentials. Julia then uses this information to invoke functions on other worker machines.

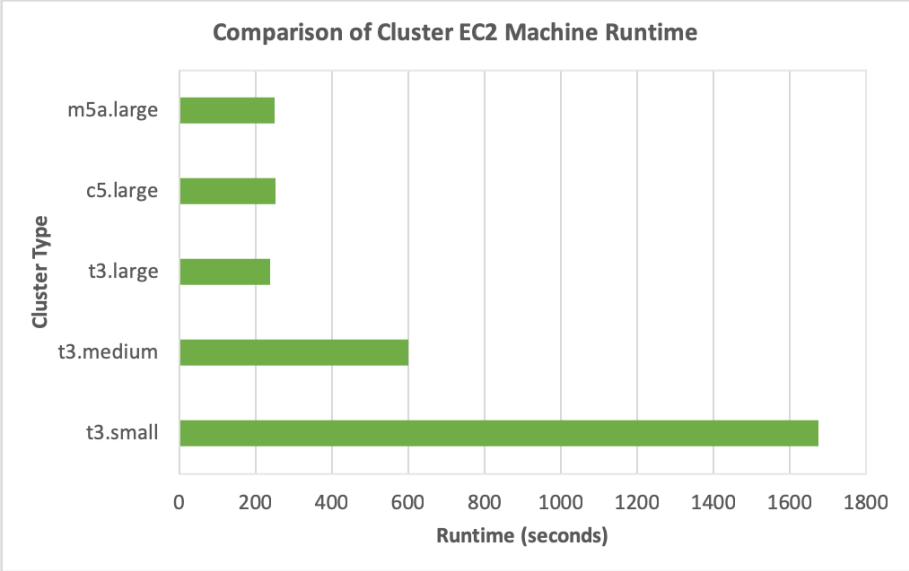

Figure 4.2: A graph showing the difference in runtime for 3 node clusters of EC2 instances.

Hypothesis

Distributed multiprocessing should provide a significant speedup for analysis tasks due to the amount of independent computations that are present. However, the speedup will greatly depend on the amount of resources available to each machine within the cluster.

Results

The analysis task used to test the performance of different EC2 clusters is the same as that in the previous test to provide consistency.

Figure 4.2 details the runtime of the analysis task over the five different clusters created. One will notice immediately that a very interesting result occurs. That is, runtime across all cluster types is significantly greater than that of a single instance of the corresponding type. This immediately invalidates the original hypothesis. The closest result to single instance performance is that of the c5.large cluster which features an increase in runtime of 22 seconds. However, this difference is relatively large, especially when coupled with the fact that the runtime is expected to decrease!

An explanation for these results is the potential communication overhead. During testing, it was noticed that all worker nodes were communicating with each other at extremely high data rates, often in excess of 100Mbps and in certain cases even bursting up to the physical port speed. This behaviour could be due to Jlsca splitting up computation on trace sets in an inefficient way. Given the fact that the author of Jlsca admits to never testing distributed processing, this theory seems plausible. In addition to this, one must recall that each of the worker instances uses a swap space to prevent execution failure when RAM is fully utilised, and that this swap space is housed within network-based storage and must be accessed over a SAN fabric. Consequently, having the work split up between multiple machines each having to use their own network-based swap space may compound the effect of high latency swap space further and contribute to the increase in runtime. The above tests were also performed with SCP scripts in place of EFS to ensure that workers had the same view of data prior to computation starting. Nevertheless, the same effects were experienced, indicating that EFS plays no part in the increased runtime of a cluster based analysis system.

Overall, it is clear to see that, a cluster-based analysis system would not provide any benefit in efficiency whatsoever. However, this may change in future if Jlsca improves the way in which work is distributed or if the latency of AWS swap space decreases.

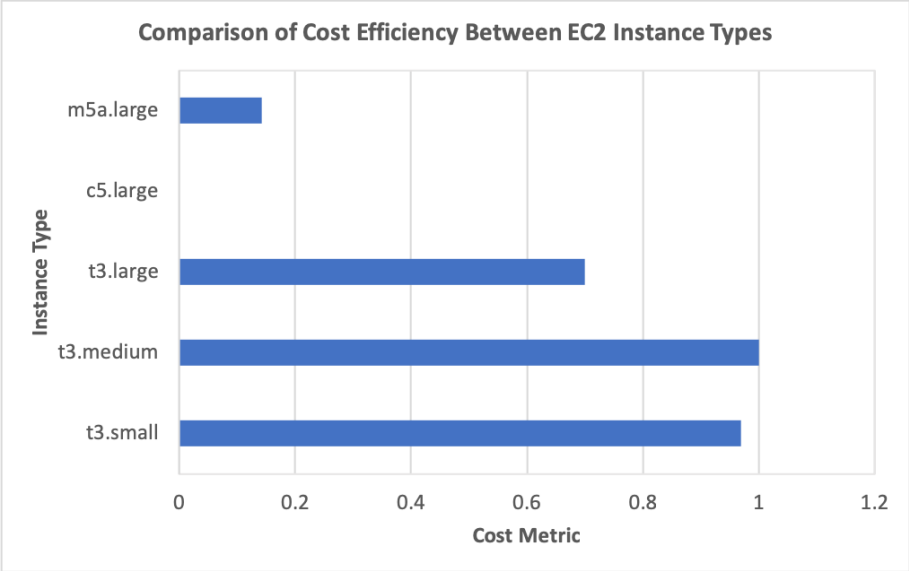

Figure 4.3: A graph showing the cost efficiency between EC2 instances.

4.2 Economic evaluation of the analysis system

Financial viability is crucial to the long term success of the SCARV Lab analysis system. Therefore, it is important to find a balance between performance and affordability. Based on the results in Section 4.1, it is clear to see that it would not be sensible from a performance or affordability aspect to choose a cluster-based analysis approach. Not only does this dramatically increase runtime for analysis tasks, but it also has the side effect of multiplying the monetary cost of all analysis tasks by the number of machines in a working cluster. This is certainly not a sensible approach to take. Therefore, one must resort to the paradigm of single instance analysis.

In order to accurately determine which instance is the most cost-effective for analysis, a metric needs to be defined. The results seen in Section 4.1.1 generally demonstrate that as instance resources increase, runtime decreases. From a monetary perspective, one would expect the runtime of an analysis task to be inversely proportional to the cost of an instance.

$$cost \propto \frac{1}{runtime} \implies cost = \frac{k}{runtime} \equiv k = cost \times runtime$$

Based on the formula above, the instance with the lowest value of k should be the most cost-effective. In order to display this data graphically, one must first normalise the data with the following formula to produce a cost-coefficient, c_i for each instance:

$$c_i = 1 - \frac{k_i - k_{min}}{k_{min} - k_{max}}$$

where k_i represents the value of k for a particular instance, k_{min} represents the minimum value of k for all instances and k_{max} represents the maximum value of k for all instances.

Figure 4.3 shows a chart displaying the normalised cost-coefficients for all instance types tested, where a value of 1 and 0 indicates the best and worst possible cost-coefficient respectively. From this chart, it is clear to see that a t3.medium instance provides the most cost-effective strategy for performing side-channel analysis tasks. However, it is important to recall that this particular instance type is limited by a 4hr 48min computation burst. Therefore, in order to scale the service appropriately, one should increase the number of t3.medium instances such that the rate of incoming analysis tasks is easily handled by the overall computational burst power in order to prevent the average runtime of analysis tasks dramatically increasing. However, given the low usage of the service currently, a single instance is perfectly acceptable and scaling can be automated during peak times as discussed in Section 3.3.

4.3 User Testing

A key element for the evaluation of any system is user testing. During the final stage of development of SCARV Lab, an effort was made to reach out to various individuals with the opportunity to try out the system as a whole and provide feedback on their experiences. Luke Mather from Cerberus Laboratories tested out the service and provided the feedback shown below:

> *I believe there is an incredible amount of value in the automation of side-channel testing and its integration within development lifecycles - particularly for open-source projects that are developed or repeatedly integrated by inexperienced or resource-constrained engineers on common platforms and in environments where side-channel attacks are of concern (e.g. use of ARM's mbedTLS library in "IoT" products). This project is a great first step in lowering the barrier to entry for this kind of testing: the acquisition of measurements comes with a significant up-front setup cost and is non-trivial to make repeatable, the quantity of systems deployed into hostile environments is only increasing, and so there's clear value to the open-source ecosystem, academia and potentially to some types of commercial entity.*
>
> *I think integration within CI frameworks is particularly valuable, and so as well as the addition of the hooks necessary to achieve that, I think a useful next feature is the inclusion of some generic-as-possible leakage detection tests with the end goal of supporting dashboard-style alerts if leakage is detected, for instance.*
>
> *Some commercial customers may have intellectual property concerns. Ultimately users must trust your service, but perhaps may not trust the cloud vendor or other third parties. IPR concerns may apply to code and acquisitions—in the latter case you could consider offering some form of encryption applied immediately after capture. The former case seems more complicated, but perhaps a user could supply an encrypted repository archive or pre-built library. In either case, the customer foregoes the ability to have the data analysed, and so a different approach entirely for any IPR-concerned customer may be to license the tooling and have the customer host some or all of the setup internally.*

Chapter 5

Conclusion

5.1 Summary of Achievements

This section directly addresses the state of the project with respect to the original aims and objectives listed in Section 1.4.2.

D1. An auto-scaling platform was certainly developed. All elements of SCARV Lab were built with dynamic autoscaling in mind from the foundation up. Even external non-scalable elements such as the acquisition system had a fair usage policy to help prevent oversubscription of available resources. From a cloud infrastructure perspective, all AWS components were engineered to be highly available. For example, SCARV Lab currently uses two application load balancers which map to the same underlying target groups, but are placed in different AWS availability zones. Therefore, if an availability zone (data centre) within a region were to fail, the service still remains operational. In addition to this, the ECS infrastructure is built with fault tolerance in mind by ensuring that as scaling events occur, instances are placed evenly between three separate availability zones within a region.

From a side-channel analysis perspective, EC2 instances were successfully set up with Jlsca, Julia and a Python-based server to allow for the execution of side-channel analysis tasks within a cloud-based environment.

D2. As shown throughout Chapter 3, a website and API system built on the Python Flask framework was created. The web-based systems do indeed facilitate the submission of side-channel acquisition jobs. One can submit an acquisition job via the simple to use, but extensive web interface or via the API. Both methods provide a secure way to submit jobs and the job manifest syntax as seen in Section 3.2 is expressive and extensible enough to allow for a wide array of parameters to be specified by a user wishing to capture power consumption data from a particular target implementation.

D3. As described in Section 3.1.6, a queuing system was created to allow for the provisioning of pending side-channel acquisition jobs to laboratory-based hardware. Section 3.2 details how a connected workstation periodically requests pending acquisition jobs from SCARV Lab, thus facilitating the ability to provision acquisition jobs to the laboratory hardware.

After power consumption data is captured by an acquisition server, secure upload is performed to the federated storage system as described in Section 3.1.8. This system is indeed a secure storage environment as acquisition results are entirely segregated between users through the use of identity federation. Further to this, to allow for the previously captured acquisition data to be easily analysed in the future, each analysis job is given a unique identifier which maps to an archive of the acquisition results which can be downloaded for further analysis in the future.

D4. Section 3.3 explains how the web interface supports the ability for users to submit an analysis job to SCARV Lab. When doing this, the appropriate acquisition job ID is the only usable parameter at present. In the future, the web form will be extended to allow users to add additional parameters into analysis job manifests. However, the analysis job manifests are similar to the acquisition job manifests in the sense that they are expressive enough to facilitate the ability for an analysis server to operate on previously acquired data sets by providing vital information such as data format, acquisition id, depository information and so on.

D5. Analysis servers have been set up within the cloud as demonstrated in Section 3.3. They are able to receive tasks via periodically requesting pending analysis jobs through the SCARV Lab API. Analysis of previously collected acquisition data is performed on EC2 instances and then the results are uploaded to the correct directory within the federated storage system ready for user retrieval via the website. In addition to this, the analysis system possesses the ability to scale if coupled with an auto-scaling group as described in Section 3.3.

R1. Within Section 4.1 both the effect of different AWS machine types and distributed multiprocessing were evaluated from an efficiency perspective with respect to the runtime of a typical analysis task. It was shown that distributed multiprocessing within an AWS cluster gave counter-intuitive results and as such, single instance analysis using a t3.large instance provided the best performance when tested against a multitude of other instance types.

R2. Section 4.2 utilises the results gathered from Section 4.1 to calculate the most cost-effective way to perform side-channel analysis in a cloud-based environment. To do this, a metric was defined denoted as the cost-coefficient, which factored in the monetary price of an instance and the time taken to perform a typical analysis task. Upon calculation of the cost-coefficient for each instance type, it was revealed that a t3.medium instance provided the most economically efficient approach to side-channel analysis within SCARV Lab.

5.2 Future Work

There is the intention to continue work on this project into the future. This section details the plans for the future development of SCARV Lab based on feedback from user testing and identification of areas of the system which could benefit from certain embellishments.

5.2.1 Integration with CI frameworks

A key element within the feedback provided during user testing was the potential for integration with a CI platform. This was also highlighted as a potential use case in Section 1.3.2. In the future, SCARV Lab will have the ability to act as a CI provider. However, instead of performing automated unit tests, side-channel acquisition and analysis will be performed automatically on a codebase through the use of web-hooks from a source code hosting platform such as GitHub. Subsequently, alerts pertaining to security vulnerabilities can then be configured to notify repository owners via email or through the SCARV Lab website.

5.2.2 Support for private repositories

Currently, a limiting factor of the side-channel acquisition system is that the target repository must be open sourced in order for side-channel acquisition to take place. One is currently able to alleviate this issue through the use of deploy keys in job manifests. However, seamless integration with private repositories through providers such as GitHub would be more user-friendly. One could potentially achieve this by registering SCARV Lab as a GitHub application or by ensuring that SCARV Lab's authentication platform accepts GitHub credentials as an OAuth provider to allow the retrieval of private repositories owned by a user or organisation.

5.2.3 Extending the analysis system

At present, the analysis system can only perform a fixed number of tasks and has limited customisability when submitting job manifests. This system should be improved in the future to provide the same level of fine-grained control that is currently offered with the acquisition system. This will allow users to customise their analysis job manifests to focus in on certain aspects that are most important to them. In the user testing feedback, leakage detection was mentioned as being a useful feature. The analysis system could be improved by adding embellishments such as these to provide a more personal service offering in the future.

5.2.4 Open sourcing the project

When utilising any product or service that has access to potentially sensitive source code, it is imperative that one trusts the vendor. This fact was highlighted in user testing whereby potential commercial customers may have particular intellectual property concerns. Therefore, it is critical that elements of SCARV Lab are open sourced so that users can verify the integrity of the system as a whole.

Open sourcing also has other benefits by encouraging community development of SCARV Lab. For example, one may wish to add support for a particular side-channel analysis method. Open sourcing allows a user to fast track the implementation of a particular feature and SCARV Lab also benefits by increasing its overall service offering.

5.2.5 Improving distributed multiprocessing with Jlsca and AWS

As demonstrated in Chapter 4, the combination of Jlsca and AWS results in extremely poor performance from a distributed multiprocessing standpoint. This was not expected at all and could be due to the way that Jlsca is splitting up work on trace sets between nodes within a cluster. Nevertheless, an investigation into this issue would be both intriguing and interesting. In addition to this, further study on this subject could potentially help to improve the efficiency of the overall analysis system if an appropriate solution is found.

Bibliography

[1] H. Aissaoui-Mehrez, P. Urien, and G. Pujolle, "Implementation Software to Secure Virtual Machines with Remote Grid of Secure Elements," *Proceedings - IEEE Military Communications Conference MILCOM*, 2014.

[2] "Containers vs. Virtual Machines (VMs): Whats the Difference?." https://blog.netapp.com/blogs/containers-vs-vms/. Accessed: 29-04-2019.

[3] N. Ruflin, H. Burkhart, and S. Rizzotti, "Social-data storage-systems," pp. 7–12, 2011.

[4] D. Page, "SCALE: Side-Channel Attack Lab. Exercises." http://www.github.com/danpage/scale. Accessed: 22-04-2019.

[5] "A beginner's guide to Amazon's Elastic Container Service." https://medium.freecodecamp.org/amazon-ecs-terms-and-architecture-807d8c4960fd. Accessed: 23-04-2019.

[6] FIPS, "Announcing the ADVANCED ENCRYPTION STANDARD (AES)," 2001.

[7] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "SoK: Security Evaluation of Home-Based IoT Deployments," *Georgia Institute of Technology*, 2018.

[8] "CAPS Assisted Products." https://www.ncsc.gov.uk/information/products-cesg-assisted-products-service. Accessed: 03-04-2019.

[9] "Riscure Labs." https://www.riscure.com/. Accessed: 03-04-2019.

[10] "Travis CI." https://travis-ci.org/. Accessed: 04-04-2019.

[11] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," 1999.

[12] "Lightweight Cryptography." https://csrc.nist.gov/projects/lightweight-cryptography. Accessed: 07-04-2019.

[13] NIST CSRC, "Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process," 2018.

[14] "eshard - esDynamic Platform." https://www.eshard.com/esdynamic-side-channel-analysis-framework/. Accessed: 07-04-2019.

[15] "CHES - Capture the Flag." http://ctf.newae.com/. Accessed: 07-04-2019.

[16] "DPA Contest." http://www.dpacontest.org/v4/index.php. Accessed: 07-04-2019.

[17] "TRS Format - PyPI." https://pypi.org/project/trsfile/. Accessed: 07-04-2019.

[18] "Jlsca Side Channel Framework." https://github.com/Riscure/Jlsca. Accessed: 07-04-2019.

[19] "Pysca Side Channel Framework." https://github.com/ikizhvatov/pysca. Accessed: 07-04-2019.

[20] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *ACM SIGOPS operating systems review*, vol. 37, pp. 164–177, ACM, 2003.

[21] "Container Security Risks." https://www.ontrack.com/uk/blog/the-world-of-data/using-docker-containers-beware-of-these-security-risks/. Accessed: 11-04-2019.

[22] "Docker Engine." https://docs.docker.com/engine/. Accessed: 11-04-2019.

[23] "AWS Elastic Container Service." https://aws.amazon.com/ecs/. Accessed: 11-04-2019.

[24] "AWS Simple Queue Service." https://aws.amazon.com/sqs/. Accessed: 11-04-2019.

[25] S. Sivasubramanian, "Amazon dynamoDB: a seamlessly scalable non-relational database service," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 729–730, ACM, 2012.

[26] "CAP Theorem." https://towardsdatascience.com/cap-theorem-and-distributed-database-management-sy Accessed: 11-04-2019.

[27] S. Gilbert and N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services," *Acm Sigact News*, vol. 33, no. 2, pp. 51–59, 2002.

[28] "AWS Amazon Aurora." https://aws.amazon.com/rds/aurora/. Accessed: 11-04-2019.

[29] "AWS DynamoDB Read Consistency." https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadConsistency.html. Accessed: 11-04-2019.

[30] "AWS Simple Storage Service." https://aws.amazon.com/s3/. Accessed: 29-04-2019.

[31] "AWS S3 Data Durability." https://docs.aws.amazon.com/AmazonS3/latest/dev/DataDurability.html. Accessed: 12-04-2019.

[32] "AWS Route 53." https://aws.amazon.com/route53/. Accessed: 29-04-2019.

[33] A. Forsyth, "Public Clouds for Software Developers: Serverless," *Slides from 2018 COMSM0010 - Cloud Computing.*

[34] "AWS Code Pipeline." https://aws.amazon.com/codepipeline/. Accessed: 18-04-2019.

[35] "AWS Elastic Container Registry." https://aws.amazon.com/ecr/. Accessed: 18-04-2019.

[36] S. B. Ors, F. Gurkaynak, E. Oswald, and B. Preneel, "Power-Analysis Attack on an ASIC AES implementation," in *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004*, vol. 2, pp. 546–552, IEEE, 2004.

[37] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology — CRYPTO' 99* (M. Wiener, ed.), (Berlin, Heidelberg), pp. 388–397, Springer Berlin Heidelberg, 1999.

[38] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 16–29, Springer, 2004.

[39] E. Oswald, "Applied Security, DPA on AES," *Slides from 2017 COMS30901 - Applied Security.*

[40] A. Garcia Asuero, A. Sayago, and G. Gonzlez, "The Correlation Coefficient: An Overview," *Critical Reviews in Analytical Chemistry*, vol. 36, pp. 41–59, 2006.

[41] P. Socha, V. Mikovsk, H. Kubtov, and M. Novotn, "Optimization of Pearson correlation coefficient calculation for DPA and comparison of different approaches," in *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pp. 184–189, 2017.

[42] D. P. Martin, L. Mather, and E. Oswald, "Two Sides of the Same Coin: Counting and Enumerating Keys Post Side-Channel Attacks Revisited," *Cryptology ePrint Archive, Report 2018/019.*

[43] M. Zheng, "Improved Results on Factoring General RSA Moduli with Known Bits," *Cryptology ePrint Archive, Report 2018/609.*

[44] R. Poussier, F. Standaert, and V. Grosso, "Simple Key Enumeration (and Rank Estimation) using Histograms: an Integrated Approach," *Cryptology ePrint Archive, Report 2016/571.*

[45] "Auth0." https://auth0.com/. Accessed: 20-04-2019.

[46] M. Jones, J. Bradley, and N. Sakimura, "RFC 7519 - JSON Web Token (JWT)," 2015.

[47] "Pico Technologies." https://www.picotech.com/. Accessed: 22-04-2019.

[48] "SCARV Organisation." https://github.com/scarv. Accessed: 22-04-2019.

[49] "Python Flask." http://flask.pocoo.org/. Accessed: 23-04-2019.

[50] "Waitress WSGI Server." https://pypi.org/project/waitress/. Accessed: 23-04-2019.

[51] "Bootstrap CSS Framework." https://getbootstrap.com/. Accessed: 23-04-2019.

[52] H. Zimmermann, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection," *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425–432, 1980.

[53] "AWS Fargate." https://aws.amazon.com/fargate/. Accessed: 23-04-2019.

[54] "DynamoDB Read/Write Capacity Units." https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadWriteCapacityMode.html. Accessed: 23-04-2019.

[55] "DynamoDB Partition and Sort Keys." https://aws.amazon.com/blogs/database/using-sort-keys-to-organize-data-in-amazon-dynamodb/. Accessed: 23-04-2019.

[56] D. Hardt, "RFC 6749 - The OAuth 2.0 Authorization Framework," 2012.

[57] "Auth0 Authorization Code Flow." https://auth0.com/docs/flows/concepts/auth-code. Accessed: 24-04-2019.

[58] "AWS Simple Email Service." https://aws.amazon.com/ses/. Accessed: 01-05-2019.

[59] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore, "OpenID Connect Core 1.0," 2014.

[60] "Matplotlib." https://matplotlib.org/. Accessed: 24-04-2019.

[61] "EC2 Network Interface Speeds." https://cloudonaut.io/ec2-network-performance-cheat-sheet/. Accessed: 29-04-2019.

[62] "AWS CloudFront CDN." https://aws.amazon.com/cloudfront/. Accessed: 01-05-2019.

[63] J. Garcia-Alfaro and G. Navarro-Arribas, "Prevention of Cross-Site Scripting Attacks on Current Web Applications," vol. 4804, 2007.

[64] "SCARV Lab Target - SCARV Project." https://github.com/scarv/lab-target. Accessed: 29-04-2019.

[65] "Julia Distributed Processing." https://docs.julialang.org/en/v1/manual/parallel-computing/index.html#Multi-Core-or-Distributed-Processing-1. Accessed: 22-04-2019.

Appendix A

Sample Acquisition Job Log

Below is a condensed acquisition job log generated by a SCARV Lab acquisition server with times and dates redacted.

```
configuration
 board-desc  = "..."
 board-id    = "scale/lpc1313fbd48"
 board-spec  = {
    "connect-id": "/dev/ttyUSB0",
    "connect-timeout": 10,
    "program-timeout": 120
 }
 depo-id     = "s3"
 depo-spec   = {
    "identity_id": "eu-west-1:a886be37-7bdd-4ea4-90a8-525de62b3ef0",
    "verify": true,
    "region-id": "eu-west-1",
    "bucket-id": "scarv-lab-traces"
 }
 device-id   = "scale-m3/ps2206b"
 driver-id   = "block/enc"
 driver-spec = {}
 id          =
  "75807cde871f2c55ae5661dc4bdda90f76b82b0b3a0156edb4f0dafe790cd3df"
 remark      = "processing"
 repo-id     = "git"
 repo-spec   = {
    "tag": "master",
    "url": "git@danpage_lab-target:danpage/lab-target.git"
 }
 scope-desc  = "..."
 scope-id    = "picoscope/ps2206b"
 scope-spec  = {
    "connect-id": "FU837/0324",
    "connect-timeout": 10000,
    "channel-trigger-id": "A",
    "channel-acquire-id": "B",
    "channel-disable-id": []
 }
 status      = 0
 submit_time = 1553096719
 trace-spec  = {
    "compress": true,
    "content": [
        "signal",
```

```
        "m",
        "c"
    ],
    "count": 10,
    "crop": true,
    "format": "pkl",
    "period-id": "auto",
    "period-spec": 0,
    "resolution-id": "auto",
    "resolution-spec": 0
}
user        = "auth0|5ba6c4aecbe8bc6b34270a62"
version     = "0.1.0"
construct board  object
construct scope  object
construct driver object
construct repo   object
construct depo   object
transfer local <- repo.
 execute
  | cmd : ['git', 'clone', '--verbose', '--depth', '1', '--branch',
           'master', 'git@danpage_lab-target:danpage/lab-target.git',
           'target']
  Cloning into 'target'...
  | result  : success (exit status = 0)
run driver -> process_prologue
 open  board
  execute
   | cmd : ['make', '-C', 'target', '--no-builtin-rules', 'deps-fetch']
    make[1]: Entering directory '${JOB}/target'
    make[1]: Leaving directory '${JOB}/target'
    Cloning into '${JOB}/target/build/scale-hw'...
   | result  : success (exit status = 0)
  execute
   | cmd : ['make', '-C', 'target', '--no-builtin-rules', 'deps-build']
    make[1]: Entering directory '${JOB}/target'
    make[2]:
     Entering directory '${JOB}/target/build/scale-hw/target/lpc1313fbd48'
    make[2]:
     Leaving directory '${JOB}/target/build/scale-hw/target/lpc1313fbd48'
    make[1]: Leaving directory '${JOB}/target'
   | result  : success (exit status = 0)
  execute
   | cmd : ['make', '-C', 'target', '--no-builtin-rules', 'build']
    make[1]: Entering directory '${JOB}/target'
    make[1]: Leaving directory '${JOB}/target'
   | result  : success (exit status = 0)
  execute
   | cmd : ['make', '-C', 'target', '--no-builtin-rules', 'report']
   make[1]: Entering directory '${JOB}/target'
   ELF Header:
    Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
    Class:                             ELF32
    Data:                              2's complement, little endian
    Version:                           1 (current)
    OS/ABI:                            UNIX - System V
    ABI Version:                       0
    Type:                              EXEC (Executable file)
```

```
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          139056 (bytes into file)
  Flags:                             0x5000200, Version5 EABI,
                                     soft-float ABI
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         2
  Size of section headers:           40 (bytes)
  Number of section headers:         10
  Section header string table index: 9
  Section Headers
   [Nr] Name              Type            Addr     Off    Size   ES Flg
   [ 0]                   NULL            00000000 000000 000000 00
   [ 1] .text             PROGBITS        00000000 010000 0014a0 00  AX
   [ 2] .data             PROGBITS        10000000 020000 0007c8 00  WA
   [ 3] .bss              NOBITS          100007d0 0207c8 0000d4 00  WA
   [ 4] .comment          PROGBITS        00000000 0207c8 00007f 01  MS
   [ 5] .ARM.attributes   ARM_ATTRIBUTES  00000000 020847 000031 00
   [ 6] .debug_frame      PROGBITS        00000000 020878 000180 00
   [ 7] .symtab           SYMTAB          00000000 0209f8 000e80 10
   [ 8] .strtab           STRTAB          00000000 021878 000663 00
   [ 9] .shstrtab         STRTAB          00000000 021edb 000052 00
  Key to Flags:
   W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
   L (link order), O (extra OS processing required), G (group), T (TLS),
   C (compressed), x (unknown), o (OS specific), E (exclude),
   y (purecode), p (processor specific)
  No version information found in this file.
  Attribute Section: aeabi
  File Attributes
   Tag_CPU_name: "Cortex-M3"
   Tag_CPU_arch: v7
   Tag_CPU_arch_profile: Microcontroller
   Tag_THUMB_ISA_use: Thumb-2
   Tag_ABI_PCS_wchar_t: 4
   Tag_ABI_FP_denormal: Needed
   Tag_ABI_FP_exceptions: Needed
   Tag_ABI_FP_number_model: IEEE 754
   Tag_ABI_align_needed: 8-byte
   Tag_ABI_enum_size: small
   Tag_ABI_optimization_goals: Aggressive Size
   Tag_CPU_unaligned_access: v6
  make[1]: Leaving directory '${JOB}/target'
 | result  : success (exit status = 0)
execute
 | cmd : ['make', '-C', 'target', '--no-builtin-rules', 'program']
  make[1]: Entering directory '${JOB}/target'
  make[1]: Leaving directory '${JOB}/target'
  Open On-Chip Debugger 0.10.0+dev-00689-g6c2020eb-dirty
  Licensed under GNU GPL v2
  For bug reports, read http://openocd.org/doc/doxygen/bugs.html
  adapter speed: 10 kHz
  adapter_nsrst_delay: 200
  cortex_m reset_config sysresetreq
  Info : J-Link V9 compiled Oct 25 2018 11:46:07
```

```
    Info : Hardware version: 9.30
    Info : VTarget = 2.615 V
    Info : clock speed 10 kHz
    Info : SWD DPIDR 0x2ba01477
    Info : lpc13xx.cpu: hardware has 6 breakpoints, 4 watchpoints
    Info : lpc13xx.cpu: external reset detected
    Info : Listening on port 3333 for gdb connections
    target halted due to debug-request, current mode: Thread
    xPSR: 0x01000000 pc: 0x1fff0104 msp: 0x10000ffc
    auto erase enabled
    Warn : Verification will fail since checksum in image
     (0x00000041) to be written to flash is different from
     calculated vector checksum (0xefffde78).
    Warn : To remove this warning modify build tools on developer PC to
     inject correct LPC vector checksum.
    wrote 8192 bytes from file ${JOB}/target/build/target.hex in
     37.157677s (0.215 KiB/s)
    shutdown command invoked
   | result  : success (exit status = 0)
  execute
   | cmd : ['make', '-C', 'target', '--no-builtin-rules', 'spotless']
    make[1]: Entering directory '${JOB}/target'
    make[1]: Leaving directory '${JOB}/target'
   | result  : success (exit status = 0)
 open  scope
  DriverVersion                 : PS2000A Linux Driver, 2.1.0.570
  USBVersion                    : 2.0
  HardwareVersion               : 1
  VariantInfo                   : 2206B
  BatchAndSerial                : FU837/0324
  CalDate                       : 01May18
  KernelVersion                 : 0.0
  DigitalHardwareVersion        : 1
  AnalogueHardwareVersion       : 1
  PicoFirmwareVersion1          : 1.3.3.0
  PicoFirmwareVersion2          : 1.0.55.0
 configure driver
  driver type        = block
  driver version     = 0.1.0
  driver sizeof( k ) = 16
  driver sizeof( r ) = 0
  driver sizeof( m ) = 16
  driver sizeof( c ) = 16
 calibrate scope
  before calibration, configuration = {
    "resolution": 8,
    "interval": 6.4e-8,
    "duration": 1
  }
  after  calibration, configuration = {
    "resolution": 8,
    "interval": 4e-9,
    "duration": 0.000332652
  }
run driver -> process
 started  acquiring trace  0 of 10
  measure via TSC    => 5259
  measure via signal => 83162
```

```
  crop wrt. +ve trigger edge @ 0
  crop wrt. -ve trigger edge @ 83162
 finished acquiring trace  0 of 10
 started  acquiring trace  1 of 10
  measure via TSC    => 5259
  measure via signal => 83162
  crop wrt. +ve trigger edge @ 0
  crop wrt. -ve trigger edge @ 83162
 finished acquiring trace  1 of 10
 ...
 ...
 ...
 started  acquiring trace  9 of 10
  measure via TSC    => 5259
  measure via signal => 83162
  crop wrt. +ve trigger edge @ 0
  crop wrt. -ve trigger edge @ 83162
 finished acquiring trace  9 of 10
 execute
  | cmd : ['gzip', '--quiet', './trace/00000000.pkl']
  | result  : success (exit status = 0)
 execute
  | cmd : ['gzip', '--quiet', './trace/00000006.pkl']
  | result  : success (exit status = 0)
 ...
 ...
 ...
 execute
  | cmd : ['gzip', '--quiet', './trace/00000008.pkl']
  | result  : success (exit status = 0)
run driver -> process_epilogue
 close board
 close scope
transfer local -> depo.
```

Appendix B

Sample Analysis Job Log

Below is a condensed analysis job log generated by a SCARV Lab analysis server with times and dates redacted.

```
Analysis job received with id:
 49862987210c8dac67b80965c69e76f75b8c498febfc58a083883721f25b17cf
Requesting acquisition job with id:
 d9a9ebb267411c483dc727521bc246e429e0664e2e637d954fe0f14d607a3a78
Downloading acquisition trace set...
Extracting acquisition trace set
Converting acquisition trace set
Extracting trs trace set
Performing attack: CONDITIONAL AVERAGE
attack stdout:
 Opened /tmp/d9a9ebb267411c483dc727521bc246e429e0664e2e637d954fe0f14d607a3a78
  /trace.trs, #traces 1000, #samples 83161 (Float32), #data 32, #title 255

 Jlsca running in Julia version: 1.1.0, 1 processes/1 workers/2 threads
  per worker

 DPA parameters
 attack:       AES Sbox CIPHER KL128 FORWARD
 mode:         CIPHER
 key length:   KL128
 direction:    FORWARD
 xor:          false
 analysis:     CPA
 leakages:     bit0,bit1,bit2,bit3,bit4,bit5,bit6,bit7
 maximization: abs global max
 combination:  +
 data at:      1

 phase: 1 / 1, #targets 16

 Attacking columns 1:83161 out of 83161 columns (run 1 out of 1)
 Running processor "Cond avg" on trace range 1:1:1000, 1 data passes,
  0 sample passes

 Averaged 1000 input traces into 16 averages, UInt8 data type,
  Float64 sample type
 CPA on samples shape (246, 83161) (range 1:83161) and data shape (246,)
 Results @ 246 rows, 83161 cols (1000 rows consumed)
 target: 1, phase: 1, #candidates 256
 rank:   1, candidate: 0x73, score: 2.828961 @ 3254
 rank:   2, candidate: 0x8a, score: 2.181422 @ 164
```

```
 rank:   3, candidate: 0x9b, score: 2.107425 @ 36743
 rank:   4, candidate: 0xb9, score: 2.107330 @ 82999
 rank:   5, candidate: 0xd9, score: 2.106527 @ 58226
 recovered key material: 73
 CPA on samples shape (249, 83161) (range 1:83161) and data shape (249,)
 Results @ 249 rows, 83161 cols (1000 rows consumed)
 target: 2, phase: 1, #candidates 256
 rank:   1, candidate: 0x32, score: 2.786605 @ 7218
 rank:   2, candidate: 0x7b, score: 2.113023 @ 21258
 rank:   3, candidate: 0xe6, score: 2.092334 @ 2711
 rank:   4, candidate: 0xa1, score: 2.081699 @ 17845
 rank:   5, candidate: 0x1b, score: 2.076363 @ 1027
 recovered key material: 32
 ...
 ...
 ...
 CPA on samples shape (252, 83161) (range 1:83161) and data shape (252,)
 Results @ 252 rows, 83161 cols (1000 rows consumed)
 target: 16, phase: 1, #candidates 256
 rank:   1, candidate: 0x5e, score: 2.800799 @ 5003
 rank:   2, candidate: 0x04, score: 2.173076 @ 70029
 rank:   3, candidate: 0xcb, score: 2.134654 @ 58414
 rank:   4, candidate: 0x43, score: 2.133063 @ 59731
 rank:   5, candidate: 0x92, score: 2.103170 @ 58658
 recovered key material: 5e
 recovered key: 7332d44101589f6542b0c4b305add75e
 236.781731 seconds (38.91 M allocations: 50.712 GiB, 9.55% gc time)

Purging acquisition job
```