Hovedoppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for elektroniske systemer

Armin Sukurica, H. D. Vincent Sun, Modestas Volkovas

Securing data integrity: Raspberry Pi-based anti-tampering system with Zymbit Protokit and Zymkey module

Hovedoppgave i BIELEKTRO, BDIGSEC
Veileder: Arild Moldsvor, Arvind Sharma
Mai 2024

NTNU
Kunnskap for en bedre verden

Armin Sukurica, H. D. Vincent Sun, Modestas Volkovas

Securing data integrity: Raspberry Pi-based anti-tampering system with Zymbit Protokit and Zymkey module

Hovedoppgave i BIELEKTRO, BDIGSEC
Veileder: Arild Moldsvor, Arvind Sharma
Mai 2024

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for elektroniske systemer

NTNU

Kunnskap for en bedre verden

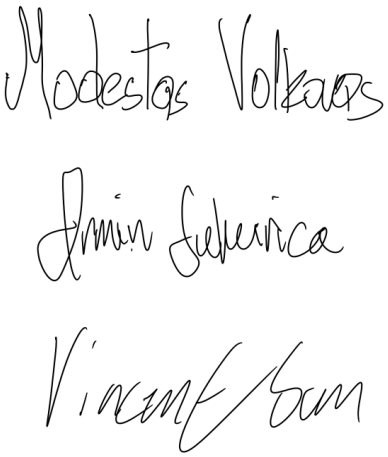
Preface

This thesis, part of our bachelor's degree at NTNU - Norwegian University of Science and Technology in Gjøvik, Norway, is a collaborative effort between the Electrical Engineering program, specializing in electronics and sensor systems, and the Digital Infrastructure and Cyber Security program.

Commissioned by Thales Norway, it focuses on developing tamper protection solutions for an anti-tamper enclosure, exploring both hardware and software strategies to secure sensitive data against potential intruders.

Our heartfelt gratitude goes to Roar Gjærevold and Anders Paulshus for their invaluable feedback, guidance, and support throughout the project, which has been a source of motivation and encouragement.

We would also like to extend our appreciation to our university supervisors, Arild Moldsvor and Arvind Sharma, for their assistance with academic writing and advice throughout this project.



Abstract

Highly sensitive information stored on a device, can be a valuable target for hackers or a foreign state. To prevent attackers from gaining access to this information, it is crucial to secure the device.
This thesis proposes a solution to protect data using a Raspberry Pi and a Zymbit Protokit with Zymkey. Additionally it delves into important aspects of information security and the practical methods of safeguarding data. To get a deeper understanding of how these security-measures work, a tamper-resilient device was created. This device is meant as a solution to safeguard the exchange and storage of data between two Raspberry Pis. The findings showcase that an anti-tamper solution with approachable products can be used to create a secure device.

Sammendrag

Høysensitiv informasjon lagret på en enhet kan være et verdifullt mål for hackere eller en fremmed stat. For å forhindre at angripere får tilgang til denne informasjonen, er det avgjørende å sikre enheten.
Denne oppgaven foreslår en løsning for å beskytte data ved bruk av Raspberry Pi og Zymbit Protokit med Zymkey. I tillegg går den inn på viktige aspekter ved informasjonssikkerhet og de praktiske metodene for å beskytte data. For å få en dypere forståelse av hvordan disse sikkerhetstiltakene fungerer, ble en manipulasjonssikker enhet opprettet. Denne enheten er ment som en løsning for å sikre utveksling og lagring av data mellom to Raspberry Pi maskiner. Funnene viser at en anti-manipulasjonsløsning med tilgjengelige produkter kan brukes til å lage en sikker enhet.

Contents

Preface . . . . . **iii**
Abstract . . . . . **v**
Sammendrag . . . . . **vii**
Contents . . . . . **ix**
Figures . . . . . **xiii**
Tables . . . . . **xv**
Code Listings . . . . . **xvii**
1 Introduction **1**
1.1 Background 1
1.2 Project scope 2
1.3 Goals 2
1.4 Thesis structure 3
2 Theory **5**
2.1 Introduction to anti-tamper technologies 5
2.2 Parameters Conditioning Attacks 5
2.2.1 Macroscopic scale 6
2.2.2 Micro-technology 6
2.2.3 Nano-technology 6
2.3 Attack potential to a security box 6
2.3.1 Expertise 6
2.3.2 Equipment and tools used in a potential attack 7
2.4 Cold boot attacks 8
2.5 Communication 9
2.5.1 Inter-Integrated Circuit (I^2C) 9
2.5.2 False positive & false negative 10
2.6 Hardware 10
2.6.1 Zymkey4 10
2.6.2 Raspberry Pi 10
2.7 Cryptography 10
2.7.1 Encryption 11
2.7.2 Symmetric key encryption 11
2.7.3 Asymmetric key encryption 11
2.7.4 Data-in-transit vs. Data-at-rest security 12
2.7.5 Algorithms 13

2.8 OSI model . . . 13
2.8.1 OSI model and encryption . . . 14
2.9 IPsec (Internet Protocol Security) . . . 15
2.9.1 The benefits of using IPsec . . . 15
2.9.2 IPsec modes . . . 15
2.9.3 IPsec protocols . . . 15
2.9.4 IPsec workflow . . . 16
2.10 Attack potential to a communication channel . . . 17
3 Execution of the project . . . 19
3.1 ProtoKit5 . . . 19
3.2 Flex PCB . . . 20
3.2.1 Measurement . . . 21
3.2.2 Design with KiCad . . . 21
3.2.3 Testing the size . . . 21
3.2.4 After receiving from the manufacturer . . . 22
3.3 Sensors for tampering detection . . . 23
3.3.1 Temperature and humidity . . . 23
3.3.2 Accelerometer and gyroscope . . . 25
3.3.3 Proximity and lux . . . 27
3.3.4 Short term variation and noise of sensors . . . 29
3.3.5 Assembling to the Protoboard . . . 29
3.4 Technical Design . . . 30
3.4.1 Prerequisites - Raspberry Pi . . . 30
3.4.2 Prerequisites - Zymkey . . . 31
3.5 IPsec configuration . . . 31
3.5.1 Strongswan . . . 32
3.5.2 IPsec authentication . . . 32
3.5.3 IPsec parameter explanation . . . 32
3.5.4 Server-side IPsec configuration . . . 33
3.5.5 Client-side IPsec configuration . . . 33
3.6 Scripts . . . 34
3.6.1 Server-side script . . . 34
3.6.2 Client-side script . . . 36
3.6.3 Decryption script . . . 37
3.6.4 Encryption script . . . 38
3.6.5 VPN monitoring using Zymkey4 . . . 38
3.6.6 Zymkey monitoring script . . . 40
4 Results . . . 43
4.1 Flex PCB . . . 43
4.2 Protoboard & sensors . . . 44
4.3 Sensor trigger levels . . . 45
4.4 Circumventing each sensor . . . 46
4.5 Secure communication . . . 46
4.5.1 Communication work flow . . . 47

4.5.2 Secure storage of data 48
4.5.3 Server-IPsec dependancy 48
4.5.4 IPsec-Zymkey dependancy 49
4.5.5 Zymkey - Sensor dependancy 49
5 Discussion & conclusion 51
5.1 Hardware discussion 51
5.2 Software discussion 52
5.3 Conclusion 53
Bibliography 55

Figures

2.1 I^2C data transfer sequence, [3]. 9
2.2 I^2C address on a sensor's silkscreen, [5]. 9
2.3 Symmetric key encryption, [14]. 11
2.4 Asymmetric key encryption, [16]. 12
2.5 OSI Model, [20]. 14
2.6 IPsec workflow 16

3.1 Components in the ProtoKit5, [30]. 20
3.2 Actuator pins pressing the switches, [30]. 20
3.3 Illustration of how flex PCBs will be connected. 21
3.4 "Paper PCBs" to verify correct measurements. 22
3.5 Solder pad torn off from flex PCB. 23
3.6 Quick fix to address the weakness. 23
3.7 Temperature and humidity sensor used, [32]. 24
3.8 Accelerometer and gyroscope sensor used, [34]. 25
3.9 Proximity and lux sensor used, [5]. 28
3.10 Areas available for sensor integration. 30
3.11 Raspberry Pi - OS Update. 31

4.1 The PCB design for the lid. 43
4.2 The PCB design for the box. 44
4.3 Holes in the enclosure for mounting protoboard and sealing the enclosure. 44
4.4 Components soldered to the board. 45
4.5 IPSec VPN tunnel status. 47
4.6 Client Raspberry Pi connecting to the server. 47
4.7 Server Raspberry Pi receiving messages from the client. 47
4.8 Encrypted message stored on the server Raspberry Pi. 48
4.9 Unlocking of encrypted data. 48
4.10 Decrypted version of the message stored on the server Raspberry Pi. 48
4.11 Check for a IPsec tunnel that allows the server to be started. 49
4.12 Continuous IPsec tunnel monitoring with automatic server shutdown if unavailable. 49
4.13 Continuous breach monitoring that brings down the IPsec tunnel. . 49

4.14 Continuous sensor monitoring with automatic Zymkey deletion. . . 50

Tables

2.1 Rating for simple tools, [1]. 7
2.2 Rating for specialized tools [1] 8
2.3 Rating for advanced tools, [1]. 8
2.4 Tools and techniques. 17

3.1 Temperature and humidity readings over a four-hour period. 25
3.2 Acceleration readings in the X, Y, and Z axes (m/s^2) over four-hour period. 26
3.3 Rotation measurement on the X, Y, and Z axes in degrees/sec over a four-hour period. 27
3.4 Proximity and lux readings over a four-hour period. 29
3.5 Acceleration readings in the X, Y, and Z axes (m/s^2) over a 10-second period. 29
3.6 IPsec counter-measures against various types of attacks. 32

4.1 Trigger levels for sensors. 45

Code Listings

3.1 Python script for interval-based temperature and humidity monitoring. 24
3.2 Python script for interval-based monitoring of the acceleration and rotation. 26
3.3 Python script for interval-based light and proximity monitoring. . . 28
3.4 IPsec configuration for the server. 33
3.5 IPsec configuration for the client. 33
3.6 Client script for connecting to the server. 34
3.7 Client script for connecting to the server. 36
3.8 Decryption script for unlocking the "ReceivedMessages" folder. . . . 37
3.9 Encryption script for encrypting the "ReceivedMessages" folder. . . . 38
3.10 Zymkey monitoring for the VPN. 39
3.11 Zymkey self detruct script based on sensor values. 40

Chapter 1

Introduction

1.1 Background

The project group has for this thesis, in response to a project assignment proposed by Thales Norway AS (henceforth called “external client”), developed an anti-tamper device. The main goal of the device is to protect sensitive data by deleting the encryption key. The device was made using a ProtoKit5, Zymkey4 and Raspberry Pi, which are all publicly available hardware components.

Thales Norway AS is a subsidiary of Thales Group, a technology and solutions provider in the markets of defence and security, aerospace and space, digital identity and security, and transport. With more than 100 years of presence in Norway, Thales Norway AS have today become a world leader in crypto- and high-grade secured communications. For this project, Thales Norway AS have supplied the project group with relevant equipment for the task.

The ProtoKit5 is a development kit consisting of several parts, such as an IP67 rated enclosure and protoboard. It is created by Zymbit, a company specializing in hardware security and IoT applications. The development kit is intended to be used to protect a Raspberry Pi. It also comes with a basic anti-tamper protection, using tamper switches. With some modification this protection can be increased significantly by adding different types of detection sensors.

The anti-tampering mechanism will work by deleting the encryption key that is stored on the Zymkey, in response to any unauthorized attempt to access the device. The deletion of the encryption key, ensures that the data on the Raspberry Pi becomes unreadable to the human eye. Additionally, the memory area will be overwritten, ensuring that the key cannot be restored. These measures will ensure that that the data stays safe even if an intruder gets a hold of the device, seeing as a functional Zymkey is needed to decrypt the data.

1.2 Project scope

While there are various ways of implementing an anti-tampering solution, it is important to recognize that fail-proof security is unattainable, especially within the span of one semester. To ensure that the development process runs smoothly and to prevent the scope from becoming overly complex the project group have set the following constraints that will be adhered to:

1. The group will use the Raspberry Pi and the ProtoKit5, handed by the external clients. These hardware components will form the backbone for the project.
2. The group will focus protecting the data at rest and in transit between two Raspberry Pis. This will involve implementing robust encryption algorithms and secure communication channels to safeguard sensitive information from potential threats.
3. The group will not be conducting tests for multiple existing attack vectors, due to not having the advanced and specialized equipment needed to perform such experiments.
4. The communication channel between the two Raspberry Pis will be set up with a home-network in mind. This is to say that the setup is not meant to be used in a larger scale infrastructure.
5. To enhance security, the group will utilize the Zymkey for storing encryption keys. This hardware security module will provide a secure environment for key storage and ensure that the key is protected from unauthorized access and tampering.
6. The normal operating conditions for the Raspberry Pis will be set between 0 - 50°C and assume a stationary environment.

1.3 Goals

The goal of this project is to take a deep dive into the world of anti-tampering security solutions and to design a functional prototype of an anti-tamper enclosure. The overarching goal of the prototype is to make it difficult for an intruder to access the plain-text data stored on the Raspberry Pi, thereby protecting the information from being compromised. To achieve this, multiple layers of security measures will be implemented. Additionally, a secure communication channel between two Raspberry Pis will be established to allow for seamless data transmission between them. By engaging in this activity, the group will acquire practical, hands-on experience in product development as well and cultivate a deeper understanding of the principles and methods used in safeguarding data.

1.4 Thesis structure

The clarification of the upcoming chapters in the thesis is listed below.

Chapter 2 - Theory This chapter describes the necessary knowledge required to understand how anti-tampering works in both hardware and software. Additionally, it will explore other relevant information in detail.

Chapter 3 - Execution of the project This chapter describes the methods that were used to accomplish the project goals and the implementation of the chosen security measures.

Chapter 4 - Results This chapter presents the results garnered from the project execution, demonstrating how the different security measures function.

Chapter 5 - Discussion & Conclusion This chapter discusses the results in further detail, examining the strengths and weaknesses of the prototype and potential future work.

Chapter 2

Theory

2.1 Introduction to anti-tamper technologies

Anti-tamper technologies are critical in ensuring the security and integrity of various devices and systems. From safeguarding ATMs and securing sealed food packages to preventing car theft and protecting hardware security modules and trusted platform modules in computers, these technologies play a vital role. The primary objective of anti-tamper measures is to detect unauthorized access or tampering attempts while avoiding false alarms that could disrupt normal operations.

The primary challenge in designing effective anti-tamper systems is to achieve a balance between sensitivity and robustness. Anti-tamper systems must detect all genuine attacks without being overly sensitive to non-threatening events that could trigger false alarms.

Several frameworks and standards guide the development and implementation of anti-tamper technologies. Notable among them are the SOG-IS (Senior Officials Group Information Systems Security) and the NIST (National Institute of Standards and Technology) FIPS (Federal Information Processing Standards) 140-2/3. These frameworks provide guidelines and specifications for evaluating and certifying the security of cryptographic modules and other security-critical components.

2.2 Parameters Conditioning Attacks

Parameters conditioning attacks are the variables, conditions, or factors that determine the success, feasibility, or complexity of an attack in a given security context. These parameters may include characteristics of the target system, such as its architecture, configuration, or vulnerabilities, as well as external factors like network conditions, attacker capabilities, and available resources. This framework for categorizing attacks by scale closely follows the discussion in, [1, page 7].

2.2.1 Macroscopic scale

According to [1], attacks at this scale "are directed at entire devices, including the complete external enclosure which often contains several components, such as PCB boards, batteries, etc." The primary goal of such an attack is to gain access to the internal components within the enclosure.

2.2.2 Micro-technology

This category, as described in [1], includes attacks targeted at "assembled electronic components, such as PCB boards that contain buses and integrated circuits (ICs)." The focus here is on "disrupting the buses that transmit data between components or targeting the connectors of the ICs themselves" [1].

2.2.3 Nano-technology

Following the analysis in [1], attacks at the nano-technology scale involve "the internal workings of ICs." Conducting these attacks "requires highly precise and specialized tools." The objectives may include "altering the behavior of the ICs or retrieving data stored within them" [1].

2.3 Attack potential to a security box

Understanding the potential attacks a security box may face is crucial for designing effective countermeasures. Below, various aspects of attack potential are discussed, including the expertise levels of attackers, the tools employed, and other relevant factors.

2.3.1 Expertise

Expertise refers to the skill or knowledge someone has within a particular field or product type. In the context of the anti-tamper enclosure, three types of experts are defined:

Novice Someone that has limited system knowledge and limited access to equipment and tools.

Skilled Someone that has specialized knowledge, experience and education. They are familiar with the security behaviour of the product or have some sort of expertise in attack techniques, and or handling of specific machines/tools that can target the enclosure.

Experts Someone with deep knowledge and extensive experience with machines or tools. They have a understanding of the hardware architecture of the enclosure, and the underlying security measures implemented within the system.

2.3.2 Equipment and tools used in a potential attack

The tables 2.1-2.3 have been put together by a group of industry experts, [1]. The tables show the different kinds of equipment that can be used in a potential attack, categorized by their price and availability.

Standard equipment refers to items easily accessible to an attacker for identifying vulnerabilities or launching attacks, which can be obtained online or from local stores.

Table 2.1: Rating for simple tools, [1].

Tool	Equipment
Heat guns	Standard
Glue	Standard
Soldering Iron	Standard
Solder paste	Standard
Chemical products	Standard
Syringe	Standard
Screwdriver	Standard
Needle	Standard
Knife	Standard
Torch	Standard
Standard drill	Standard
Drill press	Standard
Radial arm saw	Standard
Steel cutting blades	Standard
Circular saw	Standard
Hammer	Standard
Mirrors	Standard
Antennas	Standard
Microphones	Standard
Micro-cameras	Standard
Shunts	Standard
Wires and electrical probes	Standard
Borescope	Standard
Fiberscope	Standard
Signal analysis software	Standard
PC or workstation	Standard
Analogical Oscilloscope	Standard
Multimeter	Standard

Specialized equipment, often less accessible due to its cost or size, can still be acquired with some effort and is typically used only by skilled individuals and experts.

Table 2.2: Rating for specialized tools [1]

Tool	Equipment
Milling Machine	Specialized
CNC Milling Machine	Specialized
Laser Milling Machine	Specialized
Laser Equipment	Specialized
Sandblasting Machine	Specialized
Electromagnetic emitting devices	Specialized
Electrostatic emitting devices	Specialized
Signal and function processor	Specialized
Signal/Protocol Analyser	Specialized
Digital Oscilloscope	Specialized
Conductive ink printer	Specialized
Tools for chemical etching	Specialized
Tools for grinding	Specialized
Anechoic chamber	Specialized
Climate chamber	Specialized
Standard X-ray machine	Specialized
Gamma-ray generator	Specialized
Radio-frequency generator	Specialized
Standard thermal camera	Specialized
Standard tomography scanner	Specialized
FIB systems	Specialized

Bespoke equipment, which is either highly sophisticated or subject to strict control, is generally expensive and inaccessible to the public, thus it is primarily used by industry experts.

Table 2.3: Rating for advanced tools, [1].

Tool	Equipment
New Tech Design Verification and Failure Analysis Tools	Bespoke
X-ray 3-D tomograph	Bespoke

2.4 Cold boot attacks

A cold boot attack is a method used for obtaining encryption keys from devices. It is a type of attack that targets the contents of a computers RAM when the system is rebooted from a "cold" state. During the attack, the attacker gains access by physically removing a computers RAM or by rebooting the computer with a specially crafted bootable device. By doing so, the attacker can recover login credentials.

encryption keys, and other sensitive data that was stored in the RAM before the system got shut down, [2].

2.5 Communication

2.5.1 Inter-Integrated Circuit (I^2C)

I^2C, short for Inter-Integrated Circuit, is a protocol or set of rules used to connect devices and sensors to microcontroller boards or computers. Sensors and devices using I^2C only have four needed pins: power, ground, SCL (serial clock) and SDA (serial data). Transmission begins with a start condition initiated by the master, followed by sending a 7-bit address and a read/write bit. Each byte transferred requires the receiving device to acknowledge by pulling the SDA line low. Communication can include multiple bytes, each followed by an acknowledgment, and concludes with a stop condition set by the master, [3].

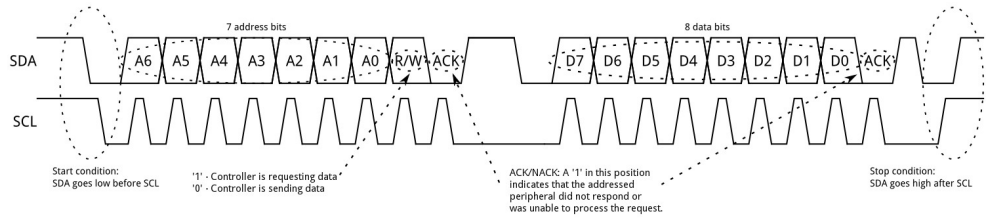


Figure 2.1: I^2C data transfer sequence, [3].

A distinctive feature of the I^2C protocol is its ability to manage multiple devices over the same bus. This is made possible by the unique addressing of each device, allowing simultaneous communication without signal collision. Devices on an I^2C bus are either assigned a fixed address, which is often printed on the device's PCB as shown in Figure 2.2 or a configurable address, adjustable via a jumper or pin setting, [4]. The fixed addresses can also impose a limitation when needing to add sensors that share the same address as each address can only be included once per bus to avoid address conflicts.

Figure 2.2: I^2C address on a sensor's silkscreen, [5].

2.5.2 False positive & false negative

False positives and false negatives refer to errors made by a predictive systems. A false positive occurs when a test incorrectly identifies a non harmful instance as a threat, leading to unnecessary outcomes based on an incorrect assumption. On the other hand, a false negative occurs when the system fails to detect an actual threat, allowing unauthorized access or tampering to go unnoticed, [6].

A false positive in an anti-tamper system might occur during normal runtime, triggering an alarm and causing unnecessary disruptions. More critically, a false negative would occur if a skilled attacker bypasses the system without detection, posing a significant security risk. Such a failure would undermine the system's effectiveness, and leave sensitive assets exposed.

2.6 Hardware

2.6.1 Zymkey4

Zymkey4 is a plug-in hardware security module developed for Raspberry Pi by Zymbit, [7]. It offers both data encryption and signing as well as multiple tamper sensors. By supporting cryptographic algorithms such as AES-256 this security module allows for strong and reliable encryption of data. In addition, its compatibility with popular programming languages such as C, C++ and Python caters to creating diverse applications based on specific needs. Furthermore, the anti-tampering mechanisms allow to easily detect an attempt of manipulation.

2.6.2 Raspberry Pi

Raspberry Pi is an affordable computer that operates on Linux and comes equipped with general-purpose input/output (GPIO) pins. These pins enable users to manipulate electronic elements, which is ideal for projects in physical computing and the Internet of Things (IoT), [8]. In this project, the Raspberry Pi's GPIO pins will be utilized to control the Zymkey4 and to collect data from sensors designed to detect tampering. The operating temperature for the Raspberry Pi ranges from 0 to 50°C, [9].

2.7 Cryptography

Cryptography is the study of encrypting and decrypting information for secure transmission between two people. This process involves two primary steps: transforming readable text (plaintext) into encoded text (cipher-text) to protect it during transmission, and subsequently converting the ciphertext back into plaintext for the intended recipient, [10].

2.7.1 Encryption

Encryption is the process of protecting (locking) data, by scrambling it and making it unintelligible for anyone who does not have a key to unlock it, [11]. In the realm of anti-tampering and information security, encryption is essential for safeguarding data. By concentrating on securing cryptographic keys instead of large amounts of plain-text data, one can substantially improve data security and refine anti-tampering efforts.

2.7.2 Symmetric key encryption

"Symmetric encryption is a type of encryption where only one key (a secret key) is used to both encrypt and decrypt electronic data. The entities communicating via symmetric encryption must exchange the key so that it can be used in the decryption process.", [12].

To illustrate this, a commonly known example of two friends, Alice and Bob will be used, [13]. In this scenario, Bob would like to send a private message to Alice via postal mail, but he is worried that the postman will read his letter. To prevent this from happening, Bob decides to put his letter inside a lockbox and lock it. Unfortunately, Bob cannot send the key alongside the box because the mailman will use it to open the box anyway. To circumvent this, Bob and Alice meet up in person and Alice is given an identical copy of the key that she can use to open the box. Now that both of them have a key, Bob can safely write a letter, put it in the box, lock it and send it to Alice for her to read the message.

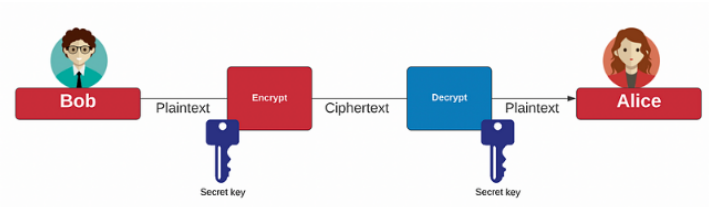


Figure 2.3: Symmetric key encryption, [14].

As seen in the example above, this method presents several challenges. For this type of encryption to be possible both the sender and the receiver must have access to the same key, meaning that if the key is lost or compromised, all encrypted data is at risk. Another challenge is the actual exchange of the key. Since the same key is used for both encryption and decryption, it must be shared between the communicating parties without being intercepted by unauthorized individuals. Asymmetric key encryption solves these problems.

2.7.3 Asymmetric key encryption

Before understanding how asymmetric key encryption works, it is important to know the types of keys this algorithm uses.

Public key This is the key that can be used by anyone to encrypt a message. It can be openly shared because a corresponding private key is needed to decrypt the message.

Private key This is the secret key that is used to decrypt a message encrypted with a public key. This key must be kept secret at all times.

In asymmetric key encryption, "anyone can encrypt messages using a public key, but only the holder of the paired private key can decrypt such a message. The security of the system depends on the secrecy of the private key, which must not become known to anyone", [15].

To illustrate this, let us revisit Bob and Alice. Using this algorithm, if Bob were to send a message to Alice, he would encrypt the message using Alices's public key. Alice, who possesses the corresponding private key, would therefor be the only one who could decrypt this message. Visually, imagine the encrypted message as a lockbox and Alice's public key as a padlock. This padlock, along with the lockbox, can be sent openly; however, it can only be unlocked with the corresponding padlock-key (private key), ensuring that only Alice can access the contents.

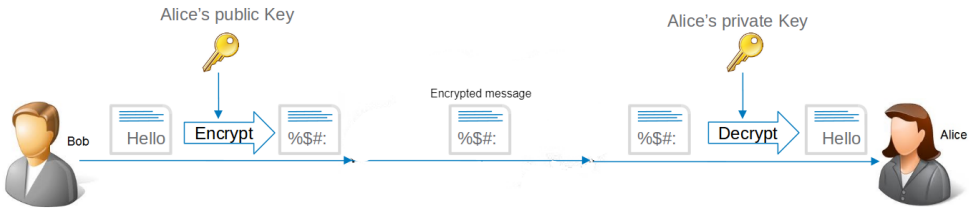


Figure 2.4: Asymmetric key encryption, [16].

As seen in the example above, this algorithm addresses one of the core weaknesses that symmetric key encryption presents, seeing as a compromised public key cannot be used to decrypt any information without also having the corresponding private key. However, this method is slower and more resource intensive than its counterpart, due to the fact that it employs two separate keys, that need to be linked together.

2.7.4 Data-in-transit vs. Data-at-rest security

Data at rest This refers to stationary data that is not actively moving. The main security risk with data at rest is unauthorized users accessing or tampering with it. Protection of data at rest can be done by encryption of the storage media, strong access controls and maintaining secure backups. The aim here is to ensure the confidentiality of the data and protect it from threats like theft and ransomware.

Data in transit This describes data that is "actively moving from one location to another such as across the internet or through a private network", [17]. The key security challenge for data in transit is safeguarding it from interception or manipulation during its journey. To do this, encryption protocols such as TLS or IPsec can be used. The aim here is to ensure confidentiality and integrity of the data as it moves between locations.

2.7.5 Algorithms

The process of transforming data into cipher-text can be accomplished using a wide variety of encryption algorithms that employ different key exchange methods. In the scope of this thesis however, these ones are the most important to know and understand;

Advanced Encryption Standard (AES) is a block cipher which is used in symmetric key encryption that encrypts data in 128-bit sized blocks. AES can support key sizes of 128, 192, or 256 bits and allows for fast processing with minimal computational overhead, making it ideal for encrypting large volumes of data efficiently.

Secure Hashing Algorithm (SHA) is a class of algorithms used to verify the integrity of data. It "takes an input (any data, like a file or a password) and produces a fixed-size string of characters, which is the hash value or digest", [18].

Diffie-Hellman (DH) is a key exchange method used to securely exchange cryptographic keys over a public network. It utilizes different MODP groups, which specify key lengths ranging from 768-bit to 3072-bit. These varying key lengths allow for different levels of security during the key exchange process.

Rivest–Shamir–Adleman (RSA) is a cryptosystem which is used in asymmetric key encryption that ensures secure data transmission. RSA handles both key generation and distribution as well as the encryption and decryption processes. RSA typically uses 1024 ,2048 or 4096-bit keys, but has the capability to support even larger key sizes, allowing for more robust security.

2.8 OSI model

"The Open Systems Interconnection (OSI) model is a conceptual framework that divides network communications functions into seven layers. The OSI data model provides a universal language for computer networking, so diverse technologies can communicate using standard protocols or rules of communication", [19].

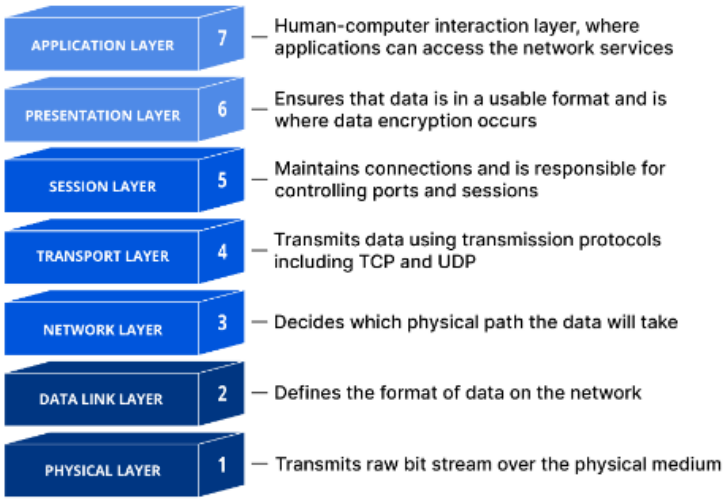


Figure 2.5: OSI Model, [20].

2.8.1 OSI model and encryption

Before deciding on a data encryption method, it is crucial to understand how encryption is done at different layers of the OSI model.

Layer 2 (Data link layer) "Data encryption at the Data Link Layer involves encrypting the data before it is transmitted over the physical medium. This process offers an extra layer of security to prevent unauthorized access to the data", [21]. MAC and LLC are examples of Data Link Layer encryption, focusing on securing data transmitted between nodes in a local network.

Layer 3 (Network layer) Encryption at this layer involves securing data as it traverses a network. Protocols such as IPsec can encrypt data packets, enabling them to be sent through secure tunnels to help preserve the confidentiality and integrity of them.

Layer 4 (Transport layer) The transport layer offers data encryption during its transmission using protocols such as TLS (Transport Layer Security). TLS ensures secure communications by encrypting the data transmitted over the Internet.

Layer 7 (Presentation layer) This is the layer that is most commonly associated with encryption. "It is responsible for the translation, encryption, and compression of data to ensure it is in a format that the application layer at the receiving end can understand", [22].

2.9 IPsec (Internet Protocol Security)

IPsec is a set of protocols used for security at the network layer of the OSI model. The primary goal of IPsec is to secure data when communication is happening between two points over a network. This is achieved by adding encryption and authentication to IP packets within a communication session, guaranteeing for integrity and confidentiality of data that travels through insecure networks, [23].

2.9.1 The benefits of using IPsec

When devices communicate over a public network such as the internet, the data that is being exchanged is sent in clear text, meaning that potential adversaries with a sufficient skill set can steal the data. Implementing IPsec significantly enhances security by encrypting this data, thus ensuring its confidentiality. Additionally, IPsec employs integrity checks to guarantee that the data remains unchanged and authentic during transit, [24].

2.9.2 IPsec modes

IPsec can be deployed in two different modes - transport mode and tunnel mode. When using transport mode, IPsec does not protect the IP header, only the payload coming from the transport layer. On the other hand, tunnel mode is designed to secure entire IP packets by encapsulating them with a new IP header.

"When comparing Tunnel Mode and Transport Mode, one key difference is the level of encryption provided. Tunnel Mode provides end-to-end security by encrypting the entire IP packet, while Transport Mode only encrypts the payload of the packet", [25].

2.9.3 IPsec protocols

IPsec consists of three primary protocols: Authentication Header (AH), Encapsulating Security Payload (ESP), and Internet Key Exchange (IKE). In combination, these protocols ensure access control, source authentication and enable the encryption of data.

AH handles data integrity and origin authentication of IP packets, by appending a header that can be used to detect changes. It is important to note however, that AH does not encrypt the packets.

ESP offers all the benefits of AH, with the addition of protecting the packet with an encryption algorithm.

IKE is a key exchange protocol that facilitates the creation of a secure and authenticated link between two peers. This ensures that transmitted data is secure and accessible only to the intended recipients, [26].

2.9.4 IPsec workflow

To ensure that data which is transferred over IP networks is secured against interception and tampering IPsec must adhere to a specific workflow, [27]. Below, the systematic process IPsec goes through to provide security effectively is described.

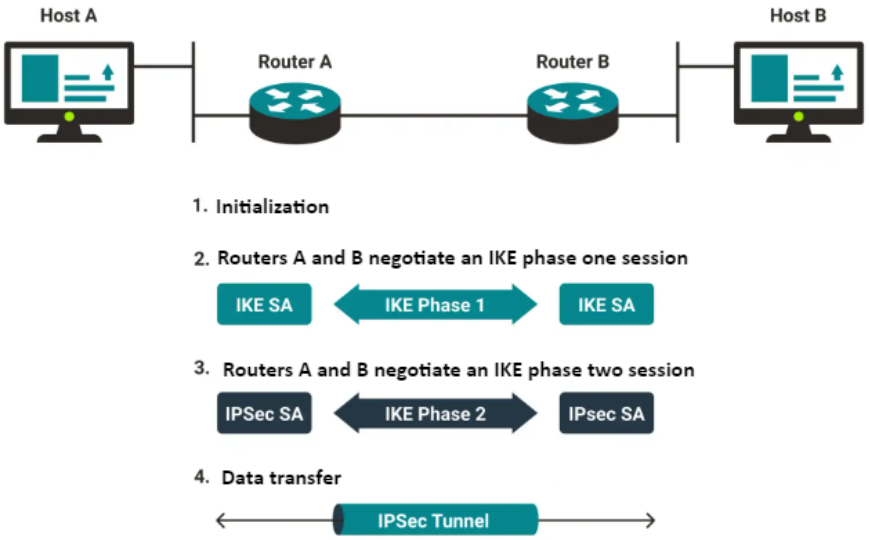


Figure 2.6: IPsec workflow

1. **Initialization**
   During the initialization phase, IPsec prepares the devices for secure communication. This involves loading configuration settings, such as security policies and rules, which dictate how IPsec should handle packets. These configurations define which traffic should be secured using IPsec and set the parameters for how security will be applied.

2. **SA establishment (IKE phase 1)**
   In this phase, the involved parties negotiate and establish a secure channel for subsequent security operations. During phase 1, the sender and the receiver select the cryptographic algorithms to be used for encryption and hashing. They agree on the authentication method and generate the initial key material necessary for authentication and encryption of the negotiation itself. This ensures that the further negotiation of security associations in phase 2 is protected against eavesdropping and tampering.

3. **Key and SA exchange (IKE phase 2)**
   Using the secure channel created in phase 1, the two parties exchange security associations that will be used for encrypting and decrypting IP traffic. Furthermore, the parties agree on which encryption and authentication methods will be used and generate session keys which are used in IPsec tunnels to enable secure data transmission across networks.

4. **Data transfer**
   The transferring of data involves encapsulating the original IP packets with additional headers containing the necessary security parameters. These encapsulated packets are then transmitted over the network, ensuring confidentiality, integrity, and authenticity of the data.

2.10 Attack potential to a communication channel

Communication channels between devices are susceptible to various forms of attacks, each employing different techniques to compromise security. In the scope of this thesis however, these ones are the most relevant ones to know and understand.

Table 2.4: Tools and techniques.

Technique	**Description**
IP spoofing	Masquerading as legitimate IP addresses to intercept or redirect communications.
Replay attacks	"Eavesdrops on network communication and replays messages at a later time, pretending to be the user", [28].
Denial of Service	Sending an excessive amount of requests to an IP address, overwhelming a server and making the service unavailable to legitimate users.
Cryptographic attacks	Exploiting improperly or weakly implemented encryption measures, to gain access to confidential data.
Man-in-the-middle Attack	Positioning oneself in a conversation between a user and an application, "either to eavesdrop or to impersonate one of the parties, making it appear as if a normal exchange of information is underway", [29].

Chapter 3

Execution of the project

In this chapter, the proposed anti-tampering measures for the ProtoKit5 will be outlined, along with the reasoning behind the chosen hardware and software security measures. The first section reviews the contents of the ProtoKit5 to clarify what is included in the kit. The second section explores the hardware security enhancements that will be incorporated into the ProtoKit5, as well as alternative solutions initially considered. Finally, the last section details the proposed software security measures designed to ensure data security and secure communication.

3.1 ProtoKit5

ProtoKit5, developed by Zymbit, is a security kit designed for Pi computers. The kit is waterproof, dustproof, radio-transparent, and tamper-evident. It includes the components shown in Figure 3.1.

1. IP67 enclosure, dustproof, waterproof enclosure with lid
2. Integrated protoboard
3. Adapter for perimeter circuits
4. GPIO connector extension
5. Standoffs for Pi, M2.5 - x3
6. Screws for Pi, M2.5 Torx T8 - x3
7. Screws for protoboard fix - x5
8. Cable glands, 3/4 - x2
9. IP67 lid with neoprene gasket
10. Screws for lid - x6
11. Tamper switch actuators - x2

The integrated protoboard included with the kit allows users to implement their own circuits to enhance the protection of the Pi computer. It is specifically designed for use with a Zymbit security module, which provides data encryption, signing capabilities, and multiple tamper sensors. Additionally, the protoboard features two switches (as shown in Figure 3.2) that trigger a response when the lid is lifted, detecting any movement of the lid.

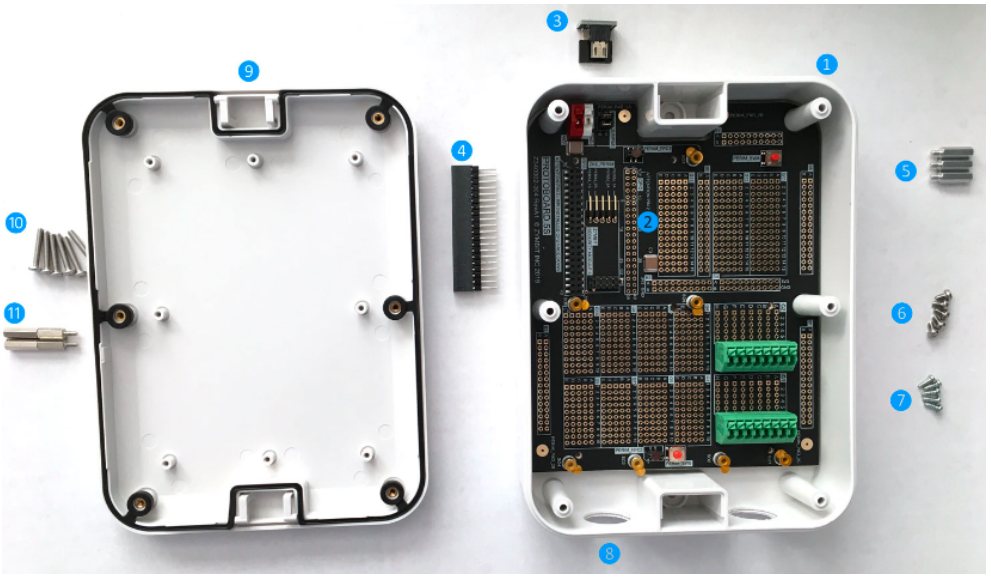


Figure 3.1: Components in the ProtoKit5, [30].

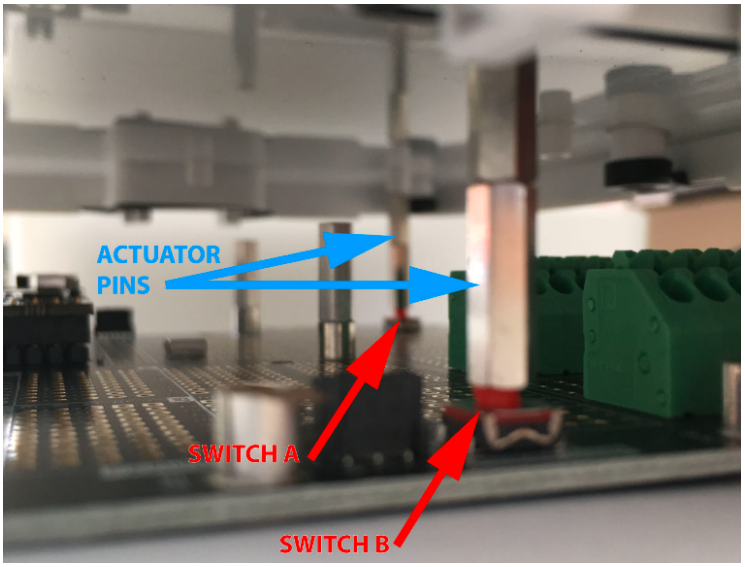


Figure 3.2: Actuator pins pressing the switches, [30].

3.2 Flex PCB

The protoboard in the development kit is equipped with basic tamper switches that can be easily bypassed by creating a large hole in the enclosure while keeping the switches pressed down. A potential attacker could create the hole through drilling, which generates vibrations, or by using thermal tools, such as a hot knife, to melt

the cover. While a vibration sensor can detect the vibrations from drilling, it may not detect the melting of the cover. A proposed security measure for both scenarios is to create an anti-tampering foil around the enclosure to detect any holes made. This can be achieved using a flex PCB that lines the inside of the enclosure, forming a closed circuit. If a hole is made in the enclosure, it will also break the circuit on the PCB. The PCB is connected to the Zymkey, which will detect the break in the circuit and trigger the deletion of the encryption key.

3.2.1 Measurement

As the intention of the flex PCB is to be used as an anti-tamper foil, a custom shape is needed to cover the inside of the enclosure. The physical specifications were measured by hand using a measuring tape, pen, and paper. After determining all the physical specifications, the shape of the PCB was drawn using KiCad.

3.2.2 Design with KiCad

KiCad, a free open-source software, was used to design the PCB. It was selected due to its user-friendly interface. After verifying the sizes, which are described in the next section, the process of designing the traces began. The detection system operates on a closed circuit, [30] enabling the flex PCBs to be connected to the protoboard as shown in Figure 3.3. The copper trace was designed with this in mind. The trace width is set to 0.2 mm and spacing at 3 mm center-to-center. To prevent design flaws, design guidelines from NCAB (ver. 1.2) were followed, [31].

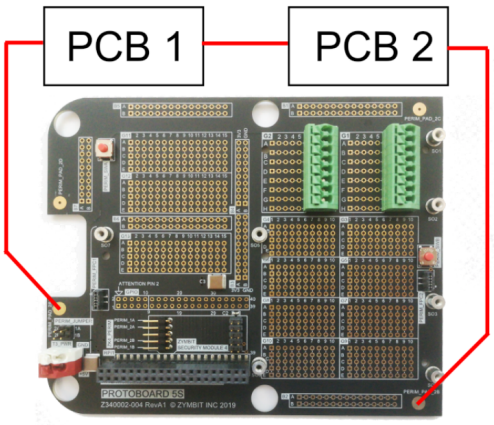


Figure 3.3: Illustration of how flex PCBs will be connected.

3.2.3 Testing the size

To ensure cost efficiency, the design was first printed on an A4 sheet of paper and then cut by hand. This was done to verify that the shape of the PCB would fit, without unnecessarily spending time and money on producing new PCBs for

each iteration. Printing the shape on the paper did however not result in same physical sizes as the design. As a result, the print settings had to be adjusted to print 103% size to compensate for the error. The first design did not fit perfectly likely due to error in physical measurements. The differences were noted and small adjustments were made to the design before printing it on the paper again. The iteration continued until the cut out paper perfectly fit in the enclosure. After being satisfied with the results, an order was sent to a manufacturer to produce the PCB.

Figure 3.4: "Paper PCBs" to verify correct measurements.

3.2.4 After receiving from the manufacturer

Three pieces of each flex PCB were ordered, with one serving as a reserve in case of damage. Upon receiving the order, the flex PCBs were glued to the inside of the enclosure. The flex PCBs were soldered to the protoboard on the pads marked "PERIM_PAD_2A" and "PERIM_PAD_2B", as illustrated in Figure 3.3. The pads "PERIM_PAD_1A" and "PERIM_PAD_1B" were not used. During the first iteration, one of the flex PCBs was damaged, as shown in Figure 3.5. A solution to this was to glue an acrylic sheet behind the solder pad to provide additional support and to use softer wires to prevent the same damage from occurring again.

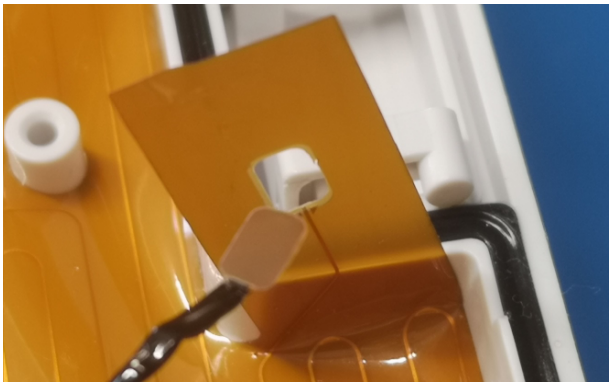


Figure 3.5: Solder pad torn off from flex PCB.

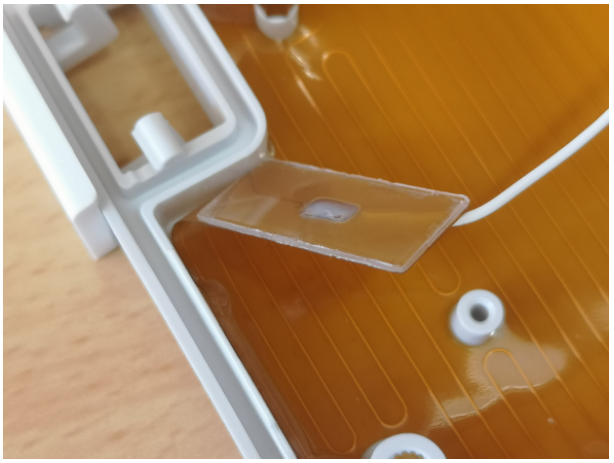


Figure 3.6: Quick fix to address the weakness.

3.3 Sensors for tampering detection

Sensors have been selected for inclusion in the anti-tamper enclosure due to their ease of implementation and their capability to effectively detect tampering. The sensors chosen are compatible with the Raspberry Pi and will all be communicating through I^2C, each with unique addresses to prevent communication conflicts on the same bus. All the sensors are purchased from Adafruit, as the sensors have a distinct address based on their sensor-type. Additionally the sensors were chosen for their reliability and effectiveness in tamper detection.

3.3.1 Temperature and humidity

A temperature and humidity sensor was chosen for the enclosure as it provides a reliable method to detect unusual environmental changes, signaling potential tampering or unauthorized access attempts. Given the harsh Scandinavian weather,

the enclosure will be placed indoors, as the Raspberry Pi inside operates optimally between 0 and 50°C, a range that external conditions frequently exceed.

As the enclosure is placed indoors, an unexpected rise in temperature could signal the use of thermal tools such as heat guns and thermal cutters to breach the enclosure, while a significant drop in temperature might indicate a cold boot attack. Similarly, a sudden increase in humidity could also suggest tampering or the introduction of foreign substances intended to disable or damage the device. For example, introducing steam or sprays could trigger condensation or directly manipulate the electronics. A humidity sensor would be crucial in detecting such irregularities.

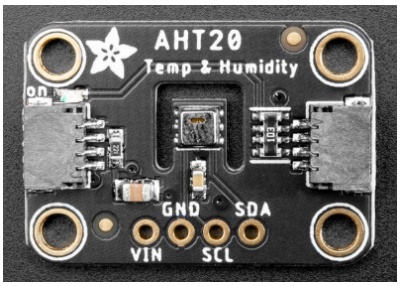


Figure 3.7: Temperature and humidity sensor used, [32].

The sensor used to detect tampering is the Adafruit AHT20 as shown in Figure 3.7. This sensor operates with an I^2C interface at address 0x38 and is designed to function within an extensive temperature range from -40 to 85°C, offering a typical accuracy of ±1°C across this entire range. Additionally, it can measure relative humidity from 0 to 100% RH, with a typical accuracy of ±3% over the full range, [32].

After installing all the necessary Adafruit libraries to acquire sensor data, [33]. Code listing 3.1 displays the script that was executed to gather data, which is summarized in Table 3.1.

Code listing 3.1: Python script for interval-based temperature and humidity monitoring.

```
import time
import board
import adafruit_ahtx0

i2c = board.I2C()  # uses board.SCL and board.SDA
sensor = adafruit_ahtx0.AHTx0(i2c)

#Gathers sensor data in 30 min intervals
while True:
    print("\nTemperature: %0.1f C" % sensor.temperature)
    print("Humidity: %0.1f %%" % sensor.relative_humidity)
    time.sleep(1800) #Delay/sleep for 30 min
```

The table showcases data from the sensor placed within the enclosure at rest, recording temperature and humidity levels at 30-minute intervals over a span of 4 hours. The temperature exhibits a steady rise throughout the period, beginning at 23.0°C and ending at 26.8°C. Humidity, however, shows an inverse trend, decreasing from 38.7% to 30.7%. This pattern suggests that the internal atmosphere of the enclosure becomes warmer and less humid over time, which could be attributed to factors such as the heat generated by the enclosed electronics.

Table 3.1: Temperature and humidity readings over a four-hour period.

Time (min)	Temperature (°C)	Humidity (%)
0	23.0	38.7
30	25.8	33.6
60	26.4	31.8
90	26.7	31.0
120	26.7	30.7
150	26.6	30.7
180	26.7	30.5
210	26.7	30.3
240	26.8	30.7

3.3.2 Accelerometer and gyroscope

The enclosure will be mounted directly to a wall or flat surface, and integrating an accelerometer and a gyroscope is strategic for detecting any unauthorized movements. The accelerometer can sense acceleration in all directions, crucial for signaling if the enclosure is being moved, tilted, or shaken. The gyroscope complements this by detecting changes in orientation and rotation, providing a comprehensive security measure against attempts to displace or manipulate the enclosure.

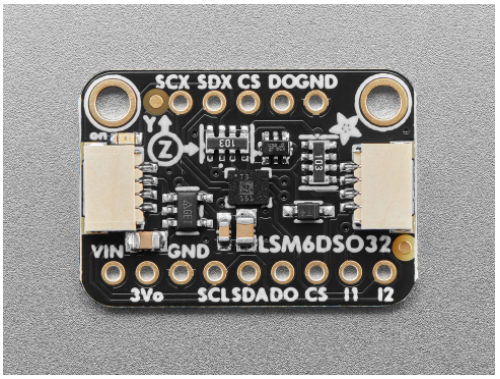


Figure 3.8: Accelerometer and gyroscope sensor used, [34].

The sensor used to detect tampering is the Adafruit LSM6DSO32 as shown in Figure 3.8. This sensor operates with an I^2C interface at address 0x6A and is designed to function within an extensive temperature range from -40 to 85°C. It offers a high update rate ranging from 1.6 Hz to 6.7 kHz. The accelerometer supports sensitivity settings of ±4, ±8, ±16, and ±32 g, while the gyroscope can measure angular velocity across a range of ±125, ±250, ±500, ±1000, and ±2000 degrees per second. The sensor also includes advanced features such as tap detection, activity detection, a pedometer, and a programmable machine learning core for basic gesture recognition, none of which will be utilized in the final product, [34].

After setting up the sensor inside the enclosure, Code listing 3.2 was used to gather data over a four-hour period while the sensor remained at rest. The results of the script are summarized in Table 3.2 and Table 3.3.

Code listing 3.2: Python script for interval-based monitoring of the acceleration and rotation.

```
import time
import board
from adafruit_lsm6ds.lsm6dsox import LSM6DSO32

i2c = board.I2C()  # uses board.SCL and board.SDA
sensor = LSM6DSO32(i2c)

#Gathers sensor data in 30 min intervals
while True:
    print("Acceleration:␣X:%.2f,␣Y:␣%.2f,␣Z:␣%.2f␣m/s^2" % (sensor.acceleration))
    print("Gyro␣X:%.2f,␣Y:␣%.2f,␣Z:␣%.2f␣radians/s" % (sensor.gyro))
    print("")
    time.sleep(1800)
```

The acceleration readings on the X and Y axes show only minor fluctuations around zero, which are within the expected range for a sensor at rest due to slight sensor noise. The Z axis consistently reads close to the expected gravitational acceleration of 9.81 m/s^2.

Table 3.2: Acceleration readings in the X, Y, and Z axes (m/s^2) over four-hour period.

Time (min)	Acceleration X (m/s^2)	Acceleration Y (m/s^2)	Acceleration Z (m/s^2)
0	0.06	0.11	9.91
30	0.49	-0.19	9.89
60	0.48	-0.18	9.90
90	0.48	-0.17	9.92
120	0.06	0.03	9.90
150	0.05	0.01	9.89
180	0.10	0.02	9.90
210	0.24	0.01	9.91

Continued on next page

Table 3.2 continued from previous page

Time (min)	Acceleration X (m/s^2)	Acceleration Y (m/s^2)	Acceleration Z (m/s^2)
240	0.19	0.01	9.92

The reading of the gyroscope across the X, Y, and Z axes are relatively stable over the four-hour period. With the gyroscope values hovering close to zero, it suggests that the sensor remained motionless, as expected for a sensor within a stationary enclosure. Minor deviations are likely due to the sensitivity of the gyroscope and inherent noise, rather than actual movement, which aligns with the expected behavior for a non-moving device.

Table 3.3: Rotation measurement on the X, Y, and Z axes in degrees/sec over a four-hour period.

Time (min)	Gyro X (radians/s)	Gyro Y (radians/s)	Gyro Z (radians/s)
0	0.01	-0.01	-0.01
30	0.01	-0.00	-0.01
60	0.01	-0.00	-0.01
90	0.01	-0.00	-0.01
120	0.01	-0.00	-0.01
150	0.01	-0.00	-0.01
180	0.01	-0.00	-0.01
210	0.01	-0.00	-0.01
240	0.01	-0.00	-0.01

3.3.3 Proximity and lux

The ProtoKit5 is equipped with buttons designed to sense if the enclosure's cover has been lifted. Additionally, the custom made flexible PCB is designed to identify any drilling or the creation of holes through the enclosure. In the event these measures are bypassed, an additional layer of security is provided by a proximity and lux sensor, which gauges the distance to the enclosure, and monitors the light levels inside.

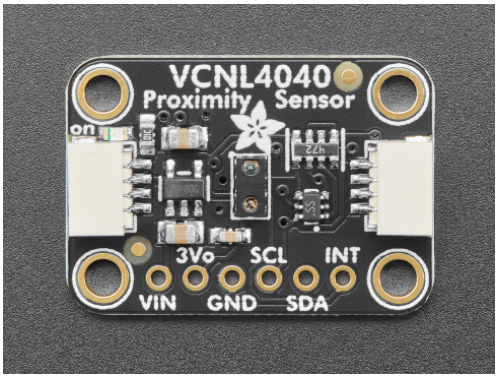


Figure 3.9: Proximity and lux sensor used, [5].

The sensor used for proximity and light measurement in the enclosure is the Adafruit VCNL4040 as shown in Figure 3.9. Operating with an I^2C interface at address 0x60, the sensor has a proximity detection capability from 0 to 200 mm and a light sensing range from 0.0125 to 6 553 lux, [5].

After setting up the sensor inside the enclosure, Code listing 3.3 was used to gather data over a four-hour period while the sensor remained at rest. The results of the script are summarized in Table 3.4.

Code listing 3.3: Python script for interval-based light and proximity monitoring.

```
import time
import board
import adafruit_vcnl4040

i2c = board.I2C()  # uses board.SCL and board.SDA
sensor = adafruit_vcnl4040.VCNL4040(i2c)

#Gathers sensor data in 30 min intervals
while True:
    print("Proximity:", sensor.proximity)
    print("Light:␣%d␣lux" % sensor.lux)
    time.sleep(1800) #Delay/sleep for 30 min
```

The data in Table 3.4 shows consistent proximity readings with minor fluctuations, indicating that the environment around the sensor remained stable over the four-hour period. Lux readings also remained steady, primarily registering at 2 lux with an occasional increase to 3 lux, suggesting minimal changes in light conditions. Overall, the sensor's environment was largely unchanged, which is ideal since the sensor will easily be able to detect any tampering attempts.

Table 3.4: Proximity and lux readings over a four-hour period.

Time (min)	Proximity	Lux
0	57	2
30	56	2
60	55	2
90	55	2
120	55	2
150	55	2
180	54	3
210	55	2
240	55	2

3.3.4 Short term variation and noise of sensors

When conducting short-term tests on the sensors, all maintained consistent values over the period, with the exception of the accelerometer. During tests performed on the stationary enclosure over a 10-second timeframe, the accelerometer produced the readings as shown in Table 3.5. This indicates that, unlike the other sensors, the accelerometer is susceptible to noise.

Table 3.5: Acceleration readings in the X, Y, and Z axes (m/s^2) over a 10-second period.

Time (sec)	Acceleration X (m/s^2)	Acceleration Y (m/s^2)	Acceleration Z (m/s^2)
1	0.04	0.09	9.91
2	0.06	0.07	9.89
3	0.05	0.07	9.91
4	0.04	0.06	9.92
5	0.05	0.06	9.92
6	0.05	0.07	9.92
7	0.03	0.09	9.89
8	0.04	0.08	9.91
9	0.05	0.08	9.90
10	0.05	0.09	9.90

3.3.5 Assembling to the Protoboard

Given the size of the sensors and the configuration of the Raspberry Pi, there are physical constraints on placing the sensors on the protoboard. Figure 3.10 illustrates the available space where sensors can be placed without interfering with the Raspberry Pi.

The sensors will be soldered within the designated areas. Only four nodes, namely

SDA, SCL, 3V (VIN on sensors), and GND, are required for operation. The connection pins for the sensors are marked, while the GPIO pins from the Raspberry Pi can easily be identified on the protoboard by consulting its schematic [30].

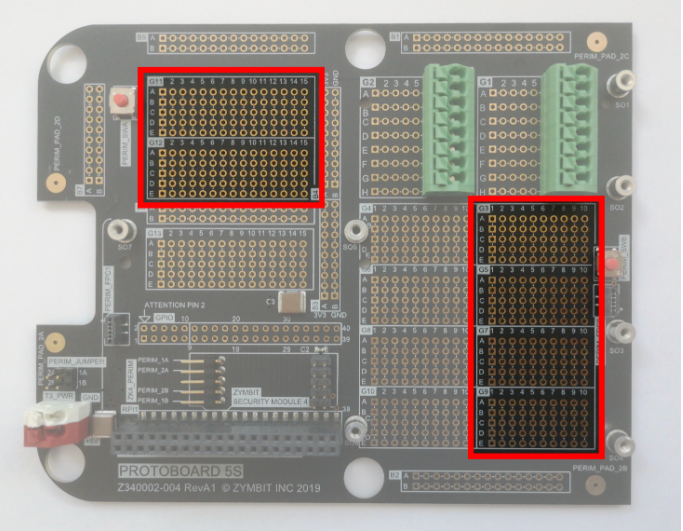


Figure 3.10: Areas available for sensor integration.

3.4 Technical Design

The primary use-case for the project involves a communication process that entails a user operating the client-side Raspberry Pi sending messages to the server-side Raspberry Pi. Upon being received on the server end, these messages are stored within a directory called "ReceivedMessages." To protect the confidentiality of this data, each message is encrypted using the Zymkey before being saved. This measure is set in place to ensure the confidentiality of the messages in the event of the server Raspberry Pi being compromised or stolen. In addition, IPsec, equipped with supplemental hand-crafted security checks is used to safeguard the messages as they are being transmitted between the client and server. This is done to prevent potential eavesdroppers from accessing or deciphering the messages while they are in transit.

Before getting to the implementation of the desired design, the steps described below had to be taken.

3.4.1 Prerequisites - Raspberry Pi

First and foremost, fresh operating systems were installed using the Raspberry Pi Imager. Using this straightforward and reliable method of loading operating systems onto SD cards, the project group ensured that each Raspberry Pi began with a clean and updated software environment.

To circumvent the logistical challenges associated with managing multiple devices through direct HDMI connections, each Raspberry Pi was configured for remote access using RealVNC Viewer. This setup enabled wireless control of the devices, allowing for simultaneous and efficient management of multiple Raspberry Pis without the need for physical switching between screens.

3.4.2 Prerequisites - Zymkey

The succesful installation of the Zymkeys were achieved by following the quick-start guide written by Zymbit, found at [35].

It is important to note that during testing, unexpected errors were encountered, leading to the operating system on one of Raspberry Pis having to be reinstalled. Coincidentally, this period overlapped with an update to the Raspberry Pi OS Bookworm, which upgraded to kernel version 6.6.y changing how the GPIO base numbers are configured, affecting the functionality of the Zymkey. Unaware of these changes, persistent issues with binding the Zymkey were faced, leading to the belief that the device had malfunctioned, prompting the order of a replacement.

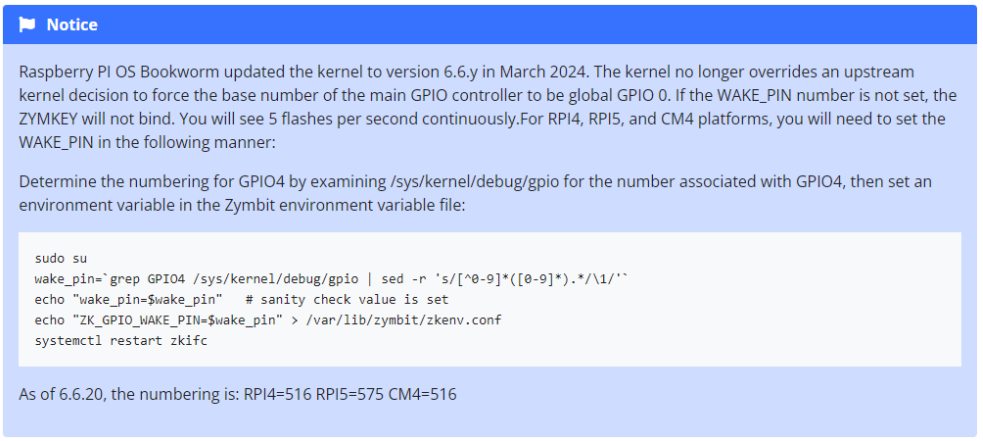


Figure 3.11: Raspberry Pi - OS Update.

A week later, clarification on how the kernel update impacted the Zymkey was published (Figure 3.11), resolving the issue and allowing continued work on the project without waiting for the new Zymkey to be delivered.

3.5 IPsec configuration

IPsec was chosen to protect the communication channel between the two Raspberry Pis. This method was selected due to its reliability and the fact that it protects the channel from multiple attacks mentioned previously in section 2.10.

Attack	Counter-measure
IP spoofing	IPsec will ensure that the packets which are sent come from a legitimate source, using either its AH or ESP protocols.
Cryptographic attacks	The IPsec setup will use strong encryption algorithms and key management protocols, ensuring that no cryptographic attacks can be successfully launched.
Man-in-the-middle attacks	IPsec will ensure that potential attackers cannot insert themselves between the communicating parties by using its authentication methods.

Table 3.6: IPsec counter-measures against various types of attacks.

3.5.1 Strongswan

"Strongswan is an open-source, modular and portable IPsec-based VPN solution", [36]. The decision to use Strongswan for implementing IPsec was based on its simple configuration and user-friendly interfaces, which made setting up and maintaining IPsec tunnels easier.

3.5.2 IPsec authentication

The IPsec setup uses certificate-based authentication, which is a method that "employs a digital certificate to identify a user, device or machine, before granting access to an application, network or other resource", [37]. Initially, pre-shared secrets were used during testing, but due to a potential risk of key compromise, certificate-based authentication was chosen for the final product.

3.5.3 IPsec parameter explanation

The server (subsection 3.5.4) and client (subsection 3.5.5) IPsec configuration was set up with the following parameters:

Encryption and Authentication Algorithms AES with a key length of 256 bits for both the IKE and ESP protocols was chosen. This was done to ensure a high level of encryption, suitable for safeguarding sensitive data against brute-force attacks. SHA256 was used for data integrity, due to it being considered one of the most secure hashing algorithms today, [18]. Lastly MODP2048 was chosen for the Diffie-Hellman group to ensure a robust key exchange process.

Key Exchange Mechanisms IKEv2 was chosen as the key exchange protocol due to its efficiency and security. This was done to ensure fast and secure establishment of IPsec VPN tunnels.

Networking settings The server Raspberry Pi was set to accept communications coming from any IP addresses using *right=%any*. Although this setting allows connections from anyone, the certificate-based authentication is in place, to prevent unwanted connections. By not restricting the server to accept only specific IP addresses, the potential future scalability of the product is increased.

3.5.4 Server-side IPsec configuration

Code listing 3.4: IPsec configuration for the server.

```
config setup
    charondebug=""

conn %default
    ikelifetime=60m
    keylife=20m
    rekeymargin=3m
    keyingtries=1
    keyexchange=ikev2
    authby=rsasig
    leftcert=serverCertificate.pem
    rightid=%any

conn secureCommunication
    left=192.168.0.115
    leftsubnet=192.168.0.115/32
    right=%any
    rightsubnet=0.0.0.0/0
    auto=add
    keyexchange=ikev2
    ike=aes256-sha256-modp2048
    esp=aes256-sha256-modp2048
```

3.5.5 Client-side IPsec configuration

Code listing 3.5: IPsec configuration for the client.

```
config setup
    charondebug=""

conn %default
    ikelifetime=60m
    keylife=20m
    rekeymargin=3m
    keyingtries=1
    keyexchange=ikev2
    authby=rsasig
    leftcert=clientCertificate.pem
    rightid=%any

conn secureCommunication
    left=192.168.0.114
    leftsubnet=192.168.0.114/32
```

```
    right=192.168.0.115
    rightsubnet=192.168.0.115/32
    auto=add
    keyexchange=ikev2
    ike=aes256-sha256-modp2048
    esp=aes256-sha256-modp2048
```

3.6 Scripts

The scripts were written in Python. Despite limited prior knowledge of this language, its easy syntax and large number of useful libraries were the driving factors behind this decision.

3.6.1 Server-side script

This script is an adaptation of [38] with additional functionality and security checks.

The purpose of this script is to setup a server which is used to receive connections and securely store messages sent from clients. The script begins by checking whether the IPsec tunnel is up, and only then allows for the server to be started. Once a connection is established, it begins listening for incoming messages, storing them in a designated folder upon receipt. The script ensures that the messages saved to this folder are encrypted using the Zymkey encryption functions, guaranteeing that the data remains unreadable even if the physical device of the server is stolen.

In addition, the script continuously monitors the status of the IPsec tunnel and will shut down automatically if it goes down, ensuring that data can not be sent over an untrusted channel.

Code listing 3.6: Client script for connecting to the server.

```
import os
import socket
import zymkey
import threading
import subprocess
import time

def checkVpnStatus(): #Checks IPsec status
    try:
        result = subprocess.run(['sudo', 'ipsec', 'status', 'secureCommunication'],
             capture_output=True, text=True)
        return 'INSTALLED' in result.stdout
    except Exception as e:
```

```
        print("Failed to check VPN status:", str(e))
        return False

def getNextNumber(directory): #Used for setting the correct file name for when
     conversation file will be saved
    files = os.listdir(directory)
    conversation_numbers = [int(f.split('_')[-1].split('.')[0]) for f in files if f
         .startswith('Conversation')]
    if conversation_numbers:
        return max(conversation_numbers) + 1
    else:
        return 1

def receive_data(): #Initiates a socket connection
    host = ""
    port = 5005

    s = socket.socket()
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind((host, port))
    print("Waiting for connection...")
    s.listen()

    try:
        while True:
            c, addr = s.accept()
            print("Connection received from:", addr)
            directory = "ReceivedMessages"
            if not os.path.exists(directory):
                os.makedirs(directory)

            conversation_number = getNextNumber(directory)
            filename = "Conversation_" + str(conversation_number) + ".txt"
            filepath = os.path.join(directory, filename)

            with open(filepath, "w") as file:
                while True:
                    data = c.recv(1024)
                    if not data:
                        print("Client disconnected, message has been written to
                             file.")
                        break
                    print("Message received")
                    value = data.decode()
                    file.write(value + "\n")
            lock_data(filepath)
            c.close()

    except KeyboardInterrupt:
        print("Server terminated by user.")

def lock_data(filepath): #Prepares the message for encryption
    try:
        with open(filepath, "rb") as file:
            data = file.read()
    except FileNotFoundError:
        print("File not found:", filepath)
        return

    try: #Encrypts the message
```

```
        encrypted_data = zymkey.client.lock(data)
        with open(filepath, "wb") as file:
            file.write(encrypted_data)
        print("File locked and updated successfully.")
    except Exception as e:
        print("Error occurred during locking:", str(e))

if __name__ == "__main__":
    if checkVpnStatus():
        print("VPN is up, starting the server...")
        print("Type "SHUTDOWN" in the console to terminate the server.")
        server_thread = threading.Thread(target=receive_data)
        server_thread.start()

        while True:
            time.sleep(5)
            if not checkVpnStatus(): #Constact check of IPsec tunnel presence
                print("VPN is down, shutting down the server...")
                os._exit(0)
            user_input = input()
            if user_input.strip() == "SHUTDOWN": #Manual shutdown of the server
                print("Shutting down server...")
                os._exit(0)
    else:
        print("VPN is down, server cannot be started.")
```

3.6.2 Client-side script

This script is an adaptation of [39] with additional functionality added.

This simple script is used to establish a client-side connection to the server. Once connection is established, it prompts the user to send messages, which are then received on the server. Once the client-user is done sending messages they close the connection by typing in “SHUTDOWN” in the console.

Code listing 3.7: Client script for connecting to the server.

```
import socket
import sys

host = '192.168.0.115'
port = 5005

s = socket.socket()
s.connect((host, port))
print(f"Connected to server at IP address {host}.")
print('Send message "SHUTDOWN" to exit the client.')

try:
    while True:
        message = input("Message -> ")
        if message.strip() == "":
            continue
        if message.upper() == "SHUTDOWN":
            s.close()
            print("Connection closed.")
```

```
            sys.exit(0)
        s.send(message.encode())
except KeyboardInterrupt:
    s.close()
    print("Connection closed.")
    sys.exit(0)
```

3.6.3 Decryption script

This script decrypts the contents which are stored in the "ReceivedMessages" folder using the Zymkey decryption functions. It is important to note that this script can only decrypt messages which were encrypted using the same Zymkey.

Code listing 3.8: Decryption script for unlocking the "ReceivedMessages" folder.

```
import os
import zymkey
from zymkey.exceptions import VerificationError

folder_path = "ReceivedMessages" #Folder in which the messages are stored

def decrypt_and_update_file(file_path): #Opens the respective file and decrypts its
      contents
    try:
        with open(file_path, "rb") as file:
            encrypted_data = file.read()
    except FileNotFoundError:
        print("File not found:", file_path)
        return
    try:
        decrypted_data = zymkey.client.unlock(encrypted_data)
        with open(file_path, "wb") as file:
            file.write(decrypted_data)
        print("File '{}' decrypted successfully.".format(file_path))

    except VerificationError:
        print("An error occurred during decryption of file '{}':".format(file_path)
             , str(e))
    except Exception as e:
        print("An error occurred during decryption of file '{}':".format(file_path)
             , str(e))

for filename in os.listdir(folder_path): #Loops through each file in the folder
    file_path = os.path.join(folder_path, filename)
    if os.path.isfile(file_path):
        decrypt_and_update_file(file_path)
```

While this script is not actively used in the communication process, it was important to have a way to decrypt the entire folder if need be. The thought process behind the implementation of this script, was a scenario where the server owner, might require access to the plain-text contents of the "ReceivedMessages" folder for potential extraction to another location. This potential use-case led to the creation of this script.

3.6.4 Encryption script

This script encrypts files stored in the "ReceivedMessages" folder using the Zymkey encryption functions. The script is designed to loop through every file in this folder, overwriting its content with the new encrypted version.

Code listing 3.9: Encryption script for encrypting the "ReceivedMessages" folder.

```
import os
import zymkey

folder_path = "ReceivedMessages" #Folder in which the messages are stored

def encryptFile(file_path): #Opens the respective file and encrypts its contents
    try:
        with open(file_path, "rb") as file:
            data = file.read()
    except FileNotFoundError:
        print("No files in the directory found:", file_path)
        return

    try:
        encrypted_data = zymkey.client.lock(data)
        with open(file_path, "wb") as file:
            file.write(encrypted_data)
        print("File '{}' encrypted successfully.".format(file_path))
    except Exception as e:
        print("An error occurred during encryption of file '{}':".format(file_path)
             , str(e))

for filename in os.listdir(folder_path): #Loops through every file in the folder
    file_path = os.path.join(folder_path, filename)
    if os.path.isfile(file_path):
        encryptFile(file_path)
```

This script was designed as a counter-part to the decryption script mentioned previously. While this exact version is not actively used in the communication process either, it was important to have a way to manually encrypt the entire folder if need be.

3.6.5 VPN monitoring using Zymkey4

This script is designed to respond to any tampering with the Zymkey, by shutting down the IPsec tunnel using built-in perimeter breach detection mechanisms. It serves a crucial role in maintaining the security of the connection, seeing as the communication is set up to be operational only within the secure confines of an active IPsec tunnel. By having this security check run in the background of the system, the continuous security and integrity of the tunnel is ensured, which in turn protects the communication process itself.

Code listing 3.10: Zymkey monitoring for the VPN.

```
import time
import os
import subprocess
import zymkey
import sys

def is_vpn_tunnel_up(): #Checks for IPsec tunnel presence
    try:
        output = subprocess.check_output(["sudo", "ipsec", "status", "
            secureCommunication"], stderr=subprocess.STDOUT, text=True)
        if "INSTALLED" in output:
            return True
    except subprocess.CalledProcessError as e:
        print(f"Failed to check VPN status: {e.output}")
    return False

def manage_vpn_tunnel():
    vpn_was_up = False

    def tampering_detected(): #Brings the tunnel down if tampering is detected
        os.system("sudo ipsec down secureCommunication")
        print("Breach detected! VPN tunnel deactivated.")

    #Initiates perimeter events for both channel 0 and 1
    zymkey.client.set_perimeter_event_actions(channel=0, action_notify=True,
        action_self_destruct=False)
    zymkey.client.set_perimeter_event_actions(channel=1, action_notify=True,
        action_self_destruct=False)

    vpn_is_up = is_vpn_tunnel_up()
    vpn_was_up = vpn_is_up
    if vpn_is_up:
        print("VPN tunnel is currently up.")
    else:
        print("VPN tunnel is currently down.")

    while True:
        time.sleep(2)
        vpn_is_up = is_vpn_tunnel_up()
        if vpn_is_up and not vpn_was_up:
            print("VPN tunnel has just been started.")
        elif not vpn_is_up and vpn_was_up:
            print("VPN tunnel has just been stopped.")
        elif vpn_is_up:
            print("VPN tunnel is currently up.")
        else:
            print("VPN tunnel is currently down.")
        vpn_was_up = vpn_is_up

        try:
            zymkey.client.wait_for_perimeter_event(timeout_ms=200)
            tampering_detected()
            sys.exit(0)
        except zymkey.exceptions.ZymkeyTimeoutError:
            pass

if __name__ == "__main__":
```

```
    manage_vpn_tunnel()
```

3.6.6 Zymkey monitoring script

This script is an adaptation of the following scripts; [40], [41], [42].
This script is designed to check sensor readings and respond to irregularities by destroying the Zymkey. By constantly making sure that the proximity, light, temperature, humidity and acceleration levels are within their specified boundaries, additional tamper-detection can be achieved. Moreover, this script adds an additional layer of security, by ensuring that even if potential adversaries were to bypass the in-built perimeter detection mechanisms of the Zymkey itself, the sensor values can prompt the destruction of the Zymkey nevertheless.

Code listing 3.11: Zymkey self detruct script based on sensor values.

```
import zymkey
import time
import board
import adafruit_vcnl4040
import adafruit_ahtx0
from adafruit_lsm6ds.lsm6dso32 import LSM6DSO32

i2c = board.I2C()
vcnl4040_sensor = adafruit_vcnl4040.VCNL4040(i2c)
ahtx0_sensor = adafruit_ahtx0.AHTx0(i2c)
lsm6dso32_sensor = LSM6DSO32(i2c)

#Parameter values taken from table 3.2
MAXVALUE = {
    "Proximity": 61,
    "Lux": 6,
    "Temperature": 32,
    "Humidity": 45,
    "Acceleration": 1,
    "Gyro": 0.1
}
MINVALUE = {
    "Proximity": 50,
    "Lux": None,
    "Temperature": 15,
    "Humidity": None,
    "Acceleration": -1,
    "Gyro": -0.1
}

#Base acceleration as described in subsection 3.3.2
BASE_ACCELERATION = (0, 0, 9.81)


#Initiate perimeter detection on both channel 1 and 0
zymkey.client.set_perimeter_event_actions(0, action_notify=True,
    action_self_destruct=False)
zymkey.client.set_perimeter_event_actions(1, action_notify=True,
    action_self_destruct=False)
```

```
def check_sensors():
    print("")
    proximity = vcnl4040_sensor.proximity
    lux = vcnl4040_sensor.lux
    temperature = ahtx0_sensor.temperature
    humidity = ahtx0_sensor.relative_humidity
    acceleration = lsm6dso32_sensor.acceleration
    gyro = lsm6dso32_sensor.gyro

#Goes through every single sensor and checks the readings against the specified
     parameters
    if not (MINVALUE["Proximity"] <= proximity <= MAXVALUE["Proximity"]):
        print("Proximity check failed. Proximity: ", proximity)
        return False
    else:
        print("Proximity check... OK")

    if MINVALUE["Lux"] is not None and not (MINVALUE["Lux"] <= lux <= MAXVALUE["Lux
         "]):
        print("Light check failed. Light: %d lux" % lux)
        return False
    else:
        print("Light check... OK")

    if not (MINVALUE["Temperature"] <= temperature <= MAXVALUE["Temperature"]):
        print("Temperature check failed. Temperature: %0.1f C" % temperature)
        return False
    else:
        print("Temperature check... OK")

    if MINVALUE["Humidity"] is not None and humidity > MAXVALUE["Humidity"]:
        print("Humidity check failed. Humidity: %0.1f %%" % humidity)
        return False
    else:
        print("Humidity check... OK")

    if any(not (BASE_ACCELERATION[i] + MINVALUE["Acceleration"] <= axis <=
         BASE_ACCELERATION[i] + MAXVALUE["Acceleration"]) for i, axis in enumerate(
         acceleration)):
        print("Acceleration check failed. Acceleration: X:%.2f, Y: %.2f, Z: %.2f m/
             s^2" % acceleration)
        return False
    else:
        print("Acceleration check... OK")

    if any(not (MINVALUE["Gyro"] <= axis <= MAXVALUE["Gyro"]) for axis in gyro):
        print("Gyro check failed. Gyro X:%.2f, Y: %.2f, Z: %.2f radians/s" % gyro)
        return False
    else:
        print("Gyro check... OK")

    return True

def monitor_perimeter_events():
    try:
        zymkey.client.wait_for_perimeter_event(timeout_ms=1000)
        perim_status_str = ""
        idx = 0
        plst = zymkey.client.get_perimeter_detect_info()
```

```
        print("Perimeter breach detected! Zymkey would be deleted here!\n" +
            perim_status_str)

    except zymkey.exceptions.ZymkeyTimeoutError:
        print("No perimeter breach detected.")

    zymkey.client.clear_perimeter_detect_info()

while True:
    if not check_sensors():
        monitor_perimeter_events()
        break
    time.sleep(2)
```

It is important to note that during testing, the parameter "*action-self-destruct*" was set to false to prevent the Zymkey from actually being deleted. In a real world scenario, this parameter would be set to true, enabling the Zymkey to be destroyed upon suspicious sensor readings.

Chapter 4

Results

This chapter presents the outcomes of the project execution and outlines what the project group has accomplished. It details the findings, the results, and the achievements up to the deadline. Furthermore, this chapter provides an objective analysis of the project's performance against the initial goals and expectations, offering insights into the effectiveness of the methodologies employed.

4.1 Flex PCB

The design of both PCBs ended up as shown in Figure 4.1 and Figure 4.2. The holes in the PCB align with the holes in the enclosure, as shown in Figure 4.3. The copper traces have a 3 mm spacing, with a thickness of 0.2 mm. Both PCBs are single-layer for cost efficiency. The flex PCB for the box will be folded, with a expected bend radius of 1 mm. With the PCB thickness of 90 µm, it complies with the IPC-2223 standard for bend radius, [43]. Additionally, the thickness of the PCB makes it easy to break, which is suitable for tamper detection. The PCBs have pads at each end to allow wires to be soldered and attached to the protoboard. Detailed physical specifications can be seen in Attachment 1.

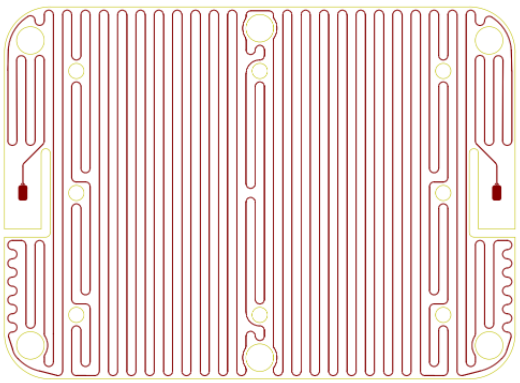


Figure 4.1: The PCB design for the lid.

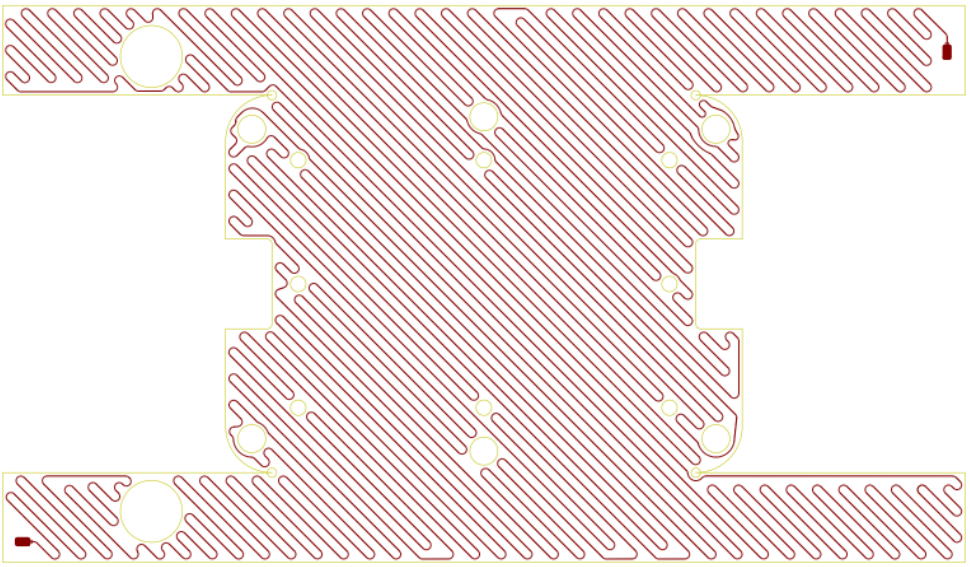


Figure 4.2: The PCB design for the box.

Figure 4.3: Holes in the enclosure for mounting protoboard and sealing the enclosure.

4.2 Protoboard & sensors

Sensors are soldered onto the protoboard as shown in Figure 4.4. SDA, SCL, 3V and GND have been extracted from the board and connected. Positioned at the top left is the gyro & accelerometer, while the light & proximity sensor is located towards the right middle. Found below is the temperature & humidity sensor. The decision for placing the sensor in this area was due to the fact that it was the closest to the Raspberry Pi. Extra space is available on the protoboard for

integrating additional sensors, if needed for future expansions.

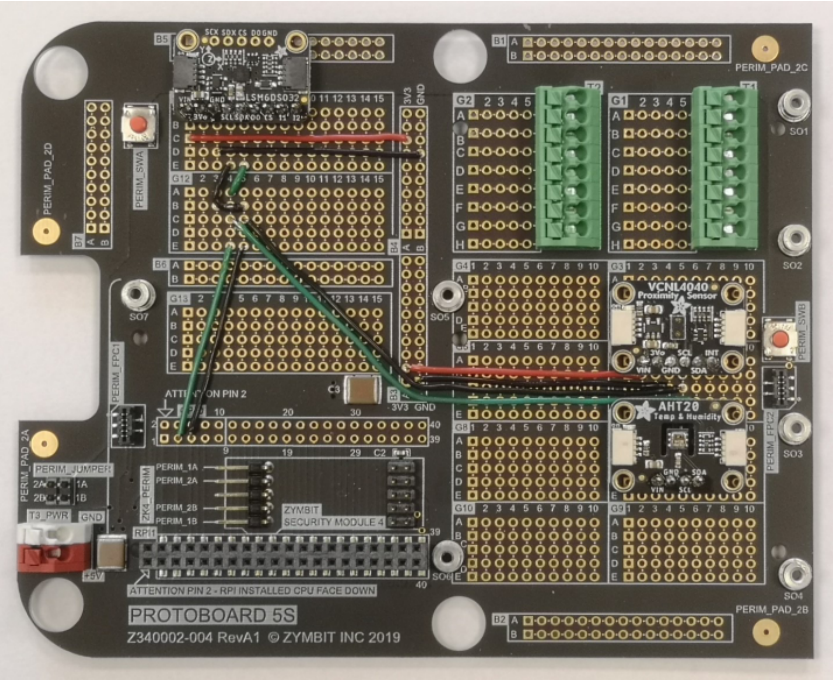


Figure 4.4: Components soldered to the board.

4.3 Sensor trigger levels

Table 4.1 outlines the sensor trigger levels for key deletion by the Zymkey, established from extensive testing that monitored the sensor readings within the enclosure. These values are carefully set to minimize the occurrence of false positives.

Table 4.1: Trigger levels for sensors.

Sensor	High trigger value	Low trigger value
Proximity	61	50
Lux	6	N/A
Temperature (°C)	32	15
Humidity (%)	45	N/A
Acceleration X (m/s^2)	1	-1
Acceleration Y (m/s^2)	1	-1
Acceleration Z (m/s^2)	1	-1
Gyro X (radians/s)	0.1	-0.1
Gyro Y (radians/s)	0.1	-0.1

Continued on next page

Table 4.1 continued from previous page

Sensortype	High trigger value	Low trigger value
Gyro Z (radians/s)	0.1	-0.1

The sensor trigger levels have been carefully chosen based on the typical behavior observed within the enclosure. Proximity readings were consistent at 55, whilst the lux levels were consistent at around 2. A small increase from that in trigger ensures detection of light exposure that deviates from the norm, and a potential lift of the lid. Accelerometer readings for the X, Y, and Z axes fluctuate slightly, with a maximum shift of ±0.5, indicating sensitivity to minor movements. Therefore, a threshold of ±1 has been set to minimize the potential for false positives from these slight displacements. The sensor consistently reads a value close to 9.81 m/s^2, which indicates gravitational acceleration. Depending on whether the device is mounted on a wall or placed on a flat surface, a different axis may align with the direction of gravity. Therefore, the trigger is set to react to a change of ±1 in the value of each axis, rather than it reaching the value of ±1 m/s^2. The gyroscope remained stable, often at 0, with occasional minor shifts to 0.01, hence the choice of 0.1 as a trigger. The temperature inside the enclosure showed a gradual increase, averaging at 26°C and not exceeding 28°C during testing. A low trigger value of 15°C accounts for possible drops due to door openings or other factors, especially if positioned near entryways which might lead to false triggers. Humidity steadily declined, not surpassing 39%, setting a higher trigger point at 45% to detect any unusual increase that might indicate tampering.

4.4 Circumventing each sensor

In the enclosure, the sensors can potentially be circumvented through specific techniques. For the proximity sensor, which measures the distance between the lid of the enclosure and itself, an attacker might simulate the lid being closed by placing an object that maintains the same distance from the sensor as the lid would. Since the enclosure will be indoors, the lux sensor can be bypassed by keeping the room or sensor in a dark environment. The temperature and humidity sensors will only be triggered by the use of thermal tools or freezing tools. The accelerometer and gyroscope will also not be triggered unless the enclosure is moved while mounted. Circumventing all the sensors will be tough as one would also need to bypass the tamper switches and the flex PCB.

4.5 Secure communication

The project group have successfully developed and implemented scripts and configuration files that help establish a secure communication channel between two Raspberry Pis. This setup allows the devices to exchange confidential data securely, safeguarding it against tampering during both transit and storage. While

there is still room for improvement, the primary objective of creating a reliable and secure communication method has been successfully achieved. The successful implementation of the design and the safety measures that have been put in place are shown in the following subsections.

4.5.1 Communication work flow

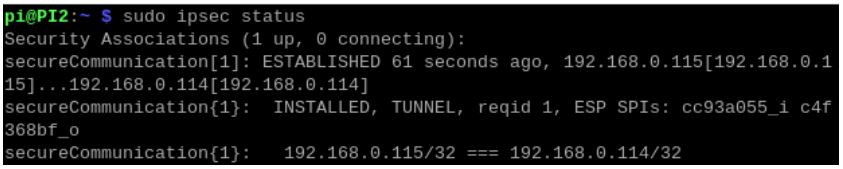


Figure 4.5: IPSec VPN tunnel status.

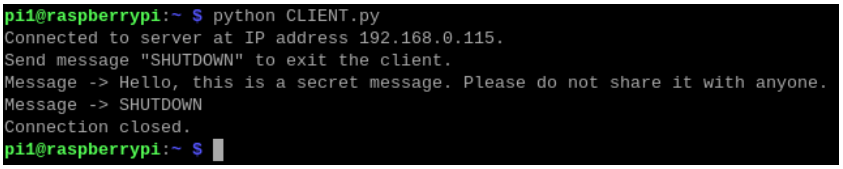


Figure 4.6: Client Raspberry Pi connecting to the server.

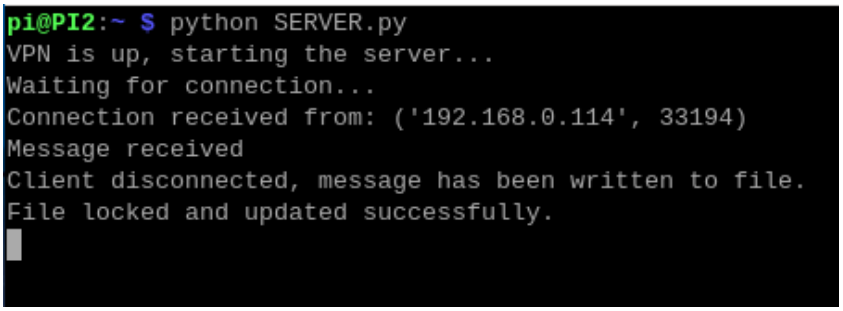


Figure 4.7: Server Raspberry Pi receiving messages from the client.

Figure 4.6 and Figure 4.7 showcase how the communication scripts work in a practical scenario. Based on the fact that the IPsec tunnel is up and running as shown in Figure 4.5, the server can be successfully started and goes on to listen for incoming messages. Once a client connects to the server and sends a message, notifications confirming these actions are shown. The received data is consequently encrypted and stored in its intended folder.

Once started, the server runs constantly, allowing for regular connections from the client, permitting multiple conversations to be started. This goes on until the server owner decides to manually stop the server by either typing in "SHUTDOWN"

into the console window, or in the case that a security measure is breached, which in turn stops the server automatically.

4.5.2 Secure storage of data

Figure 4.8: Encrypted message stored on the server Raspberry Pi.

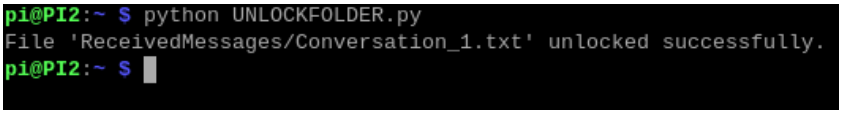


Figure 4.9: Unlocking of encrypted data.

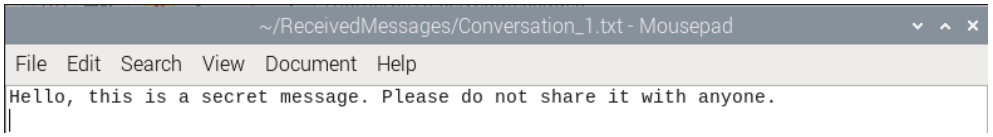


Figure 4.10: Decrypted version of the message stored on the server Raspberry Pi.

Figure 4.8 and Figure 4.10 illustrate the successful encryption of messages received during the client-server communication session. The encryption and decryption scripts are used to modify the state of the "*ReceivedMessages*" folder to either lock the data, securing it in the event of device theft, or unlock it to retrieve the plain-text data when necessary.

It is important to note though, that having a decryption script stored locally, with no access controls can lead to security risks. These risks and potential remedies to it are discussed in further detail in chapter 5.

4.5.3 Server-IPsec dependancy

Figure 4.11 and Figure 4.12 illustrate the servers operational dependency on a secure IPsec tunnel, ensuring that the client and server communication remains safe and untampered with. By mandating that the server only runs when a secure channel is established, the risk of data transmission over insecure networks is mitigated.

```
pi@PI2:~ $ python SERVER.py
VPN is down, server cannot be started.
pi@PI2:~ $
```

Figure 4.11: Check for a IPsec tunnel that allows the server to be started.

```
pi@PI2:~ $ python SERVER.py
VPN is up, starting the server...
Waiting for connection...

VPN is down, shutting down the server...
pi@PI2:~ $
```

Figure 4.12: Continuous IPsec tunnel monitoring with automatic server shutdown if unavailable.

4.5.4 IPsec-Zymkey dependancy

Figure 4.13 demonstrates the successful deployment of the enhanced security measures for the IPsec tunnel. By introducing continuous breach monitoring, the project group ensured that any attempt at interference immediately brings down the tunnel. This security measure significantly strengthens the integrity and confidentiality of the communication workflow.

```
VPN tunnel is currently up.
VPN tunnel is currently up.
deleting IKE_SA secureCommunication[1] between 192.168.0.115[192.168.0.115]...19
2.168.0.114[192.168.0.114]
sending DELETE for IKE_SA secureCommunication[1]
generating INFORMATIONAL request 2 [ D ]
sending packet: from 192.168.0.115[4500] to 192.168.0.114[4500] (76 bytes)
received packet: from 192.168.0.114[4500] to 192.168.0.115[4500] (76 bytes)
parsed INFORMATIONAL response 2 [ ]
IKE_SA deleted
IKE_SA [1] closed successfully
Breach detected! VPN tunnel deactivated.
```

Figure 4.13: Continuous breach monitoring that brings down the IPsec tunnel.

As seen in Figure 4.13, once a breach is detected, the IPsec tunnel is automatically brought down, as indicated by the "*VPN Tunnel deactived*" line. This in turn, would also shut down the server if it were up.

4.5.5 Zymkey - Sensor dependancy

Figure 4.14 demonstrates how the sensor check script works in a practical scenario. Here, once the lid of the enclosure is lifted, the proximity sensors detect an irregularity, which in turn would cause the Zymkey to be deleted. To illustrate how this would work in a different scenario, imagine the device being stolen. In order

to be stolen, the device would have to move from its original location, meaning that the movement sensors would trigger, causing the Zymkey to be deleted. The attacker would thus not be able decrypt the message contents even if they were able to get a hold of the device itself.

```
pi@PI2:~/FINALSCRIPTS $ python ZymkeyDestruct_SensorCheck.py

Proximity check... OK
Light check... OK
Temperature check... OK
Humidity check... OK
Acceleration check... OK
Gyro check... OK

Proximity check... OK
Light check... OK
Temperature check... OK
Humidity check... OK
Acceleration check... OK
Gyro check... OK

Proximity check failed. Proximity:  1
Perimeter breach detected! Zymkey would be deleted here!
```

Figure 4.14: Continuous sensor monitoring with automatic Zymkey deletion.

Chapter 5

Discussion & conclusion

The main goal of this thesis was to create an effective alternative to many existing anti-tampering enclosures. This was achieved by adding multiple security measures to a kit provided by Zymbit and then evaluating the viability of these solutions in a finished product. This chapter will discuss the proposed solutions and then conclude the work of this thesis.

5.1 Hardware discussion

The purpose of the additional hardware was to increase the amount of methods to detect attacks targeted at the device. Several sensors were added, along with a flex PCB designed to detect any physical attacks to the enclosure.

The flex PCB is capable of detecting almost any attacks on the enclosure. However, a flex PCB is not necessarily specialized for tamper detection compared to existing anti-tamper foils. The detection works only on a closed circuit, which an intruder could potentially circumvent by accessing the copper trace and shorting the circuit to maintain the loop. Reducing the spacing from 3 mm could make it even harder for an intruder to bypass. This issue could be addressed by using a specialized anti-tamper foil and a detection system that monitors total resistance. Such a system could detect tampering through sudden changes in circuit resistance. However, it is unknown if the Zymkey can continuously monitor resistance. An additional device to monitor the circuit might be necessary.

Another issue discovered is that the mounting holes used to secure the enclosure are not covered by the flex PCB (Figure 4.3). A highly advanced intruder could exploit this weakness by creating holes in these specific areas. A possible solution to this problem would be to design a new enclosure that does not require screws for mounting or to ensure that the mounting does not interfere with the flex PCB layout. Even if an intruder manages to bypass this, they would still need to circumvent each sensor and ensure the tamper switches remain pressed down.

The sensors used in the enclosure each have a fixed I^2C address. Therefore, adding more of the same type of sensor to cover a larger detection area is currently not possible on the same bus. A potential solution to this issue is to implement an I^2C multiplexer. The TCA9548A I^2C Multiplexer, identified by the group, allows for up to eight of the same type of sensor on the same data bus, providing a future option for expansion, [44].

The current enclosure is designed for indoor use. The operating temperature range for the Raspberry Pi is from 0 to 50°C. If the enclosure is to be used outdoors, this temperature range must be considered. It can be used in climates that do not exceed this range, or heating/cooling mechanisms can be added to ensure the Raspberry Pi remains within the acceptable temperature range.

5.2 Software discussion

While the scripts enable communication between the two Raspberry Pis and ensure secure storage of message contents, the setup can be enhanced by adding additional layers of security checks and integrating more hardware components.

One major vulnerability is that of having a single point of failure when it comes to encrypting and decrypting data-at-rest. In a scenario where the Zymkey malfunctions, the locked data will no longer be accessible, given that the encryption of data at rest is done solely with the use of the Zymkey and requires it to be functional to be deciphered. To improve on this aspect of the system, multiple hardware security modules could be used to manage the encryption and decryption processes in parallel. Additionally, establishing a routine for regular backups of deciphered data to be stored a separate, secure location would ensure data recoverability, even in the event of hardware failure.

The self-destruction script which uses sensor data to destroy the Zymkey is also susceptible to false positives from faulty sensors. In a scenario where one of these sensors malfunction or give a wrong reading, an unwarranted destruction of the Zymkey will follow. This, in addition to the fact that the encryption and communication system is reliant on the Zymkey, means that a faulty sensor could brick the entire system. To address this issue, multiple sensors can be used to combine readings and confirm anomalies before taking drastic actions such as wiping the Zymkey contents.

Storing a decryption script locally on the server machine, presents a significant security risk as well. If someone were to bypass the various security mechanisms and gain access to the server Raspberry Pi, they could use the decryption script to access the messages that the device holds. A practical solution to this, would be to implement access control measures such as Multi-factor Authentication (MFA). This approach would not only mitigate the risk of unauthorized external access,

but also safeguard against potential insider threats. Employees or insiders with inherent access to the machine would require additional authentication to decrypt data, significantly reducing the likelihood of sensitive information being improperly accessed or disclosed.

The security of the Raspberry Pis themselves must also be considered, as these devices lack inherent security measures. While the intended purpose of these machines is to act as client or server, users operating them can nonetheless use them to access the internet and potentially catch malware that could cause interception of the secure communication flow or other types of corruption. To harden and safeguard the Raspberry Pis against malware and other security threats, unnecessary services on these devices should be disabled. Strict firewall rules and anti-virus/anti-malware solutions for further protection should also be employed.

Lastly, it is important to consider that even a functioning device is not future proof. With the development of quantum-computers, encryption algorithms that are considered secure today, may not necessarily be as secure in the future. This must be taken into account when deploying the device for long-term operation.

5.3 Conclusion

In the realm of securing data integrity, absolute security is unattainable, due to the ever-present improvement of attack techniques and the development of new tools that could potentially harm a system. While the creation of a functional device marks a significant milestone, it is important to acknowledge that there is always room for improvement. In its current state, the device provides a solid foundation, but with the proposed future enhancements, the project group can envision the product becoming a top-notch security solution.

Bibliography

[1] SOG-IS, *Application of attack potential to hardware devices with security boxes*, Available online, version 3.1, Online; accessed 17-April-2024, 2023. [Online]. Available: https://www.sogis.eu/documents/cc/domains/hardware_devices/JIL-Application-of-Attack-Potential-to-Hardware-Devices-with-Security-Boxes-v3.1.pdf.

[2] NordVPN. 'Cold boot attack.' Online; accessed 17-April-2024. (2024), [Online]. Available: https://nordvpn.com/no/cybersecurity/glossary/cold-boot-attack/.

[3] Sparkfun, *I2c*, Online; accessed 19-April-2024, 2024. [Online]. Available: https://learn.sparkfun.com/tutorials/i2c/all.

[4] C. Nelson. 'Working with i2c devices.' Online; accessed 19-April-2024. (2024), [Online]. Available: https://learn.adafruit.com/working-with-i2c-devices/address-conflicts.

[5] Adafruit, *Adafruit vcnl4040 proximity and lux sensor - stemma qt / qwiic*, Online; accessed 20-April-2024, 2024. [Online]. Available: https://www.adafruit.com/product/4161.

[6] W. contributors, *False positives and false negatives — Wikipedia, the free encyclopedia*, Online; accessed 20-April-2024, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=False_positives_and_false_negatives&oldid=1218727934.

[7] Zymbit, *Zymkey4 - hsm for raspberry pi*, Online; accessed 21-April-2024, 2024. [Online]. Available: https://store.zymbit.com/products/zymkey4i.

[8] Opensource, *What is a raspberry pi?* Online; accessed 21-April-2024, 2024. [Online]. Available: https://opensource.com/resources/raspberry-pi.

[9] R. Pi, *Raspberry pi technical specs*, Available online, Online; accessed 22-April-2024. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

[10] IBM, *What is cryptography?* Online; accessed 20-April-2024. [Online]. Available: https://www.ibm.com/topics/cryptography.

[11] CloudFare, *What is encryption?* Online; accessed 20-April-2024. [Online]. Available: https://www.cloudflare.com/learning/ssl/what-is-encryption/.

[12] P. S.
bibinitperiod D. M. Turner, *Symmetric key encryption - why, where and how it's used in banking*, Online; accessed 21-April-2024, 2020. [Online]. Available: https://www.cryptomathic.com/news-events/blog/symmetric-key-encryption-why-where-and-how-its-used-in-banking.

[13] WttW, *Cryptography with alice and bob*, Online; accessed 4-May-2024, 2014. [Online]. Available: https://wordtothewise.com/2014/09/cryptography-alice-bob/.

[14] B. Buchanan, *Everything you want to know about aes, but were afraid to ask*, Online; accessed 28-April-2024, 2022. [Online]. Available: https://medium.com/asecuritysite-when-bob-met-alice/everything-you-want-to-know-about-aes-but-were-afraid-to-ask-b46165410ea8.

[15] W. contributors, *Public-key cryptography*, Online; accessed 21-April-2024, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Public-key_cryptography.

[16] TechNotes, *Asymmetric key cipher*, Online; accessed 28-April-2024, 2019. [Online]. Available: https://www.binderror.com/blog/cryptography_104/.

[17] N. Lord, *Data protection: Data in transit vs. data at rest*, Online; accessed 25-April-2024, 2023. [Online]. Available: https://www.digitalguardian.com/blog/data-protection-data-in-transit-vs-data-at-rest.

[18] S. G. .-. NORDVPN, *What is the sha-256 algorithm, and how does it work?* Online; accessed 25-April-2024, 2023. [Online]. Available: https://nordvpn.com/blog/sha-256/.

[19] Amazon, *What is osi model?* Online; accessed 20-April-2024, 2024. [Online]. Available: https://aws.amazon.com/what-is/osi-model/.

[20] CloudFare, *What is the osi model?* Online; accessed 28-April-2024. [Online]. Available: https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/.

[21] M. Rossevelt, *Understanding data encryption at the data link layer of the osi model*, Online; accessed 4-May-2024, 2023. [Online]. Available: https://www.newsoftwares.net/blog/data-encryption-at-the-data-link-layer-of-the-osi-model/.

[22] E. Robin, *A comprehensive guide to understanding which osi layer encrypts data*, Online; accessed 30-April-2024, 2023. [Online]. Available: https://www.newsoftwares.net/blog/guide-to-understanding-which-osi-layer-encrypts-data/#Advantages_and_Disadvantages_of_Network_Link_Layer_Encryption.

[23] CloudFare, *What is ipsec? | how ipsec vpns work*, Online; accessed 30-April-2024. [Online]. Available: https://www.cloudflare.com/learning/network-layer/what-is-ipsec/.

[24] S. Zivuku, *Ipsec: The complete guide to how it works and how to use it*, Online; accessed 30-April-2024, 2021. [Online]. Available: https://www.twingate.com/blog/ipsec.

[25] Pomerium. 'Ipsec: Tunnel mode and transport mode.' Online; accessed 21-April-2024. (2024), [Online]. Available: https://www.pomerium.com/glossary/ipsec-tunnel-mode-and-transport-mode/#:~:text=Tunnel%20Mode%20provides%20end%2Dto,host%2Dto%2Dhost%20communication..

[26] PaloAlto, *Internet key exchange (ike) for vpn*, Online; accessed 6-May-2024. [Online]. Available: https://docs.paloaltonetworks.com/network-security/ipsec-vpn/administration/ipsec-vpn-basics/internet-key-exchange-ike-for-vpn.

[27] C. Press, *How ipsec works*, Online; accessed 13-May-2024, 2002. [Online]. Available: https://www.ciscopress.com/articles/article.asp?p=24833&seqNum=6.

[28] O. Cassetto, *Cybersecurity threats: Everything you need to know*, Online; accessed 2-May-2024, 2023. [Online]. Available: https://www.exabeam.com/information-security/cyber-security-threat/.

[29] Imperva, *Man in the middle (mitm) attack*, Online; accessed 7-May-2024. [Online]. Available: https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/.

[30] Zymbit, *Protokit5*, Online; accessed 9-May-2024, 2024. [Online]. Available: https://docs.zymbit.com/tutorials/protokit/.

[31] NCAB, *Pcb design guidelines*, accessed 28-February-2024. [Online]. Available: https://www.ncabgroup.com/pcb-design-guidelines/#download.

[32] Adafruit, *Adafruit aht20 - temperature & humidity sensor breakout board - stemma qt / qwiic*, Online; accessed 24-April-2024, 2024. [Online]. Available: https://www.adafruit.com/product/4566.

[33] K. R.
bibinitperiod B. Rubell, *Adafruit aht20 temperature & humidity sensor - python & circuitpython*, Online; accessed 25-April-2024, 2024. [Online]. Available: https://learn.adafruit.com/adafruit-aht20/python-circuitpython.

[34] Adafruit, *Adafruit lsm6dso32 6-dof accelerometer and gyroscope - stemma qt / qwiic*, Online; accessed 24-April-2024, 2024. [Online]. Available: https://www.adafruit.com/product/4692.

[35] Zymbit, *Quickstart - zymkey4*, Online; accessed 15-May-2024. [Online]. Available: https://docs.zymbit.com/getting-started/zymkey4/quickstart/.

[36] strongSwan, *Strongswan*, Online; accessed 17-May-2024. [Online]. Available: https://www.strongswan.org/.

[37] Yubico, *What is certificate-based authentication?* Online; accessed 17-May-2024. [Online]. Available: https://www.yubico.com/resources/glossary/what-is-certificate-based-authentication/.

[38] Unknown, *Unknown*, Online; accessed 3-March-2024, 2024. [Online]. Available: https://www.dropbox.com/s/jva8mfb0h1ydn5v/server.py?e=2&dl=0.

[39] Unknown, *Unknown*, Online; accessed 3-March-2024, 2024. [Online]. Available: https://www.dropbox.com/s/irpzc0w3acswr2a/client.py?e=1&dl=0.

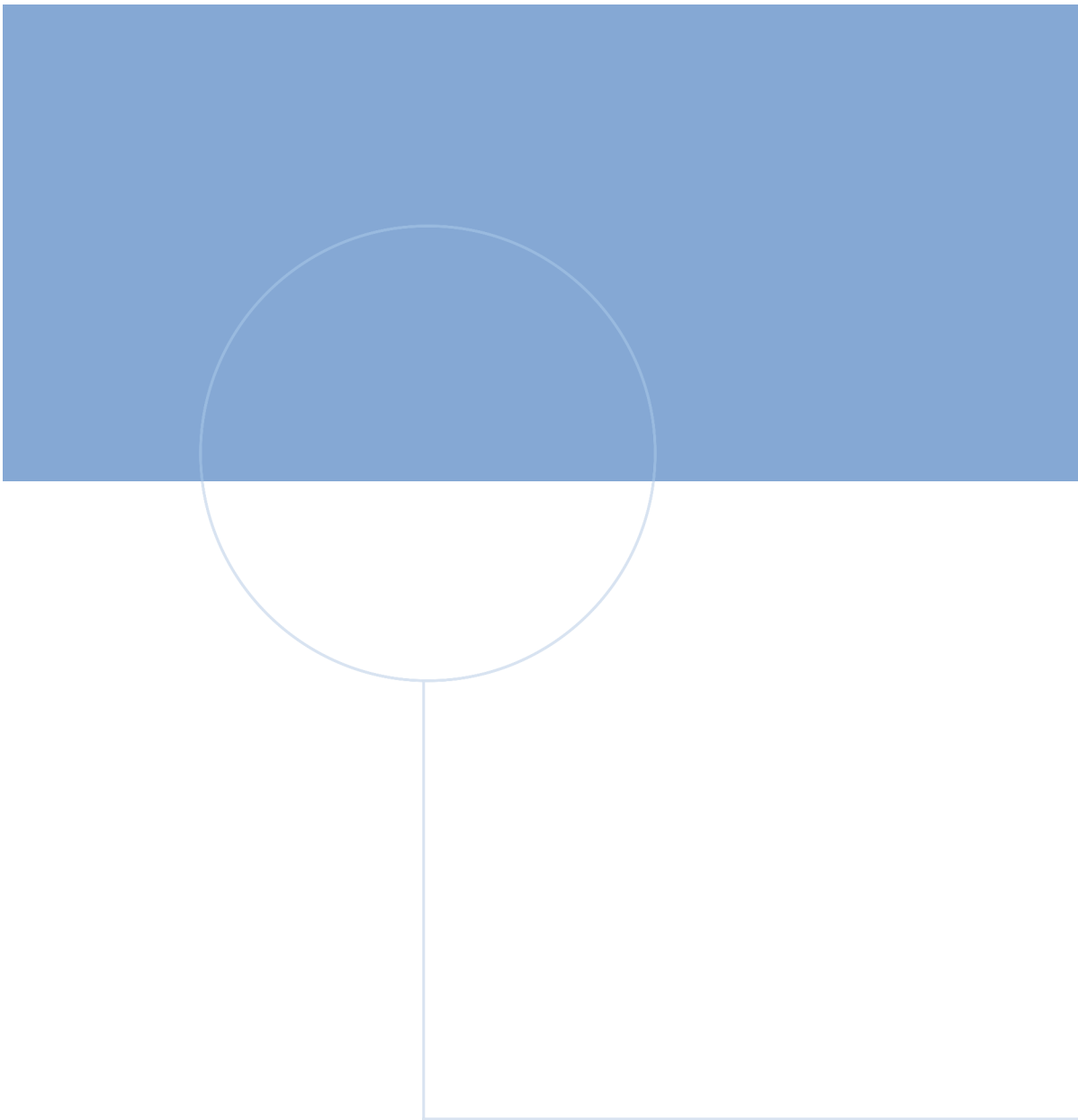
[40] B. Siepert, *Lsm6dsox, ism330dhc, & lsm6dso32 6 dof imus*, Online; accessed 16-May-2024, 2019. [Online]. Available: https://learn.adafruit.com/lsm6dsox-and-ism330dhc-6-dof-imu/python-circuitpython.

[41] K. Rembor, *Adafruit aht20 temperature & humidity sensor*, Online; accessed 16-May-2024, 2020. [Online]. Available: https://learn.adafruit.com/adafruit-aht20/python-circuitpython.

[42] B. Siepert, *Adafruit vcnl4040 proximity sensor*, Online; accessed 16-May-2024, 2019. [Online]. Available: https://learn.adafruit.com/adafruit-vcnl4040-proximity-sensor/python-circuitpython.

[43] MADPCB, *Bend radius*, Online; accessed 10-February-2024. [Online]. Available: https://madpcb.com/glossary/bend-radius/.

[44] Adafruit, *Tca9548a i2c multiplexer*, Online; accessed 20-May-2024. [Online]. Available: https://www.adafruit.com/product/2717.



NTNU

Kunnskap for en bedre verden