Dissertation Type: software development


DEPARTMENT OF COMPUTER SCIENCE

# Building a Testbed for Evaluating Privacy Enhancing Technologies (PETs)

Jacob Daniel Halsey

---

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

---

Friday 14th May, 2021

# Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of BSc 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.

Jacob Daniel Halsey, Friday 14$^{th}$ May, 2021

# Contents

# Executive Summary

The goal of this project is to produce a simple and lightweight testbed platform for evaluating privacy enhancing technologies. It should provide support for testing varied architectures and network topologies, such as client-server and peer-to-peer applications. It should also support simulating applications for different types of platforms including mobile phone apps.

Summary of work:

- I have developed a flexible command line tool called *kvm-compose* for Linux using the Rust language and *libvirt* library for building and destroying virtual testing environments.
- In the process I have made some contributions to the *libvirt-rust* language bindings open source library.
- I have then implemented some example projects using the testbed tool.

# Supporting Technologies

- *Linux KVM* (Kernel-based Virtual Machine) - https://www.linux-kvm.org/
- *Open vSwitch* Virtual multilayer switch - https://www.openvswitch.org/
- *libvirt* Virtualization API - https://libvirt.org/
- *Rust* Language, Compiler, Toolchain, etc. - https://www.rust-lang.org/
- *libvirt-rust* Rust bindings to the libvirt - https://gitlab.com/libvirt/libvirt-rust
- *clap* Rust command Line Argument Parser - https://github.com/clap-rs/clap
- *serde* Rust Serialization framework - https://github.com/serde-rs/
- *serde-yaml* YAML backend for serde - https://github.com/dtolnay/serde-yaml
- *serde-plain* Plain text backend for serde - https://github.com/mitsuhiko/serde-plain
- *thiserror* Rust error derive macro - https://github.com/dtolnay/thiserror
- *anyhow* Rust error handling framework - https://github.com/dtolnay/anyhow
- *simple_logger* Rust logging implementation - https://github.com/borntyping/rust-simple_logger
- *xml-rs* XML library for Rust - https://github.com/netvl/xml-rs
- *validator* Rust struct validation - https://github.com/Keats/validator
- *directories* User data directories library - https://github.com/dirs-dev/directories-rs
- *reqwest* Rust HTTP Client - https://github.com/seanmonstar/reqwest
- *indicatif* Rust command line progress indicator - https://github.com/mitsuhiko/indicatif
- *tempfile* Rust temporary file library - https://github.com/Stebalien/tempfile
- *casual* Rust user input parser - https://github.com/rossmacarthur/casual
- *derive-new* Rust new constructor macro - https://github.com/nrc/derive-new
- *enum-iterator* Rust macro for iterating enums - https://github.com/stephaneyfx/enum-iterator
- *rust-embed* Embeds files into Rust binaries - https://github.com/pyros2097/rust-embed

# Acknowledgements

I would like to thank my supervisor Professor Awais Rashid and co-supervisor Joe Gardiner for their project proposal and support and guidance in completing it.

# Chapter 1

# Contextual Background

The UK Research and Innovation (UKRI) is a non-departmental public body of the United Kingdom Government sponsored by the Department for Business, Energy and Industrial Strategy [1]. In October 2020 the UKRI announced the creation of the National Research Centre on Privacy, Harm Reduction and Adversarial Influence Online (REPHRAIN) [2]. The centre is made up of researchers in computer science, international relations, law, psychology, management, design, digital humanities, public policy, political Science, criminology, and sociology from five British universities including the University of Bristol.

REPHRAIN should be understood in the context of the UK government's *Online Harms White Paper* public consultation beginning in April 2019 [3], which sets out plans for new online safety measures; REPHRAIN's missions and outcomes are aligned with this paper [4].

REPHRAIN will focus on three core missions [5]:

1. Delivering privacy at scale while mitigating its misuse to inflict harms.
2. Minimising harms while maximising benefits from a sharing-driven digital economy.
3. Balancing individual agency vs. the social good.

The three missions will require looking at Privacy Enhancing Technologies (PETs); including their capabilities, applications of PETs in addressing existing online harms, mitigating the potential abuse of PETs, embedding the PETs into infrastructures, and developing new PETs. In order to facilitate this REPHRAIN intends to build a toolbox of resources including a PETs testbed. The testbed will be used researchers in developing, testing, and evaluating the PETs. The aim of this project is to develop a prototype for this testbed.

## 1.1 What are Privacy Enhancing Technologies (PETs)?

Before we discuss Privacy Enhancing Technologies we must consider what we mean by "privacy". REPHRAIN is primarily using the definitions set out by D. J. Solove in his 2006 article *A Taxonomy of Privacy* [6, 4]. Solove notes that the definition of privacy has often been very

1. Information collection
   (a) Surveillance
   (b) Interrogation
2. Information processing
   (a) Aggregation
   (b) Identification
   (c) Insecurity
   (d) Secondary Use
   (e) Exclusion
3. Information dissemination
   (a) Breach of Confidentiality
   (b) Disclosure
   (c) Exposure
   (d) Increased Accessibility
   (e) Blackmail
   (f) Appropriation
   (g) Distortion
4. Invasion
   (a) Intrusion
   (b) Decisional Interference

Figure 1.1: A Taxonomy of Privacy Violations [6].

broad or vague, and therefore sets out to develop a taxonomy of privacy violations. He has defined four groups of harmful activities (See figure 1.1).

Broadly speaking a Privacy Enhancing Technology is any solution or approach in hardware or software that helps protect a user from such privacy violations [7]. Some examples of PETs could include Onion routing such as the Tor network (which enables anonymous communication), or end-to-end encrypted messaging systems such as the Signal protocol. Kaaniche et al. [8] have defined a more comprehensive classification of PETs (see Figure 1.2).

## 1.2 Existing Solutions

There has been some existing research by Tekeoglu and Tosun [9] who have developed a privacy testbed for Internet-of-Things (IoT) devices. Their approach has some similar goals to this project in that it looks at capturing layer 2 and 3 network traffic. They note that the testbed enables experiments such as port vulnerability scans, checking what cipher suites are used (or not), and generally monitoring network traffic to see what data is being collected. However their testbed is different in that it is only designed for IoT devices; rather than general purpose PET applications.

## 1.3 High Level Objectives

Overall the high-level objective of this project is to develop a simple and lightweight testbed platform for evaluating PETs:

- The testbed should support testing various architectures and network topologies, including client/server and peer-to-peer applications, to accommodate a variety of PETs.

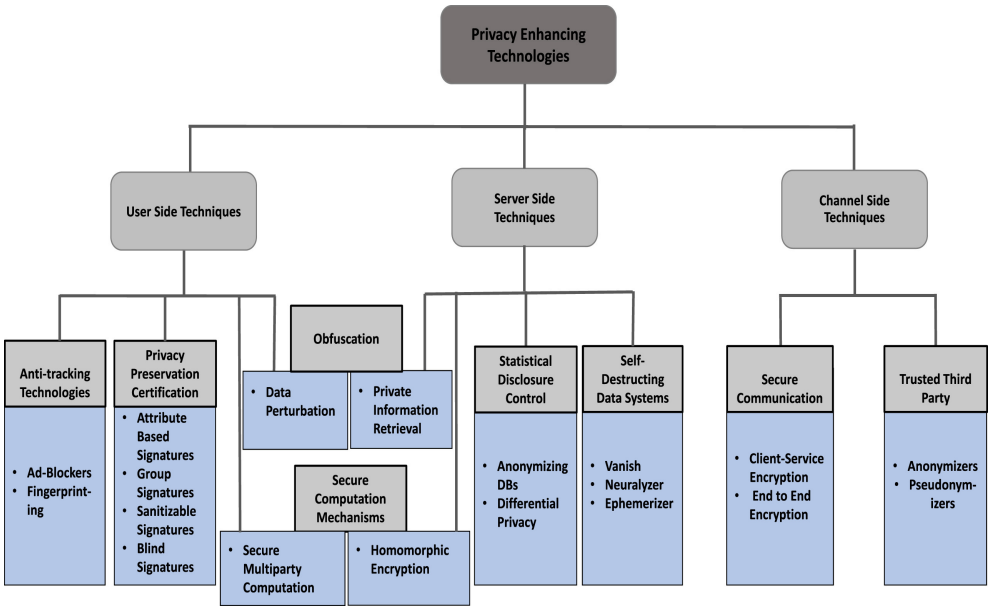


Figure 1.2: A Taxonomy of privacy enhancing technologies [8].

- The testbed must be able to collect information such as packet captures for use in evaluating the privacy properties.
- The testbed should support different platforms such as desktop and mobile apps, and both applications where the source code is available or only pre-built binaries.
- The testbed should enable a high level of automation, such that working with large test environments because feasible, and the setup can easily and programmatically be replicated.

# Chapter 2

# Technical Background

In this chapter I will discuss some of the technologies which this project depends or builds upon.

## 2.1 Virtualization

> 'Virtualization uses software to create an abstraction layer over computer hardware that allows the hardware elements of a single computer—processors, memory, storage and more—to be divided into multiple virtual computers, commonly called virtual machines (VMs). Each VM runs its own operating system (OS) and behaves like an independent computer, even though it is running on just a portion of the actual underlying computer hardware.' [10]

Virtualization[1] will therefore be a very useful technology for the testbed, since it will allow us to model an environment consisting of multiple computers such as application clients and servers, and run them all within a single machine. In addition, modern CPU extensions (such as Intel VT and AMD-V) provide hardware-assisted / accelerated virtualization support, allowing the virtual machines to have near native performance which will help in meeting the goal of minimal overhead for the testbed.

In order to use virtualization a *Hypervisor* is required, this is the software layer sits between the physical hardware and manages the virtual machines. A hypervisor may run directly on the physical machine in place of a conventional operating system - as a Type 1 or *bare-metal* hypervisor, or run within a separate host operating system - as a Type 2 or *hosted* hypervisor.

### 2.1.1 KVM

For the prototype testbed I will be using *Kernel-based Virtual Machine* (KVM), which is a kernel module for the Linux operating system that allows it to function as a hypervisor. The

[1] I am a British citizen, and this is a dissertation at a British university, and therefore I am very much aware that the correct spelling in British English is 'virtualisation', however the majority of platforms, libraries, and sources I will be referencing use the US English spelling, so I have chosen to do the same.

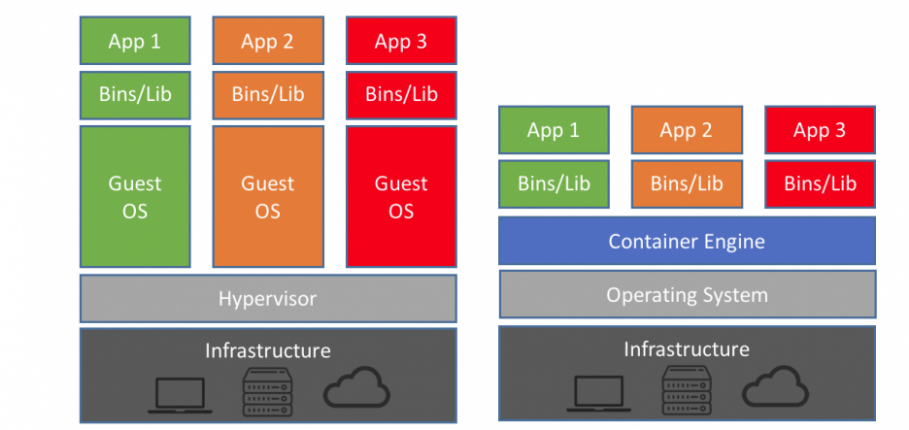


Figure 2.1: Platform Virtualization vs Containerization[3]

advantage is that KVM (and the Linux kernel itself) are free and open-source software under GNU licenses, and it is a stable and mature platform [11]. In userspace QEMU[2] may then use KVM to provide a full virtualization platform.

*libvirt* is an open-source toolkit for managing virtualization platforms [12], it supports QEMU/KVM as well as hypervisors from other vendors. It is written in C, with bindings available in many other programming languages, making it a suitable library for developing the testbed. Support for other platforms also means it would be easier to add support for additional hypervisors in future.

## 2.2 Containerization

Having discussed full platform / machine virtualization it is worth also mentioning containerization which is an alternative lightweight approach to virtualizing applications. In particular there is the Docker platform for Linux containers which has become very popular in recent years [13].

Unlike full platform virtualization, containerization does not virtualize a whole computer (or require hardware acceleration), instead it uses namespacing within the host operating system kernel to create an isolated environment. This has many practical uses, making deploying software and services very quick and easy, but unfortunately it is not ideal for our testbed, since it would mean all applications would have to use the same operating system; it couldn't be used to simulate different platforms.

While I will not be using containerization, the *Docker Compose* tool [14] used for orchestrating containers has provided some useful inspiration for the testbed. *Docker Compose* is a command line program with two core subcommands *up* and *down* which are used to either build or destroy a set of containers as defined in a YAML configuration file. The configuration file

[2]Note QEMU can also function as its own independent type 2 hypervisor but KVM is required to enable hardware acceleration, see https://www.packetflow.co.uk/what-is-the-difference-between-qemu-and-kvm/
[3]Graphic from https://blog.netapp.com/blogs/containers-vs-vms/

may define a list of containers, each with options including an image to download, an entry command to run, volumes to attach, and environment variables, the config may also define virtual networks and attach them to the containers. These are all very useful features in line with the goals for the testbed, and as such I will try to replicate them but within a fully virtualized environment (QEMU/KVM).

## 2.3 Virtual Networks

### 2.3.1 Software Defined Networking

## 2.4 The Rust Language

As you will see in Chapter 3, I have chosen to use the Rust programming language for developing the testbed. Although there are no doubt many languages which could have been used, I will provide some background on Rust, and its advantages for this project.

Rust is a modern systems programming language, originally developed by Mozilla, with its first stable release in 2015. It is designed with a focus on performance, safety and concurrency.

It has some key advantages:

- Excellent performance; on par with C/C++[4].
- Easy interaction with C libraries via FFI, making the *libvirt* [15] bindings possible.
- A simple to use package manager - *Cargo*, along with a rich ecosystem of libraries[5].
- Modern functional constructs such as sum types and pattern matching.
- Memory and thread safety are enforced at compile time [16].
- The compiler and standard library support a large number of platforms.

## 2.5 Cloud Init

[4] https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html
[5] https://crates.io/

# Chapter 3

# Project Execution

## 3.1 Modifications to *libvirt-rust*

As mentioned in section 2.1.1 I was going need to use the *libvirt* library for interacting with QEMU/KVM, as well as the bindings to access it from the Rust language: *libvirt-rust* [15]. During my initial experiments developing the project I discovered that under certain build setups / toolchains the *libvirt-rust* crate[1] failed to compile, due to it containing function declarations that did not actually exist in the *libvirt* library. This is described in full detail in my issue report: https://gitlab.com/libvirt/libvirt-rust/-/issues/1.

I submitted a fixed version of *libvirt-rust* with these invalid functions removed, but also with a test case that builds the bindings in a static library, therefore checking that all the symbols (such as functions) did actually exist in the underlying *libvirt* library. The tests are automatically run as part of the CI/CD pipeline so this sort of problem should be prevented in future. The merge request was approved by the project maintainers: https://gitlab.com/libvirt/libvirt-rust/-/merge_requests/14.

Whilst working with *libirt* I also discovered that I was receiving duplicate error messages printed to the console (via *stdout/stderr*), this was because by default *libvirt* has an error handler configured to print all errors, but at the same time the *libvirt-rust* binding functions also returned a `Result<_, virt::error::Error>` type (the idiomatic approach in Rust), which on failure I was also printing to the console via the logging framework. So I also added a `clear_error_func()` that binds to `virSetErrorFunc` along with my changes, so that the default error handler can be disabled.

[1] Rust packages such as libraries, managed through the *Cargo* package manager are known as 'Crates'

## 3.2 *kvm-compose*

## 3.3 Examples

## 3.4 Development Practice

During development I used the *Git* version control system, with free hosting from *GitHub Inc.* I also made use of the *GitHub Actions* CI/CD platform, with a workflow configured to check that the project builds and passes style checks on each push to the repository.

# Chapter 4

# Critical Evaluation

**A topic-specific chapter, of roughly 15 pages**

This chapter is intended to evaluate what you did. The content is highly topic-specific, but for many projects will have flavours of the following:

1. functional testing, including analysis and explanation of failure cases,
2. behavioural testing, often including analysis of any results that draw some form of conclusion wrt. the aims and objectives, and
3. evaluation of options and decisions within the project, and/or a comparison with alternatives.

This chapter often acts to differentiate project quality: even if the work completed is of a high technical quality, critical yet objective evaluation and comparison of the outcomes is crucial. In essence, the reader wants to learn something, so the worst examples amount to simple statements of fact (e.g., "graph X shows the result is Y"); the best examples are analytical and exploratory (e.g., "graph X shows the result is Y, which means Z; this contradicts [1], which may be because I use a different assumption"). As such, both positive *and* negative outcomes are valid *if* presented in a suitable manner.

# Chapter 5

# Conclusion

**A compulsory chapter, of roughly 5 pages**

The concluding chapter of a dissertation is often underutilised because it is too often left too close to the deadline: it is important to allocation enough attention. Ideally, the chapter will consist of three parts:

1. (Re)summarise the main contributions and achievements, in essence summing up the content.

2. Clearly state the current project status (e.g., "X is working, Y is not") and evaluate what has been achieved with respect to the initial aims and objectives (e.g., "I completed aim X outlined previously, the evidence for this is within Chapter Y"). There is no problem including aims which were not completed, but it is important to evaluate and/or justify why this is the case.

3. Outline any open problems or future plans. Rather than treat this only as an exercise in what you *could* have done given more time, try to focus on any unexplored options or interesting outcomes (e.g., "my experiment for X gave counter-intuitive results, this could be because Y and would form an interesting area for further study" or "users found feature Z of my software difficult to use, which is obvious in hindsight but not during at design stage; to resolve this, I could clearly apply the technique of Smith [7]").

# Bibliography


[1] UKRI, *Who we are*, Nov. 2020. [Online]. Available: https://www.ukri.org/about-us/who-we-are/.

[2] ——, *New centre launched to keep citizens safe online*, Oct. 2020. [Online]. Available: https://www.ukri.org/news/new-centre-launched-to-keep-citizens-safe-online/.

[3] Department for Digital, Culture, Media and Sport, *Online harms white paper*, Dec. 2020. [Online]. Available: https://www.gov.uk/government/consultations/online-harms-white-paper.

[4] REPHRAIN, *Online harms*. [Online]. Available: https://www.rephrain.ac.uk/online-harms/.

[5] ——, *Missions*. [Online]. Available: https://www.rephrain.ac.uk/missions/.

[6] D. J. Solove, "A taxonomy of privacy," *University of Pennsylvania Law Review*, vol. 154, no. 3, pp. 477–564, 2006. [Online]. Available: https://www.law.upenn.edu/journals/lawreview/articles/volume154/issue3/Solove154U.Pa.L.Rev.477%282006%29.pdf.

[7] D. Buckley, *Privacy enhancing technologies for trustworthy use of data*, Feb. 2021. [Online]. Available: https://cdei.blog.gov.uk/2021/02/09/privacy-enhancing-technologies-for-trustworthy-use-of-data/.

[8] N. Kaaniche, M. Laurent, and S. Belguith, "Privacy enhancing technologies for solving the privacy-personalization paradox: Taxonomy and survey," *Journal of Network and Computer Applications*, vol. 171, p. 102 807, 2020, ISSN: 1084-8045. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804520302794.

[9] A. Tekeoglu and A. S. Tosun, "A testbed for security and privacy analysis of iot devices," in *2016 IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, IEEE, 2016, pp. 343–348. [Online]. Available: https://ieeexplore.ieee.org/document/7815045.

[10] IBM Cloud Education, *Virtualization: A complete guide*, Jun. 2019. [Online]. Available: https://www.ibm.com/cloud/learn/virtualization-a-complete-guide.

[11] Red Hat, *Kvm vs. vmware*. [Online]. Available: https://www.redhat.com/en/topics/virtualization/kvm-vs-vmware-comparison.

[12] libvirt, *The virtualization api*. [Online]. Available: https://libvirt.org/index.html.

[13] S. J. Vaughan-Nichols, *What is docker and why is it so darn popular?* Mar. 2018. [Online]. Available: https://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/.

[14] Docker Inc, *Overview of docker compose*, 2021. [Online]. Available: https://docs.docker.com/compose/.

[15] libvirt, *Libvirt-rust*. [Online]. Available: https://gitlab.com/libvirt/libvirt-rust.

[16] A. Balasubramanian, M. S. Baranowski, A. Burtsev, A. Panda, Z. Rakamarić, and L. Ryzhyk, "System programming in rust: Beyond safety," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, ser. HotOS '17, Whistler, BC, Canada: Association for Computing Machinery, 2017, pp. 156–161, ISBN: 9781450350686. DOI: 10.1145/3102980.3103006. [Online]. Available: https://doi.org/10.1145/3102980.3103006.